UNIVERSIDAD LAICA "ELOY ALFARO" DE MANABÍ

FACULTAD DE CIENCIAS INFORMÁTICAS

CARRERA DE INGENIERÍA EN SISTEMAS

DESARROLLO E IMPLEMENTACIÓN DE UN SISTEMA ERP UTILIZANDO LA PLATAFORMA ODOO, CON INTEGRACIÓN DE SOCIAL MEDIA Y LEGISLACIÓN ECUATORIANA, PARA LA AUTOMATIZACIÓN, CONTROL DE LOS PROCESOS DE NEGOCIOS EN LA EMPRESA "TODO CRIOLLO" DEL CANTÓN MANTA.

TRABAJO DE TITULACIÓN MODALIDAD PROYECTO INTEGRADOR, PREVIO A LA OBTENCIÓN DEL TÍTULO DE INGENIERA/O EN SISTEMAS

AUTORES:

VILLACRESES LUCAS JEFFRI REYNALDO

CEDEÑO CEDEÑO BRYAN ALEXANDER

DIRECTOR DE TEMA:

Ing. José Arteaga Vera, MG

MANTA – ECUADOR

2017

UNIVERSIDAD LAICA "ELOY ALFARO" DE MANABÍ
Creada el 13 de noviembre de 1985 mediante Decreto Ley No.10, publicado en el Registro Oficial No. 313
FACULTAD DE CIENCIAS INFORMÁTICAS
Creada, Resolución H. Consejo Universitario del 11 de Julio del 2001

CERTIFICACIÓN:

En calidad de Docente de la Facultad de Ciencias Informáticas de la Universidad Laica Eloy Alfaro de Manabí, certifico:

Haber dirigido y revisado el Trabajo de Titulación Modalidad Proyecto Integrador: **"DESARROLLO E IMPLEMENTACIÓN DE UN SISTEMA ERP UTILIZANDO LA PLATAFORMA ODOO, CON INTEGRACIÓN DE SOCIAL MEDIA Y LEGISLACIÓN ECUATORIANA, PARA LA AUTOMATIZACIÓN, CONTROL DE LOS PROCESOS DE NEGOCIOS EN LA EMPRESA "TODO CRIOLLO" DEL CANTÓN MANTA."**, proyecto que cumple con los requisitos que exige la Guía Metodológica de Titulación de la Institución y el instructivo normativo para trabajos de titulación de la carrera Ingeniería en Sistemas de la Facultad de Ciencias Informáticas y, reúne los méritos suficientes para ser sometido a la evaluación del jurado examinador que designen las autoridades.

La autoría del tema desarrollado, corresponde a los señores **CEDEÑO CEDEÑO BRYAN ALEXANDER** y **VILLACRESES LUCAS JEFFRI REYNALDO**, estudiantes con estudios concluidos en la carrera Ingeniería en Sistemas, período académico 2015-2016, quien se encuentra apto para la defensa.

Particular que certifico para los fines consiguientes, salvo disposición de Ley en contrario.

Lo certifico:

Ing. José Arteaga Vera, Mg.
Docente Facultad de Ciencias Informáticas
Universidad Laica "Eloy Alfaro" de Manabí

Manta, 02 de agosto de 2017.

Formando científica, tecnológica y culturalmente a los futuros profesionales en las ciencias informáticas

TRABAJO DE TITULACIÓN MODALIDAD PROYECTO INTEGRADOR, PREVIO A LA OBTENCIÓN DEL TÍTULO DE: INGENIERO EN SISTEMAS

"DESARROLLO E IMPLEMENTACIÓN DE UN SISTEMA ERP UTILIZANDO LA PLATAFORMA ODOO, CON INTEGRACIÓN DE SOCIAL MEDIA Y LEGISLACIÓN ECUATORIANA, PARA LA AUTOMATIZACIÓN, CONTROL DE LOS PROCESOS DE NEGOCIOS EN LA EMPRESA "TODO CRIOLLO" DEL CANTÓN MANTA."

Tribunal examinador que declara APROBADO el Grado de INGENIERO EN SISTEMAS, de los señores: VILLACRESES LUCAS JEFFRI REYNALDO-CEDEÑO CEDEÑO BRYAN ALEXANDER

Ing. Jorge Pincay Ponce, Mg. _____

Ing. Rubén Solórzano Cadena, Mg. _____

Ing. Jimmy Moreira Mero _____

Manta, 25 de agosto de 2017

DECLARACIÓN EXPRESA DE AUTORÍA

Nosotros, Villacreses Lucas Jeffri Reynaldo y Cedeño Cedeño Bryan Alexander, en calidad de autores del trabajo de titulación: **"Desarrollo e implementación de un sistema erp utilizando la plataforma Odoo, con integración de social media y legislación ecuatoriana, para la automatización, control de los procesos de negocios en la empresa "Todo Criollo" del cantón Manta"**, autorizamos a la Universidad Laica "Eloy Alfaro" de Manabí, hacer uso completo o parcial del contenido de este trabajo de titulación del cual somos responsables, con fines estrictamente académicos o de investigación.

Los derechos que como autores nos corresponden, con excepción de la presente autorización, seguirán vigentes a nuestro favor, de conformidad con lo establecido en los artículos 5, 6, 8, 19 y demás artículos pertinentes de la Ley de Propiedad Intelectual y su Reglamento.

Asimismo, autorizamos a la Universidad Laica "Eloy Alfaro" de Manabí para que realice la digitalización y publicación de nuestro trabajo de titulación en el repositorio virtual, en conformidad a lo establecido en el Art. 144 de la Ley Orgánica de Educación Superior.

Manta, 02 Agosto de 2017

Villacreses Lucas Jeffri Reynaldo
CI: 131189431-3
Telf: 0990905771
Email: **e1311894313@live.uleam.edu.ec**

Cedeño Cedeño Bryan Alexander
CI: 131306167-1
Telf: 0980781987
Email: **e1313061671@live.uleam.edu.ec**

iv

Dedicatoria

Con amor sublime a los seres queridos, que constituyen razón de vuestra superación.

A la Universidad Laica “Eloy Alfaro de Manabí”, profesores y Tutor, esperando que este modesto aporte sirva para enriquecer a futuras generaciones.

Plasmamos el firme compromiso en demostrarles fructíferos resultados de nuestros logros en beneficio de la sociedad y el país.

Dedicatoria Autor: Bryan Cedeño Cedeño

De manera muy especial a Dios, por enseñarme el camino correcto de la vida, porque guiado de su mano hoy arribo a un puerto, mañana comenzaré un nuevo trayecto.

A mis padres, José Cedeño Pincay y Nory Cedeño Holguín por ser dignos ejemplos de superación y entrega, brindándome estabilidad emocional, económica y sentimental, porque gracias a ustedes, hoy puedo ver alcanzada mi meta,

A mis hermanos, Andy y Gema por haber fomentado en mí el deseo de superación y el anhelo de triunfo en la vida.

A mi novia Gema Aveiga por el apoyo incondicional en todos los momentos, por ser el motor e inspiración en mis días en el cumplimiento de mis metas.

Mis compañeros y amigos Jeffri, Jorge, Adrián, Ronny, Daniel, Junior, Yimer, Anderson, por su presencia y apoyo ante cualquier situación y convertirse así en parte de mi familia.

De manera muy especial al Sr. Rafael Palacios y familia por la confianza y consideración plasmada en nosotros, el apoyo brindado en todos los ámbitos para la consecución de este proyecto.

A la Universidad Laica "Eloy Alfaro" de Manabí, por su loable labor, que contribuye a la construcción de profesionales íntegros, creativos y competentes.

Al Director de Tesis: Ing. José Cristóbal Arteaga Vera por sus aportes que sin duda han sido transcendental para el éxito del presente trabajo de investigación.

Agradecimiento- Autor: Villacreses Lucas Jeffri

A mis padres Genny y Reynaldo por el apoyo incondicional todos estos años de estudios, por esa paciencia y confianza puesta en mi para el cumplimiento de mis metas.

Mi hermana y seres queridos, que formaron parte de este esfuerzo, brindándome apoyo moral y económico.

A mi novia Emily por ayudarme a no decaer en el estudio y brindarme su apoyo para seguir adelante en mis metas.

Mis amigos Adrián, Jorge, Bryan, Ronny, Junior, Daniel, Yimer, Anderson, por llegar a forma parte de mi familia, estando siempre a mi lado incondicionalmente apoyando en cada momento y en toda situación.

A nuestros maestros que inculcaron sus sabios conocimientos durante la trayectoria educativa.

Singular mención a los integrantes de "Todo Criollo" por el apoyo brindando para la ejecución del proyecto, en especial al Sr. Rafael Palacios por su confianza en nosotros y apoyo económico.

Índice

Resumen 1
Abstract 2
Introducción 3
Ubicación y contextualización de la investigación 5
Planteamiento del problema 6
Diagrama Causa-Efecto del Problema 8
Objetivos 9
Objetivo general 9
Objetivos específicos 9
Justificación 10
Capítulo 1 12
Marco teórico de la investigación 12
1.1. Introducción 12
1.2. Antecedentes de investigaciones 13
Los ERP en la actualidad. - 15
1.3. Definiciones Conceptuales 16
1.3.1. ERP 16
1.3.2. Social Media 17
1.3.2.1 SEO 17
1.3.2.2 SEM 18
1.3.2.3 AdWords 18
1.3.3. Odoo 19
1.3.3.1 Arquitectura 19

1.3.3.2. Lenguaje 22

1.3.3.3. Reportes QWeb Rml 22

1.3.3.4 PostgreSQL 23

1.3.3.5 XML 24

1.3.3.6 ORM 25

1.3.4. Metodologías Agiles 25

1.3.5. Scrum 26

1.3.5.1 Marco Técnico 26

1.3.5.2 Artefactos 28

1.3.5.3 Eventos 29

1.4. Fundamentación legal 32

1.5. Conclusiones relacionadas al marco teórico en referencia al tema de investigación 34

Capítulo 2 35

Diagnóstico 35

2.1. Introducción 35

2.2. Tipo(s) de investigación 36

2.2.1. Investigación Descriptiva. 36

2.3. Métodos de investigación 36

2.3.1. Método de Observación. 36

2.3.2. Método Análisis Síntesis. 37

2.3.2.1. Análisis: 37

2.3.2.2 Síntesis: 37

2.4 Herramientas de recolección de datos 37
2.4.1. Encuesta. 37

2.4.3 Observación 38

2.5. Fuentes de información de datos 38
2.5.1. Fuentes Primarias. 38

2.5.2. Fuentes Secundarias. 39

2.6. Instrumental operacional 39
2.6.1. Estructura y características de lo(s) instrumento(s) de recolección de datos 39

2.6.1.1. Encuesta 39

2.6.1.1.1. Características. 39

2.6.1.1.2. Estructura 39

2.6.1.3. Observación. 40

2.7. Estrategia operacional para la recolección y tabulación de datos 40
2.7.1. Plan para la recolección de datos. 41

2.7.2. Plan para la tabulación de datos. 42

2.7.3. Plan de análisis e interpretación de datos 43

2.8. Plan de muestreo 43
2.8.1. Población 43

2.8.2. Segmentación 44

2.8.3 Técnicas de muestreo 44

2.8.3.1. Muestreo no probabilístico 44

2.8.3.2. Muestreo por conveniencia. 45

2.8.3.3. Muestreo por cuota 45

2.8.4 Tamaño de la muestra 45

2.9 Presentación y análisis de los resultados 46

Capítulo 3 47

Diseño de la propuesta 47

3.1. Introducción 47

3.2. Descripción de la propuesta 48

3.2.1. Especificaciones técnicas 49

3.2.2. Alcance de la propuesta. 50

3.2.3. Recursos 51

3.2.3.1. Humano 51

3.2.3.2. Tecnológicos. 51

3.2.3.3. Presupuesto para la implementación. 52

3.2.4. Cronograma 53

3.2.5. Línea de tiempo 53

3.3. Etapas de la propuesta 54

3.3.1. Selección de la metodología de desarrollo 54

3.3.2. Bosquejo de la metodología Scrum 54

3.3.2.1. Definición del equipo de trabajo 54

3.3.2.2. Descripción de los objetivos del producto (Product Backlog) 55

3.3.2.3. Explicación de las iteraciones (Sprint Backlog) 55

3.3.2.4. Planeación de las iteraciones (Sprint) 57

3.3.2.5. Explicación de la retroalimentación. 58

3.3.3. Detalle de iteraciones 58

3.3.3.1. Sprint 1. Del 11 de abril al 21 de abril 58

3.3.3.2. Sprint 2 Del 24 de abril al 05 de mayo 64

3.3.3.3. Sprint 3. Del 08 de mayo al 02 de junio 71

3.3.3.4. Sprint 4. Del 05 de junio al 23 de junio 74

3.3.3.5. Sprint 5. Del 26 de junio al 07 de julio 82

Capitulo IV 86
Evaluación de resultados 86
4.1. Introducción 86
4.2. Seguimiento y Monitoreo de resultados 86
4.2.1. Evaluación de resultados por observación. 87

Conclusiones 93
Recomendaciones 94

Índice Tablas

Tabla 1 - Plan de Recolección de Datos 41
Tabla 2 - Plan de Tabulación de Datos 42
Tabla 3 – Plan de análisis e interpretación de datos 43
Tabla 4 - Población total de Manta 44
Tabla 5 - Segmentación de población de Manta 44
Tabla 6 - Recursos humanos 51
Tabla 7 - Recursos tecnológicos 51
Tabla 8 - Recursos económicos (Presupuesto) 52
Tabla 9 - Cronograma del proyecto 53
Tabla 10 - Lista de objetivos del producto 55
Tabla 11- Planeación de iteraciones 57
Tabla 12 - Historia de usuario 1 61
Tabla 13 - Listado de tablas de base de datos 68
Tabla 14 - Plan de capacitaciones 83
Tabla 15 - Detalle comparativo de Administración Contable 88
Tabla 16 - Detalle comparativo de proceso de Compras 89
Tabla 17 - Detalle comparativo de proceso de Ventas 91

Índice Gráficos e Ilustraciones

Ilustración 1 - Diagrama causa-efecto del problema 8
Ilustración 2 - Arquitectura Modelo Vista-Controlador 20
Ilustración 3 - Incrementos iterativo y continuo 28
Ilustración 4 - Increménte en Scrum 29
Ilustración 5 - Resumen Scrum 31
Ilustración 6 - Línea de tiempo del proyecto 53
Ilustración 7- Proceso de negocio - Inventario 59
Ilustración 8- Proceso de negocio - Contabilidad 59
Ilustración 9- Proceso de negocio - Ventas 60
Ilustración 10 - Proceso de negocio - Compra 60
Ilustración 11- Caso de uso- Generar código EAN 63
Ilustración 12 - Diagrama de secuencia - Generar código Ean 63
Ilustración 13 - Boceto Anexos Transaccionales 64
Ilustración 14 - Boceto de Opciones en Contabilidad 65
Ilustración 15 - Boceto de Ventas 65
Ilustración 16 - Boceto de Compras 66
Ilustración 17 - Boceto de Peso y Precio en Productos 66
Ilustración 18 - Boceto Exportar Listado 67
Ilustración 19 - Arquitectura Odoo 68
Ilustración 20 - Modelo entidad-relación de la base de datos 70
Ilustración 21 - Diseño de página web 73
Ilustración 22 - Modelo de red en instalaciones de Todo Criollo 75
Ilustración 23 - Logotipo de Hosting 76
Ilustración 24 - Instalación de Odoo 77
Ilustración 25 - Modulo de ventas 77
Ilustración 26 - Modulo de compras 78
Ilustración 27 - Modulo de Administración financiera 78

Ilustración 28 - Modulo Gestión de Inventarios 79
Ilustración 29 - Modulo Peso Precio 80
Ilustración 30 - Módulo Exportación Código EAN 80
Ilustración 31 - Modulo Contabilidad ecuatoriana 81
Ilustración 32 - Campaña de publicidad en Facebook 84
Ilustración 33 - Publicación de campaña en Facebook 84
Ilustración 34 - Resultados de campaña en Facebook 85
Ilustración 35 - Representación gráfica de resultados 85
Ilustración 36 - Cuadro comparativo de Administración Contable 88
Ilustración 37 - Grafico comparativo de proceso de Compras 90
Ilustración 38 - Grafico comparativo de proceso de Ventas 91
Ilustración 39 - Resultados pregunta 5 de encuesta 95

Resumen

Las tecnologías de la información hoy en día tienen mucha importancia en las Pymes como herramienta de toma de decisiones. Los sistemas de planificación de recursos empresariales como sistema de información y su uso en las empresas cada día va en aumento. Con este enfoque del uso primordial de la tecnología en la actualidad, nos acercamos a las empresas para ofrecer un cambio de mentalidad para el uso de un sistema ERP.

La herramienta ERP de elección fue Odoo 8.0, software de código fuente libre que nos permite modificar a conveniencia y nos ofrece una gama de infinitas posibilidades para los cambios que el cliente pueda requerir. La determinación del propietario de la empresa, se refleja en la implementación de un ERP, permitiéndole asumir el control de los procesos que van desde la compra de los productos, almacenamiento, contabilización, ventas e informes que manifieste la transparencia de los procesos del negocio y tomar decisiones de manera acertada.

Junto con la implementación del sistema, se ejecutará el desarrollo de una página web informativa, una aplicación móvil tipo catálogo para los clientes y el desarrollo de complementos para Odoo según las necesidades que se nos propongan, a manera de ofrecer los servicios de la empresa en unas de las plataformas más grandes de marketing en la actualidad como es la internet se realizara una página de Facebook para campañas publicitarias y la apertura de una cuenta en Google AdWords plataforma de marketing online. La propuesta se orienta al mejoramiento de los procesos, aumento de rentabilidad del negocio y satisfacción del cliente atendido.

Abstract

Information technologies today are very important in Pymes as a decision-making tool. Enterprise resource planning systems as an information system and their use in companies is increasing every day. With this focus on the primary use of technology today, we approach companies to offer a change of mentality for the use of an ERP system, resulting in the company "Todo Criollo" a commercial located in centric streets that opened us the doors for an implementation of several processes carried out roughly in many years of operation.

The ERP tool of choice was Odoo 8.0, free source code software that allows us to modify convenience and offers us a range of infinite possibilities for the changes that the client may require. The determination of the owner of the premises to want the implementation of an ERP is to be able to have control of the processes that go from the purchase of the products, storage, accounting, sales and reports that allow the visibility of the business and how to be able to improve. Along with the implementation of the system, the development of a web page will be executed, a mobile application type catalog for the clients and the development of complements for Odoo according to the needs that they propose us, in order to offer the services of the company in some of the largest marketing platforms today as it is the internet will make a Facebook page for advertising campaigns and opening an account in Google AdWords online marketing platform. The proposal is oriented to the improvement of the processes, increase of profitability of the business and satisfaction of the customer served.

Introducción

Son épocas de cambios globalizados en términos de tecnología y no podemos dejar pasar lo que estas herramientas ofrecen, como permitir una visibilidad completa de la compañía en cualquier ámbito, ya sea financiero o administrativo. Los sistemas de planificación empresarial conocidos como ERP lideran este ámbito, en cuestiones de toma de decisiones, ya que integra los principales procesos de la empresa en una sola plataforma, la optimización de procesos de negocios es una virtud de los sistemas de planificación empresarial que nos permite estar un paso delante de la competencia.

Dentro de la investigación en relación al tema, se encuentra el ámbito del desarrollo de la propuesta, el problema principal que nos induce a la implementación y como se logrará la solución de este por medio de los objetivos que se plantean.

El documento consta de 4 capítulos, dividido en marco conceptual, metodología, desarrollo y la evaluación de resultados, detallados de la siguiente manera:

El capítulo I, está enfocado en el marco teórico de la investigación, en el detalle de los conceptos de los temas más inusuales dentro de la investigación y que contribuyan al entendimiento del proyecto, una concepción de los lineamientos seguidos legalmente, según lo propone el gobierno en planes como el de comercio electrónico que nos direccionan para la culminación del proyecto.

En el capítulo II, se detallará la metodología de investigación usada como es la descriptiva, que nos proporciona herramientas para el entendimiento del problema y direccionamiento para la visualización de posibles soluciones para la empresa “Todo Criollo”. Explicando los métodos de investigación usados y las técnicas para recopilar la información, mismas que nos ayudan esclarecer sobre cuál es la población hacia donde debemos dirigir el estudio.

En el capítulo III, se desenvuelve la propuesta usando la metodología de desarrollo ágil scrum, partiendo desde la toma de requisitos, elaboración de diagramas de caso de uso, la estimación y creación de una pila del producto que nos sugiere la metodología, dividida en sprints o iteraciones con tareas diferentes en cada una con el fin de lograr un producto entregable al final de cada iteración. En cinco sprints de 10 a 20 días se logró el desarrollo de la implementación con un tiempo de 65 días hábiles.

El capítulo IV, está determinado por la evaluación de los resultados de la implementación, aplicando un método comparativo sobre los procesos que la empresa manejaba de forma manual y el aporte de las herramientas tecnológicas usadas, generando tablas y gráficos que nos ayudan a comprender los resultados estimados en el uso del sistema Odoo.

Ubicación y contextualización de la investigación

En la actualidad las tecnologías de la información y comunicación, son una parte esencial en muchas empresas alrededor del mundo agregando valor a las actividades operacionales y en general a la gestión empresarial, que permite competitividad, concentración y conocimiento sobre el negocio. Es de destacar que la implementación de algún sistema de información no garantiza resultados a corto plazo, ayuda a integrar partes del negocio para la visualización del flujo de diferentes procesos. La implantación de las Tics dentro del área comercial es un proceso complejo antes de que un sistema de información quede operacional, puede que ocurran varias fallas por la complejidad de los negocios y características únicas.

Dentro del ámbito de tecnológico de la investigación:

Existen herramientas para el área de aplicación tecnológica en las microempresas, los sistemas de planificación empresarial son en la actualidad una de las soluciones más usadas en temas informáticos, conocidos como ERP son sistemas integrados que permiten la revisión de los procesos requeridos por la empresa, estos son accesibles de manera gratuita o de paga. Las herramientas Open Source ERP son alternativas accesibles para pequeñas y medianas empresas, que pueden llegar a generar cambios en los procesos que se intervengan.

Planteamiento del problema

Manta es una de las ciudades con mayor desarrollo en Manabí, que es visitada por cientos de personas ya sea en el ámbito turístico y comercial. En la actualidad el sector comercial de la ciudad ha tenido un avance en los procesos administrativos, financieros en el campo de la tecnología y el uso de herramientas informáticas para la planificación de recursos empresariales.

Se destaca que las empresas de comercialización y productos de primera necesidad estén a la vanguardia en el ámbito tecnológico, buscando así el mejoramiento de los servicios que estas brindan a la colectividad.

El análisis realizado en la empresa "Todo Criollo" ha permitido determinar que la problemática se enfoca en la parte administrativa y financiera, donde sus procesos carecen del uso de tecnología que permitan un mejor desarrollo de los mismos, generando un déficit en el crecimiento de la empresa.

Aplicada la investigación preliminar permitió diagnosticar las siguientes situaciones problemáticas:

- La ausencia de un control de inventario manual o tecnológico, genera el desconocimiento parcial de existencia de los productos en bodega.
- El escaso control en los procesos contables, ocasiona errores y pérdida de información sobre la tributación de la empresa.

- La deficiencia en el registro de las compras, origina la repetitiva petición de productos existentes a un mismo proveedor.
- La débil capacitación del personal en temas tecnológicos, crea un conflicto al momento de la facturación en el punto de venta.
- El desconocimiento en el uso de tecnologías para las campañas publicitarias por medio de internet, produce pérdida de clientes potenciales que usan este medio frecuentemente.

Dichas las bases teóricas la cual nos indican cuán importante es "La investigación científica es una tarea dirigida a la solución de los problemas. La primera etapa es reducir el problema a términos concretos y explícitos"; tomando así esta referencia para canalizar nuestro estudio de implementación del ERP, por medio de esta propuesta se reduzca el problema encontrado, dando así nuestro aporte a la colectividad.(Méndez, 2010)

Diagrama Causa-Efecto del Problema

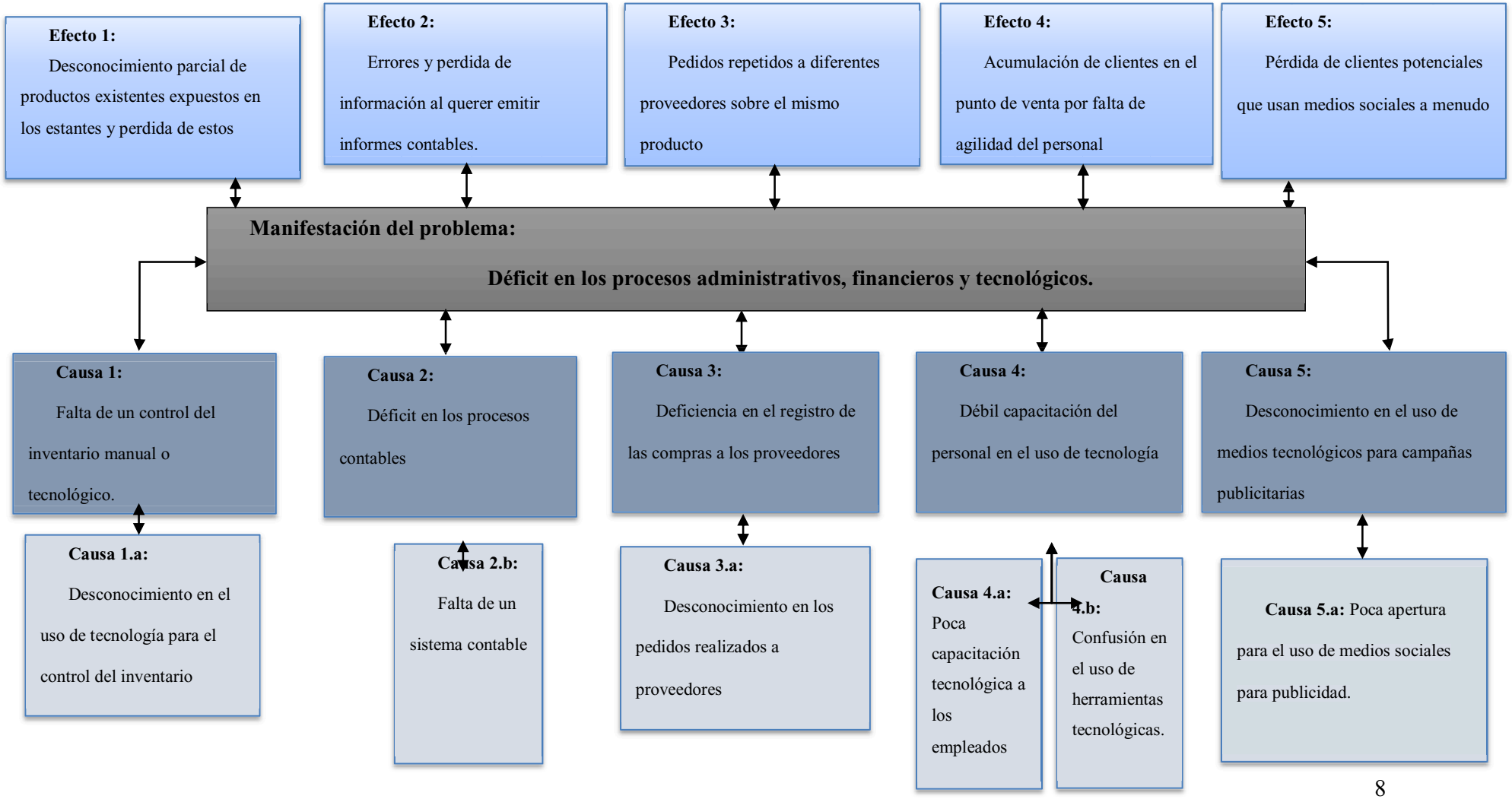


Ilustración 1 - Diagrama causa-efecto del problema

Objetivos

Objetivo general

Implementar un sistema ERP utilizando plataforma Odoo, para automatización integral de procesos administrativos, financieros y tecnológicos en la empresa "Todo Criollo" del Cantón Manta.

Objetivos específicos

- Diagnosticar la situación actual de la gestión integral de procesos que soporta la administración en la empresa.
- Determinar herramientas de la metodología de investigación y recursos necesarios en coordinación con la empresa, que permita establecer resultados, para ejecución del proyecto.
- Determinar herramientas y recursos de tecnología de información para la adaptación e integración en la implementación del sistema.
- Implementar módulos de inventario, facturación, contabilidad ofrecidos por plataforma Odoo.
- Desarrollar módulos para integrarse a la plataforma Odoo y una aplicación móvil tipo catálogo con conexión a medios sociales.

Justificación

Se ha observado que en la actualidad la tecnología que se puede aplicar a una empresa ha avanzado mucho, para ello existen diversas aplicaciones, como es el caso de los ERP Open Source, que lanzan día a día versiones de sus sistemas para contribuir al mejoramiento de procesos de negocios.

Un ERP es un software de gestión empresarial que permite planificar y controlar todos los procesos operativos y recursos de una empresa, integrando varias funciones de gestión en un único sistema. Con el uso de los sistemas ERP se trata de conseguir que todos los datos de la empresa estén integrados y conectados. (Lara Martinez , 2011)

Odoo, es un sistema de gestión empresarial de código abierto y sin coste de licencias. Se trata de un ERP tecnológicamente muy avanzado y que integra los procesos de toda la empresa ofreciendo una gestión eficiente de esta. Este sistema nos ofrece integración, potencia, interfaz intuitiva, flexibilidad, no tener restricciones del vendedor, conecta a sus usuarios, clientes y proveedores. (Praxya, 2013)

Una vez descrito el problema de cómo se evidencia el déficit en los procesos administrativos, financieros y tecnológicos los cuales conllevan a presentar dificultades internas y externas en los diferentes ámbitos de la empresa, se toma en consideración la aplicación de un sistema ERP que

automatice de manera integral las técnicas y métodos que la empresa requiera, con la propuesta que se está planteando, se llevaría de una mejor manera los procesos descritos. En virtud de ello, se ha considerado aportar con la empresa “Todo Criollo” para realizar una implementación de un sistema ERP que permita mejorar los procesos administrativos, financieros y tecnológicos, como el control de la mercancía en la empresa, la contabilidad, las ventas y compras de productos, la facturación tradicional, junto con la realización de técnicas de Social Media para manejar la publicidad en las redes sociales. Donde el gerente pueda conocer cuántos productos tienen en existencia y el control de las facturas emitida, obteniendo informes necesarios para la toma de decisiones en la empresa y una mejora a largo plazo.

Capítulo 1

Marco teórico de la investigación

1.1. Introducción

El marco teórico nos permite tener una visión claro sobre lo que se debe investigar, conceptos abordados en el proyecto y definiciones desconocidas. "Es necesario definirse entre lo que se sabe y lo que no se sabe con respecto al proyecto para delimitar el problema que se va a investigar". (Sabino, 1996).

El segundo punto del capítulo, considera los conceptos más relevantes del proyecto ejecutado, nos adentra en la historia de los sistemas de planificación empresarial, sus inicios, procesos de cambios y crecimiento exponencial en los últimos años y como estos forman parte de muchas empresas en el mundo. Tomando como referencia las universidades del país y las investigaciones concretadas en estas, nos enfocamos en tres archivos almacenados en los repositorios virtuales de dichas instituciones, para conocimiento de la situación del país en sistemas de planificación informáticos.

Como tercer punto, detallaremos las definiciones de los temas relaciones con el proyecto, como temas tecnológicos, metodológicos y de importancia para la investigación que nos darán una idea para la implementación del sistema de planificación bajo la herramienta Odoo.

El cuarto punto nos enfatiza que sin una ley que nos rija no podemos realizar la concepción de un proyecto, estas nos direccionan sobre los parámetros que debemos tener en cuenta para la ejecución, desarrollo y seguridad de la aplicación a implementar.

1.2. Antecedentes de investigaciones

Los programas de planificación de recursos empresariales, o ERP, pueden parecer un invento moderno. Sus orígenes se remontan a más de 60 años atrás, prácticamente en la "prehistoria" de la informática. (Dataprix, 2014)

Los ERP están presentes actualmente en la mayoría de las grandes empresas y cada vez más en las pymes, el contar con un software planificador de recursos empresariales es un "lujo" muy reciente, ya que hace aproximadamente veinte años solo estaban al alcance de las grandes multinacionales y tenían unas funciones bastante más limitadas que los programas que se conocen actualmente. (Dataprix, 2014)

Cronología de la evolución de los sistemas ERP:

1950: como tantas otras innovaciones, los ERP fueron un invento militar. A finales de la Segunda Guerra Mundial, el ejército de Estados Unidos empezó a usar programas informáticos para gestionar las complejas tareas de producción y logística del esfuerzo bélico. (Dataprix, 2014)

1960: la aparición de las primeras computadoras comerciales para empresas marcó el inicio de una nueva forma de gestionar la información en los negocios. En esa época, lo habitual era que el

software básico se entregara con la compra del hardware, aunque luego se podían contratar desarrollos a medida para adaptarlo a las necesidades de cada compañía. (Dataprix, 2014)

1970: en una época caracterizada por la incipiente escasez de ciertas materias primas como el petróleo, hicieron su aparición los programas MRP (Planificación de Necesidades de Materiales, en inglés). A diferencia de las aplicaciones de la década anterior, eran capaces de controlar no solo dónde y cómo se usaban los materiales, sino también de prever cuándo iban a ser necesarios y en qué cantidad. (Dataprix, 2014)

1980: los programas que usaban las empresas para planificar su producción evolucionaron para empezar a incluir otros ámbitos además de las materias primas. (Dataprix, 2014)

1990: es la década en la que nace el ERP tal y como lo conocemos hoy. Se atribuye a la consultora Gartner haber acuñado el término "ERP" (Sistema de Planificación de Recursos Empresariales, en inglés) para definir los nuevos programas de planificación empresarial que llegaban al mercado y cuyo alcance superaba ampliamente los ámbitos tradicionales de la fabricación y las finanzas, por lo que no tenía sentido seguir llamándolos MRP.

2000: los ERP se popularizaron y empezaron a integrar funciones que hasta entonces realizaban otras aplicaciones, como la gestión de las relaciones con los clientes (CRM) o la gestión de la

cadena de suministro (SCM). Eso llevó a algunos autores a proponer una nueva categoría denominada "Extended ERP" o ERP.

Los ERP en la actualidad. - En la segunda década del siglo XXI, estamos asistiendo a profundas transformaciones en los ERP con el fin de adaptarlos a las nuevas tendencias tecnológicas como el iCloud computing, los dispositivos móviles o el Software como Servicio (SaaS). (Dataprix, 2014)

La investigación sobre diferentes trabajos realizados en universidades ecuatorianas en relación a implementación de ERP, que han servidor de referencia en este proyecto están determinadas por:

Implantación de una herramienta erp software libre y desarrollo del anexo transaccional para la empresa de distribución de leche andina para Imbabura, (Tobar, 2011) sobre el manejo integral de los procesos comerciales de la empresa y anexo transaccional.

Otro de los proyectos hace referencia al análisis, diseño, desarrollo e implementación de un ERP (Enterprise resource planning) “Acsoft” de los módulos administrativo y contable para la empresa disprolim dedicada al sector industrial y químico, (Paredes Guerrero & Cabrera Gallardo, 2012) proyecto que propone facilitar y automatizar la gestión en los diferentes procesos de la empresa Disprolim.

El siguiente trabajo que contribuyó al proyecto fue la implementación del sistema "ERP social" en la unidad educativa intercultural "La Paz" y el despacho parroquial de "Cacha" de la ciudad de Riobamba y reingeniería del sistema, (Morales Mejía & Auquilla Buñay, 2015) este ofrece la automatización de los procesos como el registro de la información, así como los procesos de emisión de certificados para agilitar la atención brindada a la comunidad.

1.3. Definiciones Conceptuales

1.3.1. ERP

Sistema de planificación denominado ERP, siglas del nombre en inglés Enterprise Resource Planning, surgió de la necesidad de englobar todos los datos referentes a la totalidad de la cadena de producción de las empresas, con el fin de brindar información confiable en tiempo real. Mediante los sistemas ERP se realiza el seguimiento de las diversas áreas de una compañía, es de la fabricación de un producto, pasando por la logística, la distribución, el control de stock, la contabilidad de la organización y demás. (Marker, 2016)

Se trata básicamente de un software desarrollado para el manejo eficaz de la información de las empresas, que permite tomar decisiones acertadas en los momentos oportunos, gracias a la veracidad de los datos que se manejan mediante el ERP.

1.3.2. Social Media

Se denomina en sí como el futuro de la comunicación, un arsenal de herramientas y plataformas basadas en internet que aumentan y mejoran el compartir información. Este medio hace que la transferencia de textos, fotografías, audio, video e información en general, fluya entre los usuarios e internet. El social media tiene relevancia no solo entre los usuarios regulares de internet, sino en los negocios. (Definicion Social Media, 2015)

Como no podía ser de otro modo, detrás de esta transformación está Internet y, más concretamente, las redes sociales. La razón es evidente: antes de que aparecieran en escena, la única manera de obtener información sobre un determinado producto o servicio pasaba por solicitarla a un comercial. (López, Quer, & Valdés, 2014)

1.3.2.1 SEO

Se denomina posicionamiento en buscadores, posicionamiento web u optimización en motores de búsqueda (SEO por sus siglas en inglés, de Search Engine Optimization que se traduce, 'Optimización para motores de búsqueda') al proceso de mejorar la visibilidad de un sitio web en los diferentes buscadores, como Google, Bing o Yahoo de manera orgánica, es decir sin pagarle al buscador para tener acceso a una posición destacada en los resultados, según un determinado criterio de búsqueda. (Calvo, 2013)

Este posicionamiento se logra de manera natural realizando tareas de optimización en las páginas web. Con el objetivo de aparecer en las primeras posiciones de los buscadores y aumentar el tráfico de visitas en una página web, es conveniente que en los sitios se apliquen tareas de optimización

1.3.2.2 SEM

SEM es el término que se refiere a las campañas de anuncios a través de las plataformas e Google. SEM corresponde a las siglas en inglés Search Engine Marketing (marketing de buscadores) y como el SEO, el SEM es una de las técnicas más demanda y utilizada dentro del marketing online gracias a su facilidad a la hora de medir resultados, así como por los datos que arroja. El SEM es una modalidad de marketing en internet cuyo objetivo es aumentar la visibilidad de las páginas web en los "resultados de pago" (anuncios) de los motores de búsqueda a través de un sistema de pago por clic. (Solutum, 2016)

1.3.2.3 AdWords

AdWords es un programa publicitario de Google, y se trata de una herramienta rápida y fácil de utilizar que permite adquirir anuncios de coste por clic (CPC) que estén de forma correcta orientados, independientemente de cuál sea su presupuesto. Con este método, el anunciante sólo paga cuando un cliente hace clic en el anuncio, más allá de la cantidad de veces que éste aparezca en la web de Google. (Catt, 2011)

Google inserta publicidad en forma de AdWords, también conocidos como enlaces patrocinados o sponsor Ed links. Los anuncios de AdWords se muestran junto a los resultados de las búsquedas de Google (a la derecha o bien arriba), y sólo aparecen para determinadas palabras que el anunciante elige. Así mismo se ven en los sitios de búsqueda y de contenido de la creciente red de Google, que incluye AOL, Earthlink, HowStuffWorks y Blogger. De esta manera los anuncios consiguen llegar a un público muy amplio.

1.3.3. Odoo

Odoo es un entorno de desarrollo y un conjunto de aplicaciones de negocio completamente integradas y modulares que permiten escalar la solución completa e ir construyendo un sistema a medida de las necesidades del negocio. (Comunitea, 2016)

Esto permite que Odoo sea la aplicación perfecta para Pymes y pequeños negocios, permitiéndoles disponer de una aplicación de gestión completa a un coste muy reducido.

Pero la estructura modular de Odoo, su capacidad de adaptación y los miles de módulos disponible le permiten también crecer para adaptarse a empresas de mayor tamaño, cubriendo igualmente todas sus necesidades.

1.3.3.1 Arquitectura

Un Model-view-controller (MVC) es un patrón arquitectónico utilizado en la ingeniería de software. En las aplicaciones informáticas complejas que presentan gran cantidad de datos al usuario, a menudo se desea separar los datos (modelo)), De modo que los cambios en la interfaz

de usuario no afectan el manejo de los datos y que los datos pueden ser reorganizados sin cambiar la interfaz de usuario El controlador de vista de modelo resuelve este problema desacoplando el acceso a los datos y la lógica empresarial de la presentación de datos e Interacción del usuario, introduciendo un componente intermedio: el controlador. " (Plata, 2014, págs. 1-4)

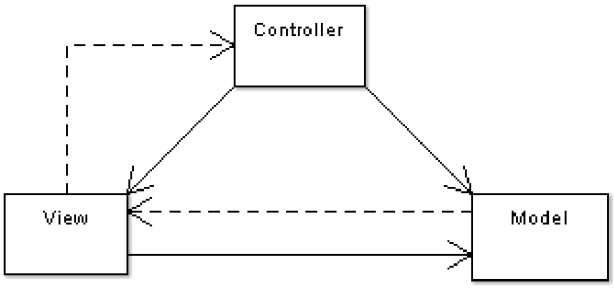


Ilustración 2 - Arquitectura Modelo Vista-Controlador

Por ejemplo, en el diagrama anterior, las líneas sólidas de las flechas que comienzan desde el controlador y van a la vista y al modelo significan que el controlador tiene un acceso completo tanto a la vista como al modelo. La línea discontinua para la flecha que va de la vista al controlador significa que la vista tiene un acceso limitado al controlador. Las razones de este diseño son:

Fue diseñado para reducir el esfuerzo de programación necesario en la implementación de sistemas múltiples y sincronizados de los mismos datos. Sus características principales están dadas por el hecho de que, el Modelo, las Vistas y los Controladores se tratan como entidades separadas; esto hace que cualquier cambio producido en el Modelo se refleje automáticamente en cada una de las Vistas. Este modelo de arquitectura se puede emplear en sistemas de representación gráfica de datos, donde se presentan partes del diseño con diferente escala de aumento, en ventanas separadas. (Romero, 2012, págs. 2-4)

De vista a modelo: el modelo envía notificación a la vista cuando sus datos han sido modificados para que la vista vuelva a dibujar su contenido. El modelo no necesita conocer el funcionamiento interno de la vista para realizar esta operación. Sin embargo, la vista debe tener acceso a las partes internas del modelo.

De la vista al controlador: la razón por la cual la vista tiene acceso limitado al controlador es porque las dependencias desde la vista al controlador necesitan ser mínimas: el controlador puede ser reemplazado en cualquier momento.

En Odoo, podemos aplicar esta semántica modelo-vista-controlador con:

Model: Las tablas de PostgreSQL.

View: las vistas se definen en archivos XML en Odoo.

Controller: Los objetos de Odoo.

El sistema de Odoo está formado por tres componentes principales

- El servidor de base de datos PostgreSQL, el cual contiene todas las bases de datos, cada una contiene todos los datos y la mayoría de los elementos de configuración relacionados con el sistema Odoo,
- El uso del servidor de Odoo el qué contiene toda la logística empresarial y asegura que se ejecute de manera óptima,

- El servidor web, una aplicación separada llamada la Web de cliente Open Objeto, que le permite conectarse a Odoo desde navegadores web estándar y no es necesario cuando se conecta utilizando un cliente GTK.

1.3.3.2. Lenguaje

Odoo ha sido desarrollado totalmente en una tecnología Web, basada en estándares abiertos. Construcción a 3 capas: BBDD, servidor y dispositivos de usuario. De tal manera que se define de la siguiente forma:

- Base de datos: Postgresql
- Lenguaje de programación: Python
- Lenguaje de programación web: HTML 5, JavaScript y css
- Está dotado de un entorno/framework de desarrollo rápido de aplicaciones (RAD) denominado Openobject.

Desde su versión 8.0 (Odoo, anteriormente OpenERP) incluye también un entorno de desarrollo web que permite construir aplicaciones móviles y web a medida. (OdooMrp, 2012)

1.3.3.3. Reportes QWeb Rml

QWeb es un motor (engine) de plantillas por Odoo. Está basado en XML y es utilizado para generar fragmentos y páginas HTML. QWeb fue introducido por primera vez en la versión 7.0

para habilitar vistas Kanban más ricas, y con las versiones 8.0, también se usa para la generación de reportes y páginas web CMS (CMS: Sistemas Manejadores de Contenido). (Odoo, 2013)

1.3.3.4 PostgreSQL

Es un sistema de gestión de base de datos relacional orientada a objetos y libre, publicado bajo la licencia BSD. Es más completo que MySQL ya que permite métodos almacenados, restricciones de integridad, vistas, etc. Cómo muchos otros proyectos de código abierto, el desarrollo de PostgreSQL no es manejado por una sola empresa, sino que es dirigido por una comunidad de desarrolladores y organizaciones comerciales las cuales trabajan en su desarrollo. Dicha comunidad es denominada el PGDG (PostgreSQL Global Development Group).

Es el sistema de gestión de bases de datos de código abierto más potente del mercado y en sus últimas versiones no tiene nada que envidiarles a otras bases de datos comerciales. Utiliza el lenguaje SQL para llevar a cabo sus búsquedas de información, las bases de datos generadas dentro de servidores de SQL son bases de datos relacionales. Utiliza un modelo cliente/servidor y usa multiprocesos en vez de multihilos para garantizar la estabilidad del sistema. Un fallo en uno de los procesos no afectará el resto y el sistema continuará funcionando. (SantaMaría, 2015, págs. 20-21)

1.3.3.5 XML

EL XML (eXtensible Markup Language, o lenguaje de Marcación Extendida) es un metalenguaje de marcación recomendada por W3C:

- Fue creada para ser auto descriptiva, transportar y almacenar datos, tiene su enfoque en el contenido propiamente dicho.
- Permite identificar claramente los elementos bibliográficos que componen al documento.
- Permite verificar la estructura y exactitud del artículo, y asegurar que cumple con las especificaciones con que debe contar el contenido de acuerdo con las reglas de formación.
- Lenguaje que puede ser entendido tanto por humanos como por maquinas

XML posibilita mayor flexibilidad, neutralidad y compatibilidad con sistemas o formatos futuros; capacidad de preservación digital del contenido a largo plazo, sin importar los sistemas operativos o programas de cómputo que existan en el futuro. Facilita la interoperabilidad entre sistemas y bases da dados. Los Mega jornal fueron pioneros en la utilización del XML en toda cadena de producción editorial. PLOS One, Nature e eLife. (Scielo, 2016)

1.3.3.6 ORM

Object-Relational mapping, o lo que es lo mismo, mapeo de objeto-relacional, es un modelo de programación que consiste en la transformación de las tablas de una base de datos, en una serie de entidades que simplifiquen las tareas básicas de acceso a los datos para el programador.

Desde hace muchos años el lenguaje más usado para acceder a las bases de datos relacionales ha sido el SQL.

El ORM permite convertir los datos de tus objetos en un formato correcto para poder guardar la información en una base de datos (mapeo) creándose una base de datos virtual donde los datos que se encuentran en nuestra aplicación, quedan vinculados a la base de datos (persistencia). (González, 2014).

1.3.4. Metodologías Agiles

Las metodologías en general se clasifican según su enfoque y características esenciales, las más recientes, que se fueron gestando a finales del siglo pasado y que se han comenzado a manifestar desde principios del actual, se han denominado "metodologías ágiles" y surgen como una alternativa a las tradicionales, estas metodologías se derivan de la lista de los principios que se encuentran en el "Manifiesto Ágil". (Manifiesto Agil, 2012).

La aparición de las metodologías ágiles no puede ser asociada a una única causa, sino a todo un conjunto de ellas, si bien es cierto que la mayoría de autores lo relacionan con una reacción a las metodologías tradicionales, ¿cuáles fueron las causas de esta reacción?, los factores que

comúnmente se mencionan son la pesadez, lentitud de reacción y exceso de documentación, en definitiva, falta de agilidad de los modelos de desarrollo formales; otro punto importante sería la explosión de la red, las aplicaciones Web y las aplicaciones móviles, así como el crecimiento notorio del movimiento open Source. (Balaguera, 2013).

1.3.5. Scrum

Scrum es un marco de trabajo que nos permite encontrar prácticas emergentes en dominios complejos, como la gestión de proyectos de innovación. No es un proceso completo, y mucho menos, una metodología. En lugar de proporcionar una descripción completa y detallada de cómo deben realizarse las tareas de un proyecto, genera un contexto relacional e iterativo, de inspección y adaptación constante para que los involucrados vayan creando su propio proceso. Esto ocurre debido a que no existen ni mejores ni buenas prácticas en un contexto complejo. Es el equipo de involucrados quien encontrará la mejor manera de resolver sus problemáticas. Este tipo de soluciones serán emergentes. (Martín, 2013).

1.3.5.1 Marco Técnico

El marco técnico de scrum, está formado por un conjunto de prácticas y reglas que dan respuesta a los siguientes principios de desarrollo ágil:

- Gestión evolutiva del producto, en lugar de la tradicional o predictiva.

- Calidad del resultado basado en el conocimiento tácito de las personas, antes que en el explícito de los procesos y la tecnología empleada.
- Estrategia de desarrollo incremental a través de iteraciones (Sprint).

Se comienza con la visión general del resultado que se desea, y a partir de ella se especifica y da detalle a las funcionalidades que se desean obtener en primer lugar. Cada ciclo de desarrollo o iteración (sprint) finaliza con la entrega de una parte operativa del producto (incremento). La duración de cada sprint puede ser desde una, hasta seis semanas, aunque se recomienda que no exceda de un mes. (Menzinsky & Palacio, 2016)

El marco técnico de scrum está formado por:

Roles:

- El equipo scrum.
- El dueño del producto.
- El Scrum Master.

Artefactos:

- Pila del producto.
- Pila del sprint.
- Incremento.

Eventos

- Sprint.

- Reunión de planificación del sprint.
- Scrum diario.
- Revisión del sprint.
- Retrospectiva del sprint.

Se denomina sprint a cada ciclo o iteración de trabajo que produce una parte del producto terminada y funcionalmente operativa (incremento) pueden adoptar dos tácticas diferentes para mantener un avance continuo en el proyecto:

- **Incremento iterativo:** basado en pulsos de tiempo prefijado (timeboxing)
- **Incremento continuo:** basado en el mantenimiento de un flujo continuo, no marcado por pulsos o sprints.

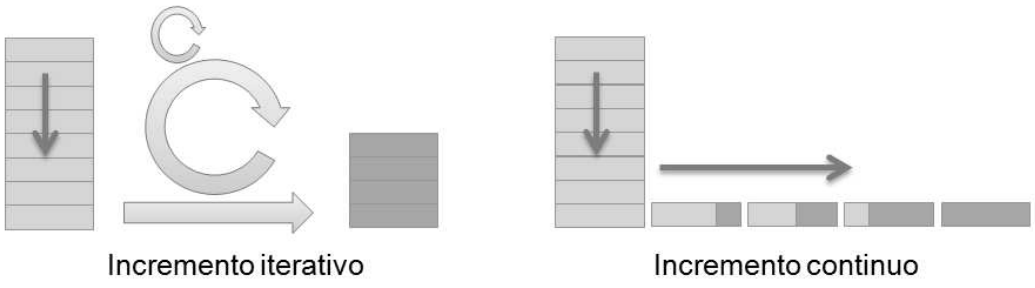


Ilustración 3 - Incrementos iterativo y continuo

1.3.5.2 Artefactos

Pila del producto: (product backlog) lista de requisitos de usuario, que a partir de la visión inicial del producto crece y evoluciona durante el desarrollo.

Pila del sprint: (sprint backlog) lista de los trabajos que debe realizar el equipo durante el sprint para generar el incremento previsto.

Incremento: resultado de cada sprint.

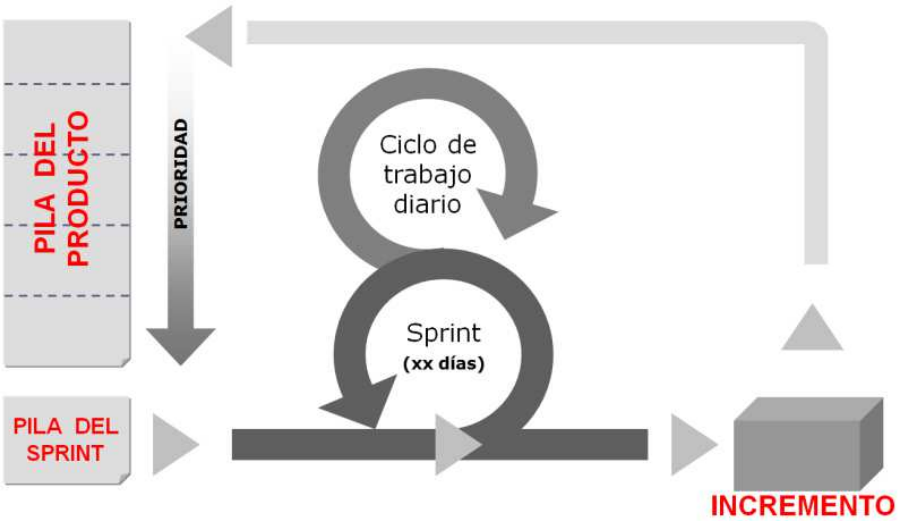


Ilustración 4 - Incremente en Scrum

1.3.5.3 Eventos

Sprint: nombre que recibe cada iteración de desarrollo. Es el núcleo central que genera el pulso de avance a ritmo de "tiempos prefijados" (time boxing).

Reunión de Planificación del sprint: reunión de trabajo que marca el inicio de cada sprint en la que se determina cuál es el objetivo del sprint y las tareas necesarias para conseguirlo.

Scrum diario: breve reunión diaria del equipo, en la que cada miembro responde a tres cuestiones:

1. El trabajo realizado el día anterior.
2. El que tiene previsto realizar.

3. Cosas que puede necesitar, o impedimentos que deben eliminarse para poder realizar el trabajo.

Cada persona actualiza en la pila del sprint el tiempo o esfuerzo pendiente de sus tareas, y con esta información se actualiza a su vez el gráfico con el que el equipo monitoriza el avance del sprint. (Menzinsky & Palacio, 2016)

Revisión del sprint: análisis e inspección del incremento generado, y adaptación de la pila del producto si resulta necesario.

Una cuarta reunión se incorporó al marco estándar de scrum en la primera década de 2.000:

Retrospectiva del sprint: revisión de lo sucedido durante el Sprint. Reunión en la que el equipo analiza aspectos operativos de la forma de trabajo y crea un plan de mejoras para aplicar en el próximo sprint. (Menzinsky & Palacio, 2016)

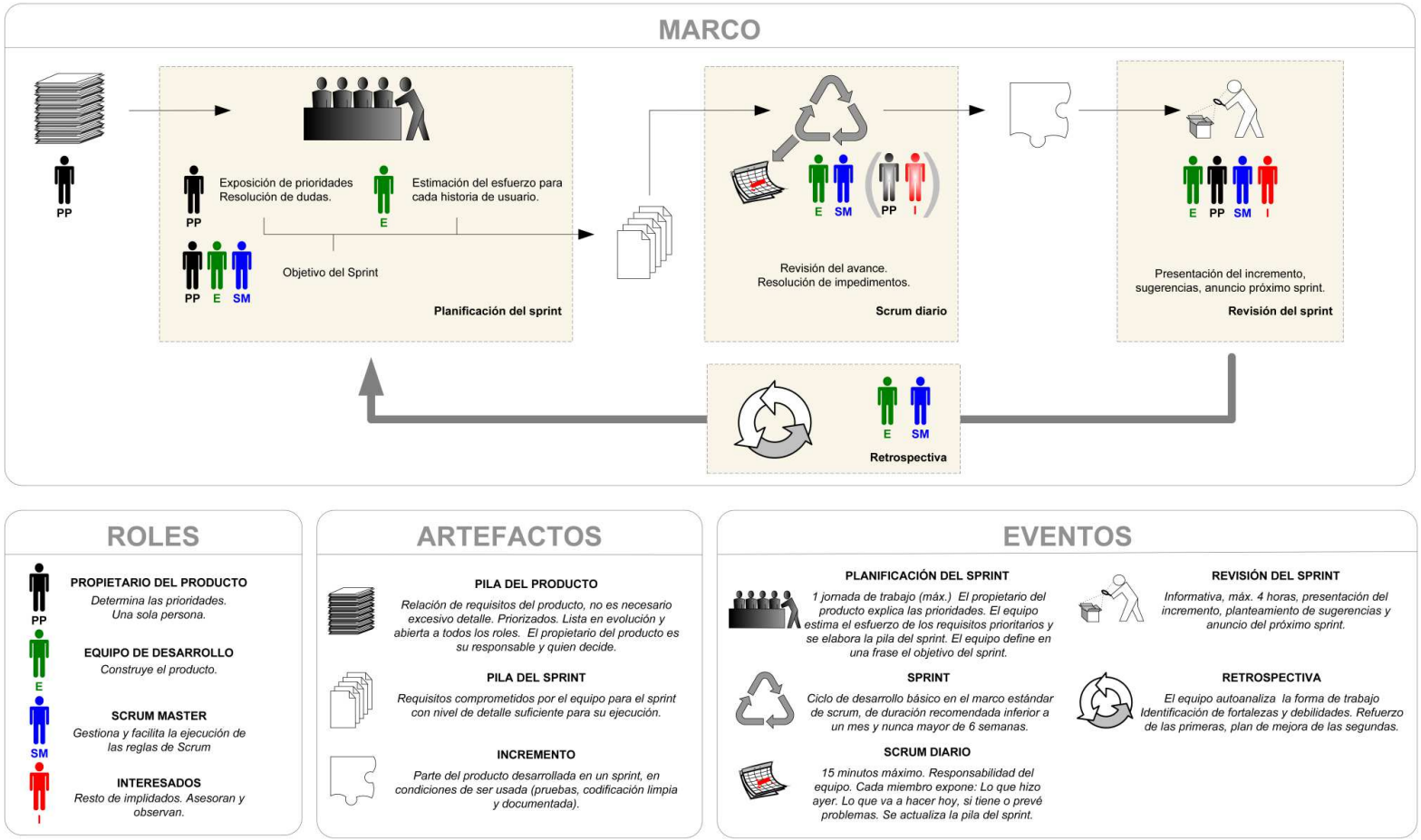


Ilustración 5 - Resumen Scrum

1.4. Fundamentación legal

Desde el año 2000 la integración de la tecnología en la sociedad ha dado pasos gigantes, como ejemplo el internet, este es el punto de partida que ha permitido a la tecnología ser parte de nuestra vida. Pero a medida que esta crece, tiende la necesidad de crear regulaciones y salvaguardar la información de accesos no autorizados, estas medidas adoptadas dependen del país y de como ellos apliquen el marco legal sobre el uso indiscriminado del software.

Centrándose en Ecuador en los últimos quince años se han dado una serie de regulaciones al uso de software, que va desde el uso en las empresas publicas hasta la salvaguarda de la información en diferentes medios.

La implementación del software libre en el ecuador se ve plasmado en los numerales de los decretos que se han efectuado, estos buscan que Ecuador tenga una soberanía y autonomía tecnológicas y con el uso de este alcanzar un ahorro en gasto público.

El COESC, Código Orgánico de la Economía Social del conocimiento promueve el uso de software libre en empresas públicas, con estos lineamientos podemos tomar un punto de partida para dirigir el uso de estas alternativas libres en las empresas privadas.

En la Constitución se garantiza la soberanía nacional, y se definen los sectores estratégicos entre los cuales están las tecnologías como hardware y software:

"Art 3. Son deberes primordiales del Estado: 2. Garantizar y defender la soberanía nacional."

Además, se garantiza el acceso a las tecnologías, la capacitación, su desarrollo y la integración regional.

"Art 16. Todas las personas, en forma individual o colectiva, tienen derecho a:

- 2. El acceso universal a las tecnologías de información y comunicación." (Constitucion, 2008)

En el plan nacional del buen vivir se contemplan estrategias y objetivos, los cuales incitan el cambio a la matriz productiva del país, con la propuesta hecha al comercial todo criollo podemos recalcar la ayuda a la colectividad para el uso de tecnológicas en las Pymes.

"Objetivo 10: Impulsar la transformación de la matriz productiva.

Una producción basada en la economía del conocimiento, para la promoción de la transformación de las estructuras de producción.

La transformación de la matriz productiva supone una interacción con la frontera científico técnica, en la que se producen cambios estructurales que direccionan las formas tradicionales del proceso y la estructura productiva actual, hacia nuevas formas de producir que promueven la diversificación productiva en nuevos sectores, con mayor intensidad en conocimientos, bajo consideraciones de asimetrías tecnológicas entre países. (Plan del Buen Vivir, 2013)

El gobierno plantea con el plan nacional del gobierno electrónico, un principio de adecuación tecnológica, este nos interesa para la ejecución del proyecto.

"Principio de adecuación tecnológica:

Garantiza que las administraciones elegirán las tecnologías más adecuadas para satisfacer sus necesidades, por lo que se recomienda el uso de estándares abiertos y de software libre en razón de la seguridad, sostenibilidad a largo plazo y la socialización del conocimiento." (Plan Nacional de Gobierno Electronico, 2014)

1.5. Conclusiones relacionadas al marco teórico en referencia al tema de investigación

Las definiciones referenciadas en este capítulo contribuyeron a comprender los temas más relevantes para el desarrollo del proyecto formando las bases de la investigación, para facilitar la aplicación de estos. La intención de estos conceptos tomados de varias fuentes es obtener información que nos direccione en la ejecución del proyecto. Los proyectos referenciados, obtenidos de los repositorios virtuales de sus universidades respectivas a nivel nacional, contribuyeron a conocer de que forma los sistemas de planificación empresarial están presentes en el Ecuador. La normativa legal ayudó al direccionamiento y medidas que debimos tomar para el desarrollo e implementación en las empresas privadas, las normativas y condiciones que se pueden plantear, teniendo en cuenta un enfoque de las empresas publicas reguladas por el gobierno en el uso de software.

Capítulo 2

Diagnóstico

2.1. Introducción

En el ámbito de la investigación la estructura que se va a escoger nos permitirá obtener respuestas del objeto de estudio, esta estructura es un enlace entre problema y solución que según el tipo de investigación observaremos fenómenos que se presenten en el proyecto, así poder entender el problema, analizarlo y generar posibles procedimientos que nos ayuden a esquematizarlo.

El segundo y tercer punto está enmarcado en el tipo de investigación y los métodos que se aplicaron para determinar los resultados obtenidos. La investigación descriptiva nos proporciona ideas sobre el fenómeno estudiado por medio de métodos que se definió para obtener estas ideas, así como el método de observación dando una valoración inicial al problema, método análisis-síntesis que permite el desmembramiento del problema y resumirlo en ideas claras. Las fuentes donde se obtiene la información primordial sobre el problema estudiado son de gran importancia ya que interactúa de manera directa con los implicados en el tema, siendo estos los que conocen la realidad del problema, usando técnicas y herramientas de recolección de datos que son tipo encuestas, determinando la información más crucial de los beneficiaros directos e indirectos. Como punto final se obtiene los resultados de la recolección de información, alcanzando así una idea clara de cómo se debe actuar ante la situación de este problema y donde focalizar la concepción del sistema.

2.2. Tipo(s) de investigación

2.2.1. Investigación Descriptiva.

Durante la investigación descriptiva, se procedió a realizar las encuestas. Una vez diseñadas y recolectada la información, validamos su contenido mediante cuadros estadísticos a 399 encuestas. Estas se realizaron el mes de febrero en distintas zonas de la cuidad con la colaboración de personas ajenas a la investigación, los resultados de las Encuentas contribuyeron a la descripción de las posibles soluciones y los problemas que podemos encontrar en la empresa.

2.2.2. Investigación de campo.

La investigación para el estudio de la implementación de un ERP en la empresa, comprende al campo tecnológico con referencia al área comercial que es donde la empresa se desenvuelve y los medios de difusión que se utilizaron para las campañas de marketing digital.

2.3. Métodos de investigación

2.3.1. Método de Observación.

Se considero la aplicación de este método, ya que "Observar es advertir los hechos como se presentan, de una manera espontánea, y consignarlos por escrito"; al ser este método un procedimiento que involucra en todos los aspectos a la investigación de campo, la cual se ha descrito con anterioridad, este método nos ha permitido observar los hechos por escrito de manera real y conocer las dificultades que los beneficiarios directos e indirectos perciben en la Empresa "Todo Criollo" (Méndez, 2010)

2.3.2. Método Análisis Síntesis.

2.3.2.1. Análisis:

- En la empresa "Todo Criollo" del cantón Manta, no se toma en cuenta la tecnología para el proceso de inventario y contabilidad.
- Aplicativo antiguo de punto de venta.
- No se lleva el inventario de sus productos.
- No se lleva contabilidad en algún sistema de información.
- Clientes insatisfechos en muchas ocasiones por la lenta atención.

2.3.2.2 Síntesis:

Relacionando los elementos descritos en el anterior análisis, el objetivo planteado nos direcciona a "Implementar un sistema ERP utilizando la plataforma Odoo, para la automatización integral de los procesos administrativos, financieros y tecnológicos en la empresa Todo Criollo.

2.4 Herramientas de recolección de datos

2.4.1. Encuesta.

La aplicación de esta herramienta contribuye al eficiente acceso a la información para los encuestadores implicados en la investigación, es importante enfatizar las necesidades que se ha podido denotar en la actualidad en la empresa Todo Criollo, se ha estructurado un banco de seis preguntas cerradas a 399 encuestados, captando las opiniones de los clientes del comercial y del uso de canales de comunicación, la medición se dio en escala nominal y ordinal.

2.4.3 Observación

Al ser este uno de los que guarda relación con el tema tratado, se evidencia que esta presenta como una técnica objetiva de recolección; la cual ayuda a obtener información aun cuando no se la está buscando, siendo independiente de las personas a estudiar y sin intermediarios, esto repercute a que no exista distorsiones. En nuestro caso lo observado ha sido fundamental por el contacto con los beneficiaros directos, de la empresa, describiendo esta como una observación participante en la cual pasamos a formar parte del grupo o una situación determinada

2.5. Fuentes de información de datos

2.5.1. Fuentes Primarias.

Este tipo de fuentes es la que obtienen a primera mano, ya que, al ser directas y personalizadas, siendo nuestro tema orientado a la observación, se ha escogido a la encuesta como fuente primaria, debido que al ser percibidos sin intermediación se logran resultados concretos sobre la solución a los problemas que se presenten.

Entre estas encontramos:

- Personal administrativo de la empresa "Todo Criollo"
- Clientes
- Encuestas
- Libros
- Libros virtuales

2.5.2. Fuentes Secundarias.

Para recabar información en las diferentes investigaciones que se pueden dar, es necesario recurrir a este tipo de fuentes, como las fuentes virtuales, internet, material documental, los cuales nos ayudaran en el proceso de plasmar el problema y brindar la solución adecuada. Usamos las siguientes fuentes:

- Linkografías
- Revistas
- Periódicos
- Repositorios Digitales

2.6. Instrumental operacional

2.6.1. Estructura y características de lo(s) instrumento(s) de recolección de datos

2.6.1.1. Encuesta

2.6.1.1.1. Características. Se ha considerado para la aplicación de encuesta a los beneficiarios indirectos, por lo que en el estudio tendremos a 399 encuestados, estos son lo que interactúan con el sistema independiente de la empresa. Al ser esta herramienta considerada básica para la investigación, dado que nos da resultados objetivos sobre como los clientes ven el problema en la empresa.

2.6.1.1.2. Estructura. – La encuesta está diseñada con 6 preguntas cerradas en escala nominal, y al no tener conocimiento del tema a tratar queda por finalizada la encuesta. Esta encuesta estará detallada en anexo A.

2.6.1.3. Observación.

2.6.1.3.1. Característica. - Esta se caracteriza por obtener los datos de medios visuales, donde se observa en todos los ámbitos las características del problema encontrado en la empresa para así realizar un análisis de forma clara y precisa sobre la necesidad que pueda precisar la empresa "Todo Criollo", dando así resultados para poder seguir con el cumplimiento de los objetivos planteados con anterioridad.

2.6.1.3.2. Estructura. – Se ha realizado una ficha para así plasmar en escritos lo observado, en la cual se encuentran puntos claves para resumir el problema. La estructura de la observación estará en el anexo C.

2.7. Estrategia operacional para la recolección y tabulación de datos

Recolectar los datos implica elaborar un plan detallado de procedimientos que nos conduzcan a reunir datos con un propósito específico. Este plan incluye determinar:

a) ¿Cuáles son las fuentes de donde se obtendrán los datos? Es decir, los datos van a ser proporcionados por personas, se producirán de observaciones o se encuentran en documentos, archivos, bases de datos, etcétera.

b) ¿En dónde se localizan tales fuentes? Regularmente en la muestra seleccionada, pero es indispensable definir con precisión.

c) ¿A través de qué medio o método vamos a recolectar los datos? Esta fase implica elegir uno o varios medios y definir los procedimientos que se utilizaran en la recolección de los datos. El método o métodos deben ser confiables, válidos y objetivos.

d) Una vez recolectados, ¿de qué forma se va a plantear para que puedan analizarse y respondan al planteamiento del problema?

2.7.1. Plan para la recolección de datos.

Tabla 1 - Plan de Recolección de Datos
Fuente: Elaboración propia

ACTIVIDADES	DESCRIPCIÓN	RESPONSABLE
¿Qué datos recolectar?	Requerimientos, realidad.	Jeffri Villacreses
¿Dónde y cómo recolectarlos?	Ciudadanía del Cantón Manta en general y la empresa Todo Criollo.	Jeffri Villacreses y Bryan Cedeño
¿Quién lleva a cabo las encuestas?	Las encuestas las realizo Byran Cedeño	Bryan Cedeño
¿Qué métodos se aplicarán en el proceso?	Se aplicarán procesos de análisis síntesis.	Jeffri Villacreses
¿Qué tipo de preguntas realizar?	Se centrarán en preguntas cerradas.	Jeffri Villacreses

¿Quién estructurará el diseño y construye las preguntas?	El equipo de trabajo diseño las preguntas.	Bryan Cedeño y Jeffri Villacreses
¿Cómo presentarlos?	En cuadros estadísticos.	Jeffri Villacreses y Bryan Cedeño

2.7.2. Plan para la tabulación de datos.

Tabla 2 - Plan de Tabulación de Datos
Fuente: Elaboración propia

ACTIVIDADES	DESCRIPCIÓN	RESPONSABLE
¿Qué datos se va a ordenar, tabular y ordenar?	Los datos obtenidos por las encuestas y cuestionarios	Jeffri Villacreses y Bryan Cedeño
¿Qué información se va a presentar?	La información necesaria para la ejecución del proyecto	Jeffri Villacreses
¿Qué clase software se aplicará?	Se usará Excel para la creación de los gráficos.	Jeffri Villacreses y Bryan Cedeño
¿Cómo presentarlos?	Por medio de diapositivas	Jeffri Villacreses y Bryan Cedeño

2.7.3. Plan de análisis e interpretación de datos

Tabla 3 – Plan de análisis e interpretación de datos
Fuente: Elaboración propia

ACTIVIDADES	DESCRIPCIÓN	RESPONSABLE
¿Quién analizara los datos?	Los autores del documento.	Jeffri Villacreses y Bryan Cedeño
¿De que forma se realizara el análisis?	Por medio de los métodos de investigación descritos.	Jeffri Villacreses
¿Cuándo se realiza esta actividad?	La segunda semana de abril	Jeffri Villacreses y Bryan Cedeño
¿Cómo presentarlos?	Por medio de diapositivas	Jeffri Villacreses y Bryan Cedeño

2.8. Plan de muestreo

Aquí el interés se centra en "qué o quiénes", es decir, en los participantes, objetos, sucesos o comunidades de estudio (las unidades de análisis), lo cual depende del planteamiento de la investigación y de los alcances del estudio. (Sampieri, 2010)

2.8.1. Población

La Población considerada para llevar a cabo la investigación se encuentra en el país Ecuador, provincia de Manabí, Cantón Manta, se usará la información del Instituto Nacional de Estadísticas y Censo (INEC, 2010). Manta cuenta con 226.477 habitantes, se considera entonces que la población es infinita por cuanto pasa de los 100.000 habitantes, según demuestra a continuación:

Tabla 4 - Población total de Manta
Fuente: Censo INEC 2010

MANTA		URBANO	RURAL	**Total**
	MANTA	217.553	3.569	221.122
	SAN LORENZO	-	2.647	2.647
	SANTA MARIANITA	-	2.708	2.708
	Total	217.553	8.924	226.477

2.8.2. Segmentación

Se ha segmentado tomando en consideración a la población más potencial de consumidores a partir de los 15 a 64 años de edad, que totaliza 145.027, de los cuales se cuenta con 70.940 hombres y mujeres 74.087.

Tabla 5 - Segmentación de población de Manta
Fuente: Censo INEC 2010

Población del Cantón Manta por: Grandes grupos de edad	Hombre	Mujer	Total
Población del Cantón Manta por: De 0 a 14 años	35052	34455	69507
Población del Cantón Manta por: De 15 a 64 años	70940	74087	145027
Población del Cantón Manta por: De 65 años y más	5411	6532	11943
Población del Cantón Manta por: Total	111403	115074	226477

2.8.3 Técnicas de muestreo

2.8.3.1. Muestreo no probabilístico

La población de 145.027 segmentada entre las edades de los 15 a 64 años, este grupo se consideró ya que tienen ya un conocimiento básico del tema a tratar cuya información que se

recepta será necesaria para una toma de decisión de acuerdo a la propuesta planteada, dando así dos clases de muestras aplicadas, las cuales son:

2.8.3.2. Muestreo por conveniencia.

Este tipo de muestreo nos permitirá medir la eficiencia o ineficiencia en el servicio que la empresa "Todo Criollo" tiene para los procesos de negocios que ellos manejan.

2.8.3.3. Muestreo por cuota

Se ha considerado por cuanto se tiene la certeza que dentro de la agrupación seleccionada nos darán sus opiniones que se las da por acertadas y representa una aportación muy valedera para el presente estudio.

2.8.4 Tamaño de la muestra

Se ha considerado que en la población del Cantón Manta de edades entre 15 a 64 años; son los que nos aportan con sus criterios y opiniones dentro de las encuestas planteadas, tal es así que se determinó a un número de 399, determinada mediante la fórmula siguiente:

Descripción
Fórmula:
n = muestra

$$n = \frac{N}{E^2(N-1)+1}$$

$$n = \frac{145.027}{0.0025\,(145.026)+1}$$

$$n = \frac{145.027}{0.5^2\,(145.027-1)+1}$$

$$n = \frac{145.027}{363,56}$$

$$n = 399$$

2.9 Presentación y análisis de los resultados

Análisis de resultados:

- Con los resultados obtenidos en la encuesta realizada, y específicamente en las primeras preguntas se puede concluir que el internet es el canal de comunicación más usado de las personas de 15 a 64 años representando el 33% de los encuestados, esto permite tener una idea clara sobre el direccionamiento de las campañas publicitarias según la edad y el canal de comunicación usado, aportando así al desarrollo de una aplicación móvil.
- Debido a que, en el centro de Manta, específicamente en el comercial "Todo Criollo" es el más concurrido según los resultados de las preguntas tres y cuatro en la encuesta, se toma a consideración para tomar medidas en la implementación que permita mejoras en el proceso administrativo específicamente la atención al cliente y la facturación por puntos de venta.
- Según los encuestados la atención que ellos obtienen en actuales momentos nos indica que no es la más adecuada para el servicio que se debería brindar, con esto concluimos que los procesos administrativos y tecnológicos están desatendidos en temas de herramientas informáticas obsoletas y poca ayuda al usuario al momento de realizar una compra.

Los detalles de la encuentra realizada estará detallado en el anexo B.

Capítulo 3

Diseño de la propuesta

3.1. Introducción

La implementación de sistemas de planificación empresarial en la actualidad constituye un eje primordial para el funcionamiento de diferentes departamentos de instituciones comerciales, centralizando la información y brindando informes para la toma de decisiones de las áreas implicadas en los procesos automatizados. Estos ayudan de manera exponencial a la eficacia de los empleados.

Iniciando la propuesta se detallan estamentos para la ejecución del proyecto y se deslinda por Odoo herramienta para implementación, instando módulos a instalar, complementos desarrollados e integraciones con campañas de marketing digital.

La siguiente parte del capítulo consta de especificaciones técnicas para el uso del sistema, el alcance del proyecto delimitando módulos a instalar y detallando recursos necesarios para la implementación del sistema y una línea de tiempo que seguimos para la conclusión del proyecto.

Como punto final está la metodología ágil de desarrollo scrum, la cual permite una interacción con el usuario final, retroalimentando continuamente el sistema, y así poder en poco tiempo realizar cambios que permitan mejorar. Dividida en 5 sprint de 10 a 20 días, con una duración total de 65 días por todo el proyecto.

3.2. Descripción de la propuesta

Esta propuesta se plantea para el mejoramiento de los procesos en la empresa Todo Criollo, mediante la implementación de un ERP. La herramienta seleccionada es Odoo, que cuenta con una serie de módulos esenciales para el manejo de la empresa y con el beneficio de crear y modificar a conveniencia el código, junto a una planificación de marketing digital y la creación de una página web para la empresa, con una aplicación móvil estilo catalogo para los clientes.

Los módulos a implementar son los siguientes, se encuentran divididos en desarrollados y predesarrollados:

Predesarrollados:

- Módulo Administración Financiera
- Módulo Compra
- Módulo de Venta
- Módulo de Inventario

Desarrollados:

- Módulo de Peso Precio
- Módulo de exportación de productos y Códigos Ean
- Adaptación del módulo de Punto de Venta

- Módulo Anexo Transaccional
- Módulo de legislación ecuatoriana

Social Media:

- Facebook
- Google AdWords

Aplicación Móvil:

- Android

3.2.1. Especificaciones técnicas

Las especificaciones técnicas que se tomaran en cuenta para la ejecución del proyecto son:

- Base de datos: PostgreSQL.
- Servidor: Linux
- Framework: Odoo
- Lenguaje de programación: Python.

3.2.2. Alcance de la propuesta.

Esta propuesta busca en la empresa "Todo Criollo" mejorar los procesos administrativos, financieros y tecnológicos, específicamente en la venta, compra, facturación, inventario y contabilidad, viendo en estos un déficit notable para el desarrollo de la institución.

La implementación de los módulos básicos del Odoo ERP, siendo estos los siguientes:

- Administración Financiera.
- Punto de Venta
- Ventas
- Gestión de inventario
- Compras

El desarrollo de los complementos para el correcto funcionamiento del sistema ERP, vienen acompañado de un sitio web informativo sobre los productos y de una aplicación móvil tipo catálogo; los complementos desarrollados serán de acorde a lo establecido en las historias de usuarios y al análisis con lo que determinaremos cuales son los puntos en los que el sistema ERP debe mejorar.

Con la implementación de la primera fase para el mejoramiento de los procesos en la empresa damos por finalizado lo propuesto por medio de este documento, teniendo planes a futuros luego de la titulación que contribuirán a la colectividad, al comercial "Todo Criollo" y para nuestra formación profesional en el ámbito de los negocios.

3.2.3. Recursos

3.2.3.1. Humano

El recurso humano es la concepción de las personas implicadas en la ejecución del proyecto, a continuación, se detallan los implicados:

Tabla 6 - Recursos humanos
Fuente: Elaboración propia

Recursos Humanos	Función
Rafael Palacios Mero – Gerente	Capitalizador de la ejecución del proyecto.
Marisela Saltos – Secretaria	Manejo del sistema y proporcionadora de requisitos.
José Cristóbal Arteaga Vera - Tutor	Tutor del presente trabajo de titulación
Jeffri Villacreses Reynaldo – Autor	Desarrollador del presente trabajo de titulación
Bryan Cedeño Cedeño - Autor	Desarrolladora del presente trabajo de titulación

3.2.3.2. Tecnológicos.

Tabla 7 - Recursos tecnológicos
Fuente: Elaboración propia

Tecnológicos.	Función
Computadoras	Para el funcionamiento del punto de venta
Impresora	Imprimir los reportes generados por el sistema
Servidor Linux	Contiene los servicios de Odoo y la base de datos
Impresoras de Punto de Venta	Imprimir las facturas para los clientes
Ip Publica – Internet de UAV.	Permite el funcionamiento del servidor, fuera del complejo de la empresa

Impresora térmica de etiquetas.	Imprimir las etiquetas de los productos del sistema.
Play Store	Contendrá la aplicación móvil catalogo
Hosting	Hospedara la aplicación móvil.

3.2.3.3. Presupuesto para la implementación.

Tabla 8 - Recursos económicos (Presupuesto)
Fuente: Elaboración propia

Cantidad	Recurso	Costo U.	Total
Presupuesto: Recurso Humano.			
240	Horas del trabajo de titulación.	$9,00	$2160,00
160	Horas de tutorías	$0,00	$0,00
Presupuesto: Tecnológico.			
1	Impresora	$375,00	$375,00
2	Impresora Punto de Venta	$250,00	$500,00
1	Impresora de Etiquetas	$560,00	$560,00
1	Servidor Linux	$900,00	$900,00
2	Computadoras de Escritorio	$400,00	$800,00
1	Ip publica	$60,00	$60,00
1	Servicio de internet	$50,00	$150,00
1	Servicio de hosting	$84,00	$84,00
Presupuesto: Viáticos			
1	Gasolina	$300,00	$300,00
1	Varios	$200,0	$200,00
		Subtotal	$5439,60
		IVA	$649,39
		Total	$6089,00

3.2.4. Cronograma

Tabla 9 - Cronograma del proyecto
Fuente: Elaboración propia

Cronograma del proyecto		
Fecha Inicio.	**Fecha Fin**	**Descripción**
11-04-2017	11-04-2017	Inicio
11-04-2017	21-04-2017	Análisis
24-04-2017	05-06-2017	Diseño
08-05-2017	02-06-2017	Desarrollo
05-06-2017	23-06-2017	Implementación
26-06-2107	07-07-2017	Documentación
07-07-2017	07-07-2017	Fin

3.2.5. Línea de tiempo

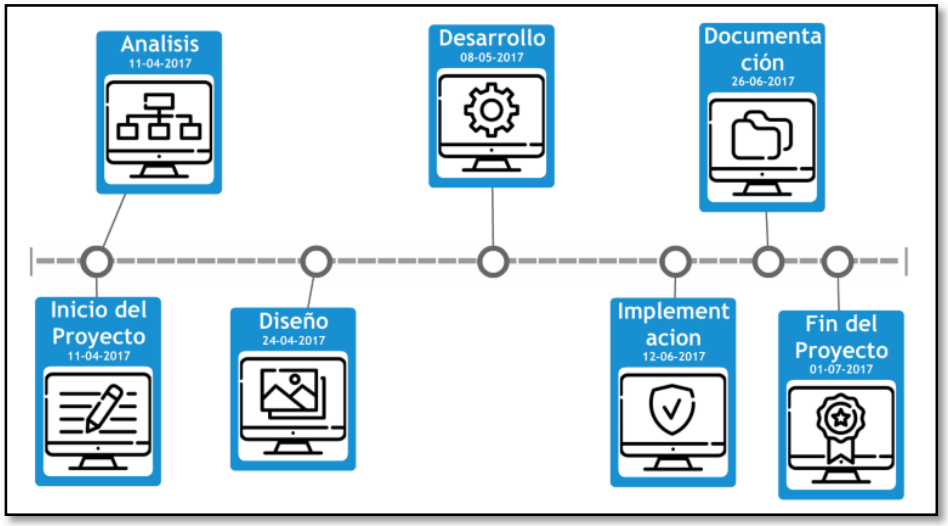


Ilustración 6 - Línea de tiempo del proyecto
Fuente: Elaboración propia

3.3. Etapas de la propuesta

3.3.1. Selección de la metodología de desarrollo

La metodología ágil, se usa con más frecuencia en nuevos sistemas de gestión que se encargan de dar respuestas a ciertos problemas en el transcurso de la ejecución del proyecto, es por esta virtud el cual elegimos la metodología Scrum para la concepción del proyecto, en donde realizamos una lista de requisitos priorizados, demostrando los resultados en cada iteración, dividendo las tareas por equipos de trabajo y asignando roles a los participantes del proyecto.

3.3.2. Bosquejo de la metodología Scrum.

La planificación en la metodología scrum es esencial, para ello se estableció un bien definido equipo de trabajo, estableciendo de forma clara y concisa los objetivos del proyecto en el product backlog y a la vez estos objetivos son divididos en iteraciones de máximo 20 días planificando en cada una de estas las tareas a realizar en el sprint backlog. Estableciendo cada día reuniones cortas de 10 minutos para verificar los avances de las diferentes tareas y realizar la retroalimentación y determinar un plan de acción para solucionar los inconvenientes suscitados.

3.3.2.1. Definición del equipo de trabajo

El equipo de trabajo en scrum se definió de la siguiente manera:

- **Product Owner.** – El dueño del producto es el puente entre los clientes y el grupo de desarrollo, este rol será desempeñado por Bryan Cedeño Supervisor del departamento de Tics en la empresa.

- **Scrum Master. –** Aquel que nos guiará en el proceso de scrum y organiza las reuniones, estará a cargo del Ing. José Arteaga.
- **Equipo de desarrollo.** – Realizará el proceso de la creación del producto, será desempeñado por Jeffri Villacreses, estudiante de la universidad y parte del equipo de Tics de la empresa.

3.3.2.2. Descripción de los objetivos del producto (Product Backlog)

Tabla 10 - Lista de objetivos del producto
Fuente: Elaboración propia

Orden	Titulo	Estimación
1	Requisitos, análisis	2
2	Diagramas y bocetos del sistema.	4
3	Modelado de la base de datos	7
4	Desarrollo de módulos	9
5	Infraestructura Física	7
6	Implementación del ERP	6
7	Seguridad, pruebas e Informes.	6
8	Campaña Social Media	5

3.3.2.3. Explicación de las iteraciones (Sprint Backlog)

Una vez planteada la pila del producto procederemos a definir los sprint y cuales tareas haremos en cada uno de ellos:

3.3.2.3.1. Descripción del Sprint 1

- Análisis del Negocio
- Recolección de las historias de usuarios
- Requisitos Funcionales
- Requisitos no Funcionales
- Diagramas de Casos de uso
- Diagramas de Secuencia

3.3.2.3.2. Descripción del Sprint 2

- Bocetos del sistema(Interfaces)
- Arquitectura de Odoo.
- Modelado de la base de datos
- Diagrama de la base de datos

3.3.2.3.3. Descripción del Sprint 3

- Desarrollo del aplicativo
- Desarrollo de módulos adicionales
- Desarrollo de aplicativo móvil
- Desarrollo de página web

3.3.2.3.4. Descripción del Sprint 4

- Instalación de red local.
- Adecuación de red cliente-servidor
- Adquisición de hosting para alojamiento web
- Implementación de los módulos provistos por Odoo.
- Instalación de módulos desarrollados.
- Configuración de Odoo.

3.3.2.3.5. Descripción del Sprint 5

- Capacitaciones
- Campaña de marketing por medio de Facebook.
- Campaña de marketing por medio de Google.

3.3.2.4. Planeación de las iteraciones (Sprint)

Las iteraciones tendrán una duración de 10 a 20 días laborables distribuidos de la siguiente manera:

Tabla 11- Planeación de iteraciones
Fuente: Elaboración propia

#	Fecha	Días
Sprint 1	Del 11 de abril al 21 de abril	10
Sprint 2	Del 24 de abril al 05 de mayo	10
Sprint 3	Del 08 de mayo al 02 de junio	20
Sprint 4	Del 05 de junio al 23 de junio	15
Sprint 5	Del 26 de junio al 07 de julio	10

3.3.2.5. Explicación de la retroalimentación.

Las reuniones de retroalimentación se realizarán horas después de la presentación del incremento del producto en la cual se procederá a conversar, debatir y contrastar las tareas no terminadas o con complicaciones para darle una resolución inmediata.

3.3.3. Detalle de iteraciones

A continuación, se pondrán a conocimiento las tareas que en cada sprint se dieron durante la ejecución del proyecto.

3.3.3.1. Sprint 1. Del 11 de abril al 21 de abril

3.3.3.1.1. Reunión de planificación del sprint 1

El primer sprint está enmarcado en el análisis de la situación actual del negocio y de la recolección de los requisitos. Una vez detallados este se procederá a la diagramación de los casos de usos y la secuencia del sistema.

3.3.3.1.2. Análisis del negocio.

El proceso del inventario actual en la empresa solo se concentra en la recepción del producto y el almacenamiento en bodega no se lleva un control de existencias.

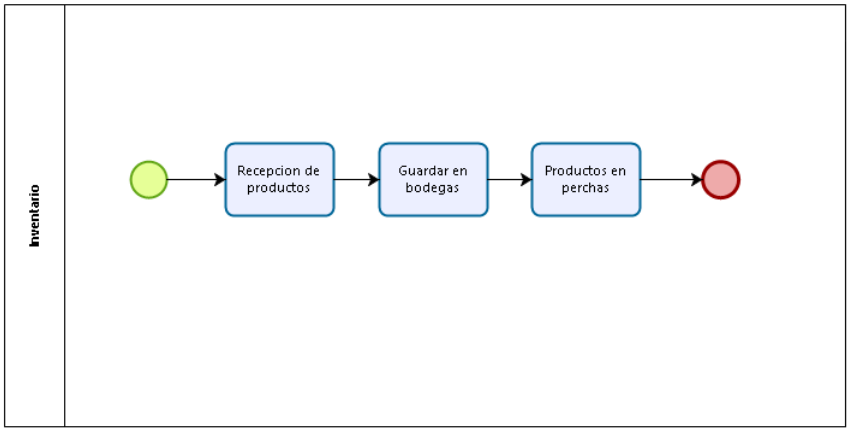


Ilustración 7- Proceso de negocio – Inventario
Fuente: Elaboración propia

La contabilidad en la empresa solo se concentra en un contador externo a los cuales se le entregan las facturas y no se comparten ningún proceso más que intervenga la empresa.

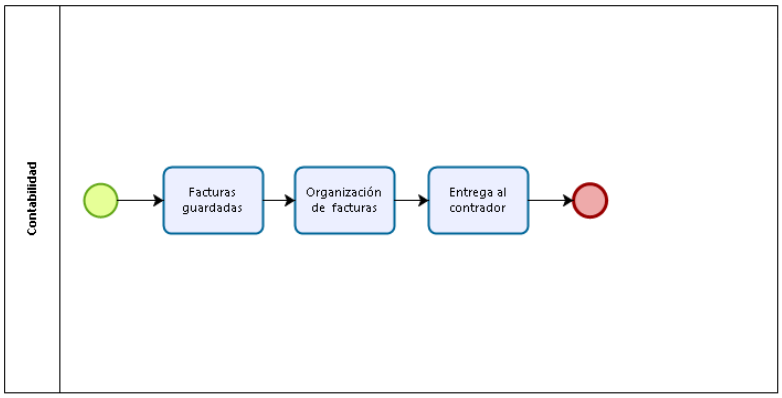


Ilustración 8- Proceso de negocio – Contabilidad
Fuente: Elaboración propia

Detalle del proceso de venta actual en la empresa:

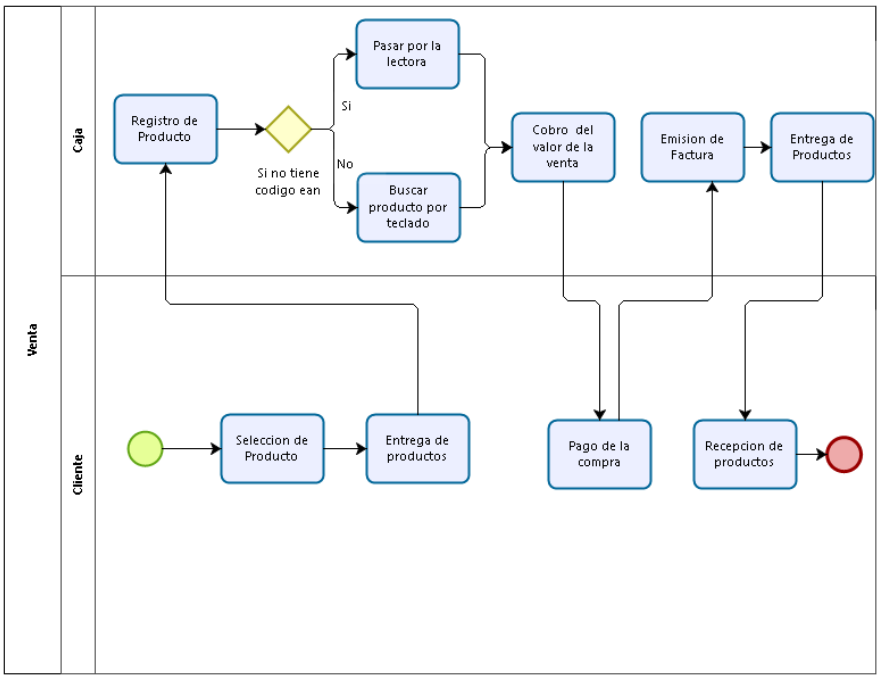


Ilustración 9- Proceso de negocio – Ventas
Fuente: Elaboración propia

Proceso de compra actual en la empresa:

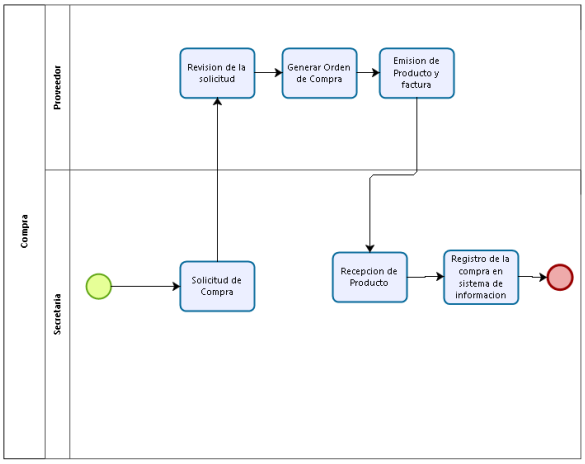


Ilustración 10 - Proceso de negocio – Compra
Fuente: Elaboración propia

3.3.3.1.3. Historias de Usuarios.

En este punto se detallarán las historias de usuario, que por medio de las primeras reuniones con los Stackholder se dieron a relucir detallando y priorizando cada historia de usuario según su dificultad. La primera historia de usuario estará puesta en este punto, las demás estarán en la sección de anexo B.

Tabla 12 - Historia de usuario 1
Fuente: Elaboración propia

HISTORIA DE USUARIO	
Numero: 1	**Nombre de la funcionalidad:** El sistema permita generar un código ean exclusivo para los productos envasados.
Usuario: Administrador, Usuario	
Prioridad en el negocio: Baja	**Riesgo de desarrollo:** Baja
Descripción: El usuario al podrá por medio de un clic generar un código ean prestablecido generando valores empezados por cinco al negocio.	
Observaciones: que el botón de generar aparezca con letra subrayada	

3.3.3.1.4. Requisitos funcionales.

- Roles de usuarios
- Facturación
- Control de inventario
- Impresión de códigos
- Control en variación de productos
- Anexo transaccional

- Exportación de archivo xls.
- Punto de venta
- Retenciones-Impuestos

3.3.3.1.5. Requisitos no funcionales.

- Servidor Linux
- Red LAN
- PostgreSQL 9.5
- Lenguaje Python
- Ip Publica
- Framework Odoo
- IDE SublimeText

3.3.3.1.6. Diagramas de casos de uso.

Generar código Ean. - El usuario o administrador podrá generar automáticamente un código Ean para el producto que desee, buscando primero el producto y luego en la pantalla de información dar clic en generar código.

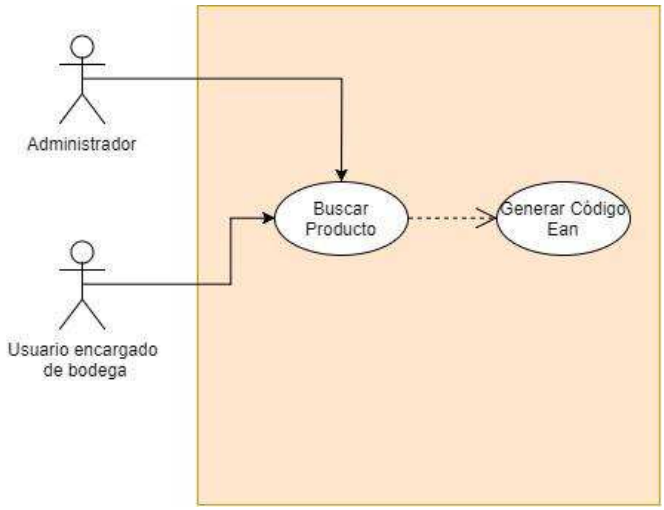


Ilustración 11- Caso de uso- Generar código EAN
Fuente: Elaboración propia

Los diagramas estarán adjuntos en el anexo C.

3.3.3.1.7. Diagramas de secuencia.

Generar código Ean.

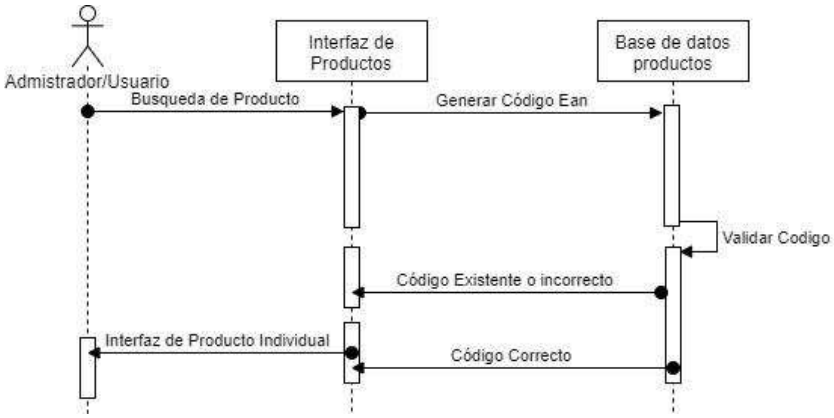


Ilustración 12 - Diagrama de secuencia - Generar código Ean
Fuente: Elaboración propia

Los diagramas estarán especificados en el anexo D.

3.3.3.2. Sprint 2 Del 24 de abril al 05 de mayo

3.3.3.2.1. Reunión de planificación del sprint 2

La reunión número dos se enmarca en el desarrollo de los diseños de los módulos adicionales, una introducción a la arquitectura en Odoo y cuáles van a ser las tablas las cuales se intervendrán en el desarrollo del proyecto.

Asistentes:

- Cedeño Cedeño Bryan (Scrum team) dueño del producto.
- Villacreses Lucas Jeffri (Scrum team) desarrollador.

Actividades:

3.3.3.2.2. Diseños de los módulos a desarrollar.

Boceto Anexos Transaccional

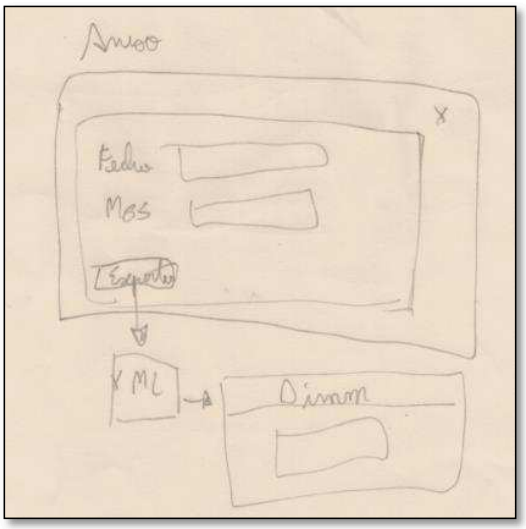


Ilustración 13 - Boceto Anexos Transaccionales
Fuente: Elaboración propia

Boceto de Opciones en Contabilidad

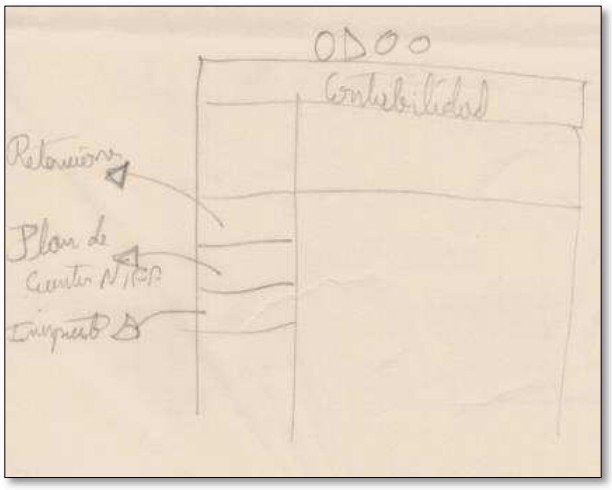


Ilustración 14 - Boceto de Opciones en Contabilidad
Fuente: Elaboración propia

Venta.

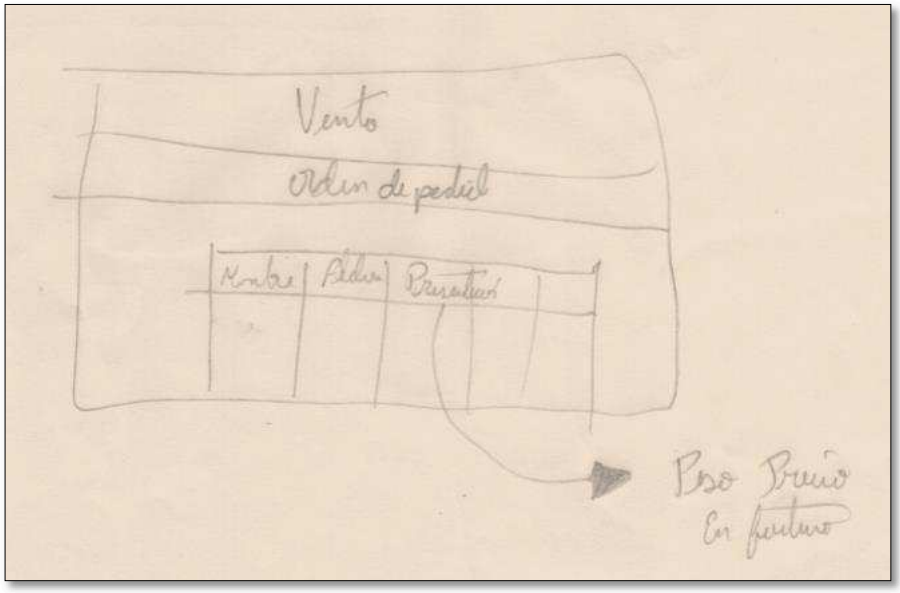


Ilustración 15 - Boceto de Ventas
Fuente: Elaboración propia

Compra

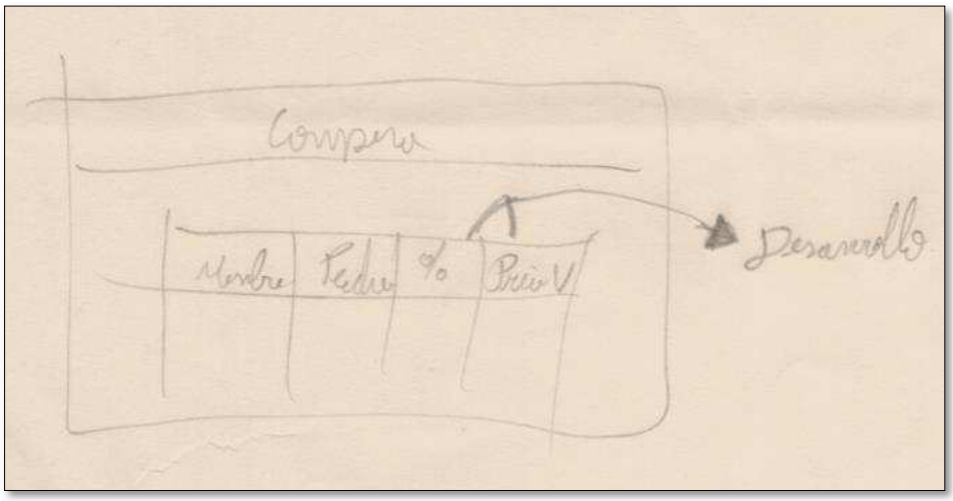


Ilustración 16 - Boceto de Compras
Fuente: Elaboración propia

Peso y Precio en Productos

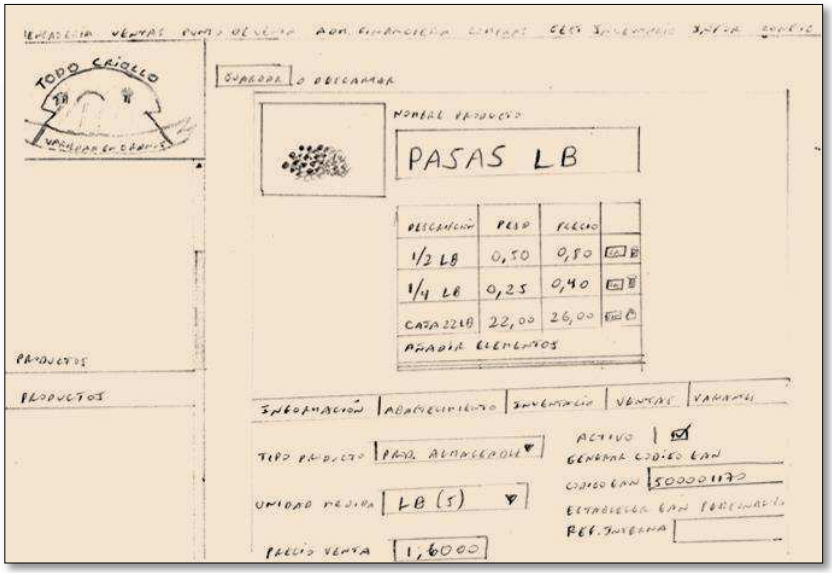


Ilustración 17 - Boceto de Peso y Precio en Productos
Fuente: Elaboración propia

Exportar Listado

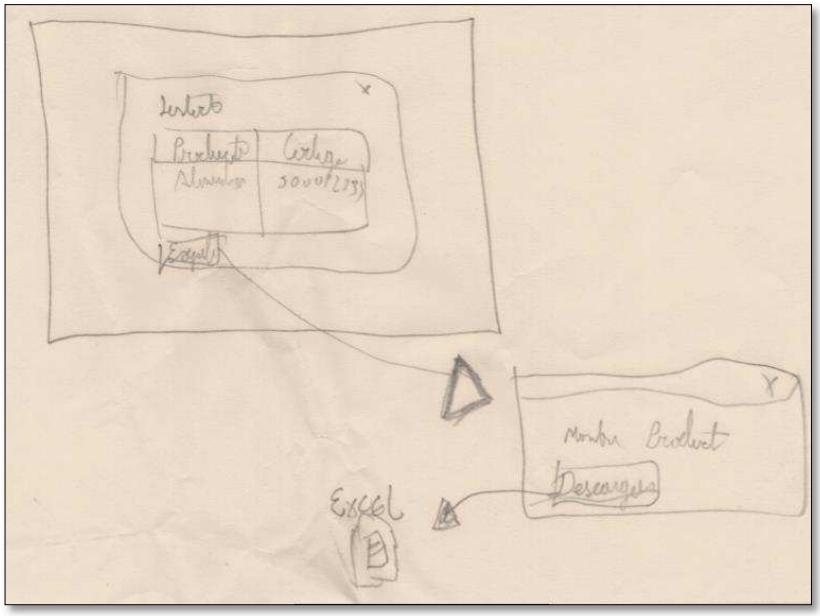


Ilustración 18 - Boceto Exportar Listado
Fuente: Elaboración propia

3.3.3.2.3. Explicación de la arquitectura Odoo.

La arquitectura de Odoo al ser cliente servidor, permite a todos los usuarios de la empresa trabajen sobre los mismos datos, esto ayuda que la información este siempre disponible para su visualización, por otro lado, contribuye a que las máquinas clientes tengan un menor consumo de recurso en su procesamiento. Este intercambio que Odoo maneja entre el servidor y el cliente se realiza mediante XML-RCP y J-son.

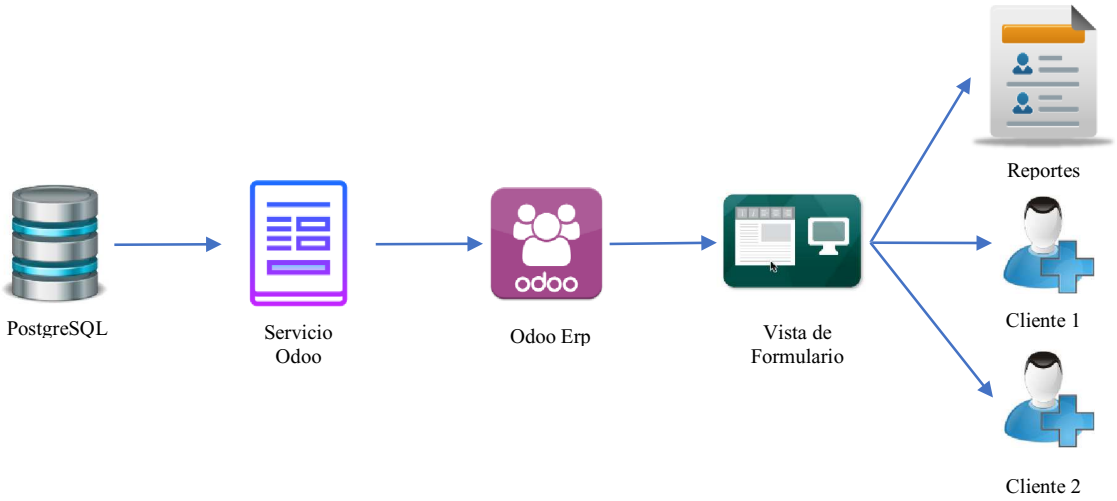


Ilustración 19 - Arquitectura Odoo
Fuente: Elaboración propia

3.3.3.2.4. Modelo de la base de datos.

Listado de tablas que contendrán los principales datos de la implementación y los datos de los módulos desarrollados.

Tabla 13 - Listado de tablas de base de datos
Fuente: Elaboración propia

Tabla	Descripción
Product_product	Los datos principales de los productos
Product_template	Los datos detallados de los productos los cuales pueden ser más extensos y relacionados con otras tablas
Product_presentacion	Tabla que contiene los datos de los pesos y precios de los productos que son tomados en cuenta en el módulo Peso_precio.
Producto_imprimir	Contiene los datos de cuando exportamos los productos para la impresión de etiquetas.
Export_producto	Guarda los datos de peso precio al momento de imprimir

Product_category	Los datos de esta tabla son las variantes que los productos pueden tener.
Pos_order_line	Detalle de las ordenes que se dan al momento de la venta en el punto de venta
Pos_order	Datos principales del punto de venta
Res_partner	Datos del cliente o proveedor

3.3.3.2.5. Diagrama entidad-relación.

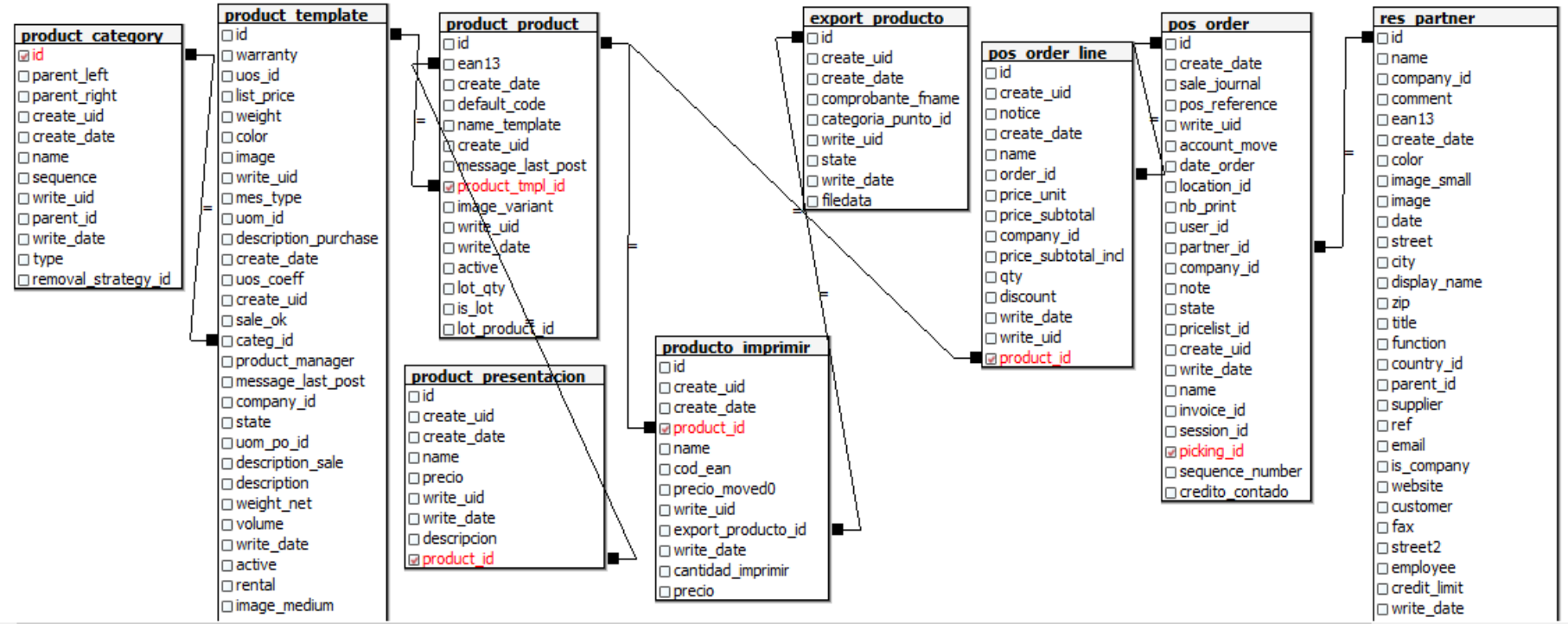


Ilustración 20 - Modelo entidad-relación de la base de datos
Fuente: Elaboración propia

3.3.3.3. Sprint 3. Del 08 de mayo al 02 de junio

3.3.3.3.1. Reunión de planificación del sprint 3

En el sprint número tres, la reunión se basará en la definición de las tareas para la creación del aplicativo en las diferentes plataformas, ya sean los módulos adicionales, la aplicación móvil del catálogo de productos y la página web informativa.

Asistentes:

- Cedeño Cedeño Bryan (Scrum team) dueño del producto.
- Villacreses Lucas Jeffri (Scrum team) desarrollador.
- Rafael Palacios (Stackholder), cliente.

Actividades:

3.3.3.3.2. Desarrollo de módulos adicionales

Luego de tener claro las especificaciones de los requisitos que la empresa necesita se dan a relucir los módulos adicionales y los complementos de los módulos ya instalados.

Listado de los complementos a desarrollar**:**

- **Peso_precio:** Complemento que me permita establecer el peso de diferentes medidas de los productos sin que me afecte el inventario o tenga que crear otro ítem o producto.

- **Exportar_producto:** Modulo que me permita imprimir o exportar un documento que por medio del programa Bartender me permita imprimir los productos de la base de datos.
- **Pos_arreglos:** Complemento en el punto de venta, poniendo acorde a los datos requeridos al crear un nuevo cliente ya que necesitamos varios datos que Odoo no pide para poder llevar una correcta contabilidad.
- **Ln_10_ec:** Modulo que nos permite implementar gran parte de cambios sobre la contabilidad en Ecuador.
- **Codigo_impuesto:** Modulo que permite configurar los impuestos y sus códigos para obtener información para declaraciones al servicio de rentas internas.
- **Reportes_ecuador:** Modulo que configura las facturas, las retenciones y los cheques lo cual quedan acorde a lo requerido por la empresa.

3.3.3.3.3. Desarrollo de aplicativo móvil

Dado como un aporte más para la empresa se procedió promover la creación de una aplicación móvil que nos permita tener actualizados los productos más significativos para la empresa y que los precios de este producto estén a tiempo real.

Esta aplicación se la realizara en Android Studio, para los diferentes dispositivos Android, que permitirá a los diferentes usuarios por medio de internet conocer los precios de los productos que ellos deseen comprar.

3.3.3.3.4. Desarrollo de página web

La creación de la página web se da debido a las circunstancias de la implementación del marketing digital y de una manera de promocionar nuestros productos, por medio de una página informativa propia, la cual poseerá los productos principales que la empresa provee y una serie de reseñas.

Sera desarrollada por medio del framework WordPress, alojado en un hosting, que está en chicago, adquirido principalmente para este fin.

Ilustración 21 - Diseño de página web
Fuente: Elaboración propia

3.3.3.4. Sprint 4. Del 05 de junio al 23 de junio

3.3.3.4.1. Reunión de planificación del sprint 4

En esta reunión se distribuirán las tareas, las cuales están enmarcadas en la instalación de la red local para que el sistema funcione correctamente y la instalación e implementación del servidor Odoo en conjunto con los módulos adicionales desarrollados.

Asistentes:

- Cedeño Cedeño Bryan (Scrum team) dueño del producto.
- Villacreses Lucas Jeffri (Scrum team) desarrollador.
- Lucas Villacreses Erick (Stackholder), consultor externo.

Actividades:

3.3.3.4.2. Bosquejo e implementación de la red local.

Dados los requerimientos de la plataforma Odoo al ser un entorno web y con los requisitos provisto por el cliente, bosquejamos la red de esta forma, teniendo en cuenta el uso de una Ip publica para el uso exterior del sistema por parte del gerente de la empresa.

Para las conexiones se usaron cable UTP de categoría 5, con un Switch D-link para la distribución de la red; un router y Access Point facilitados por la empresa proveedora de internet marca Cisco. El uso de canaletas adhesivas para las paredes del local y Jacks rj45 para el soterramiento de ciertos cables.

La red de manera interna fue distribuida en una subred para mayor seguridad, registrando cada dispositivo electrónico que quiera conectarse a la red debe pedir permiso al administrador de la red, debido al uso de la identificación única de cada dispositivo como es la MAC. Una vez finalizada la adecuación de la red de manera local y externa quedo en estas condiciones:

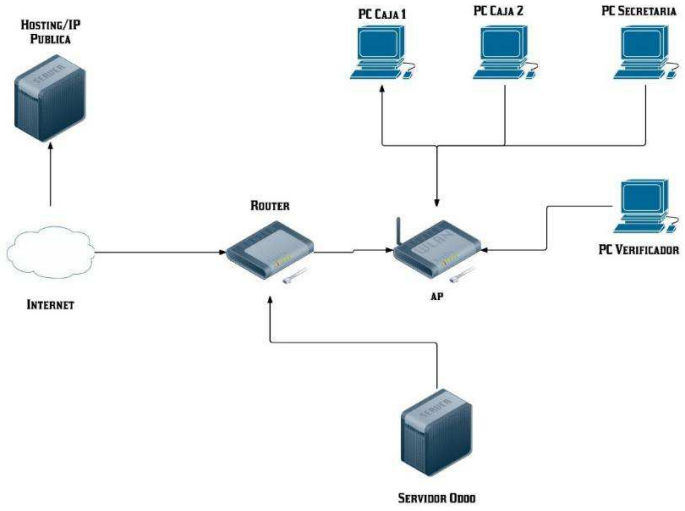


Ilustración 22 - Modelo de red en instalaciones de Todo Criollo
Fuente: Elaboración propia

3.3.3.4.3. Descripción del hosting adquirido.

En la búsqueda de un proveedor que nos permita alojar nuestra página web, y respaldos de nuestra base de datos, recurrimos a siteground, el cual nos ofrece una disponibilidad muy buena y con una ayuda técnica disponible las 24 horas durante todo los días del año, siendo este uno de los servidores más rápidos y confiables en el momento, al contar con lo necesario y ayudarnos a configurar de manera rápida y eficiente nuestro sitio web, en conjunto con un correo institucional procedimos al contrato por un año de esta plataforma.

Ilustración 23 - Logotipo de Hosting

3.3.3.4.4. Implementación de los módulos base de Odoo.

Luego de la adquisición de los equipos necesarios y la adecuación de la red para el uso del sistema, se procedió a la instalación de los requerimientos para que el ERP Odoo funcione.

3.3.3.4.4.1. Instalación de Odoo.

El servidor contara con un sistema operativo Linux en forma base, la instalación de Odoo fue por líneas de comando con un entorno tipo Core. Los detalles de la instalación serán expuestos en el manual técnico del proyecto.

La accesibilidad al servidor solo será por medio de SSH o por el puerto web el número 8069 de manera interna y 80 de manera externa, la base de datos postgres estará accesible de manera externa por medio del puerto 5432 que servirá para configuraciones directas.

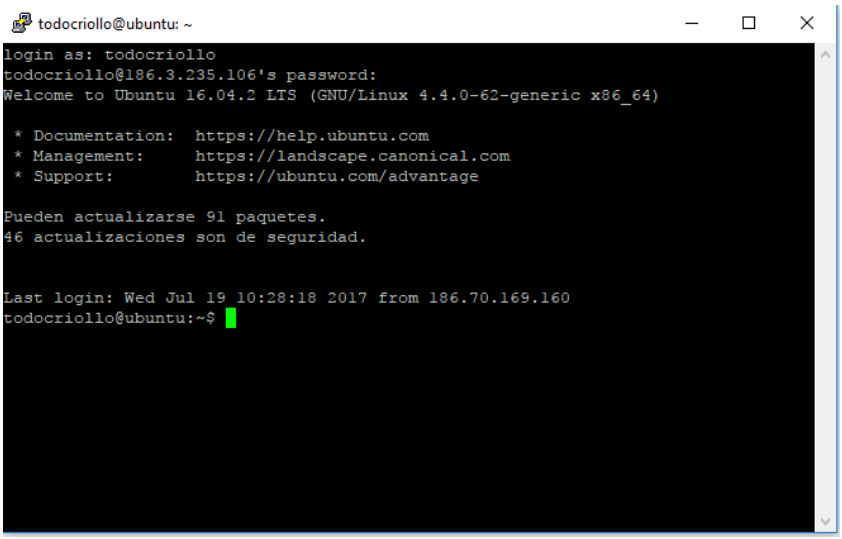


Ilustración 24 - Instalación de Odoo
Fuente: Elaboración propia

3.3.3.4.4.2. Instalación de Módulos principales.

Los módulos provistos por Odoo que vamos a utilizar serán:

Ventas. – Nos permitirá que la encargada de facturación pueda emitir los documentos necesarios para la venta y se instalara también el punto de venta, por lo tanto, podremos vender por caja.

Ilustración 25 - Modulo de ventas
Fuente: Elaboración propia

Compras. - Este módulo nos permitirá llevar un control de las adquisiciones de la empresa y las deudas que tendremos con los proveedores.

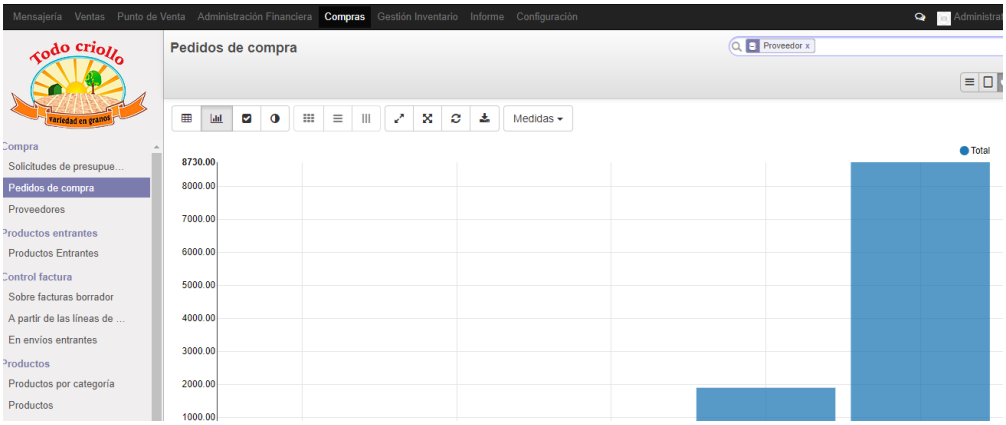


Ilustración 26 - Modulo de compras
Fuente: Elaboración propia

Administración Financiera. – Con este módulo se instalarán las bases de la contabilidad en Odoo partiendo de esto podremos configurar y desarrollar complementos para poder configurar Odoo para la legislación ecuatoriana en el tema de contabilidad.

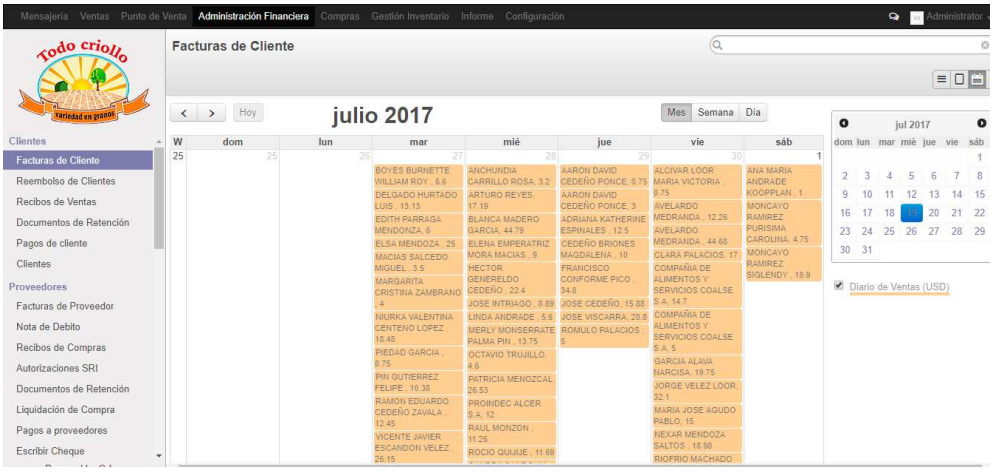


Ilustración 27 - Modulo de Administración financiera
Fuente: Elaboración propia

Gestión de inventario. - El módulo de inventario nos contribuirá a llevar de manera ordenada y eficiente las transacciones las cuales alimentaran la bodega para saber la existencia de los productos en tiempo real.

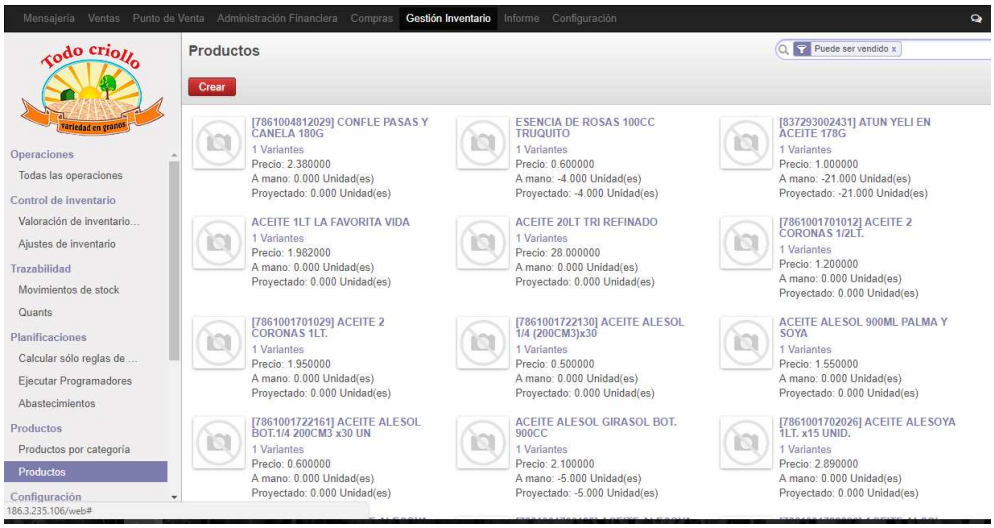


Ilustración 28 - Modulo Gestión de Inventarios
Fuente: Elaboración propia

3.3.3.4.5. Instalación de módulos desarrollados y configuración.

Para el correcto funcionamiento del ERP en la empresa, se dio la necesidad de desarrollar complementos para cumplir con los requerimientos de la empresa, desde el manejo de los productos, la impresión de código Ean, las adecuaciones en el punto de venta y complementos para la contabilidad.

Modulo Peso Precio

Permite al usuario controlar, la forma en la que se pueden vender las variantes de los diversos productos sin afectar al inventario, sin crear nuevos productos, solo con poner el peso se calcula automáticamente un precio sugerido modificable a conveniencia.

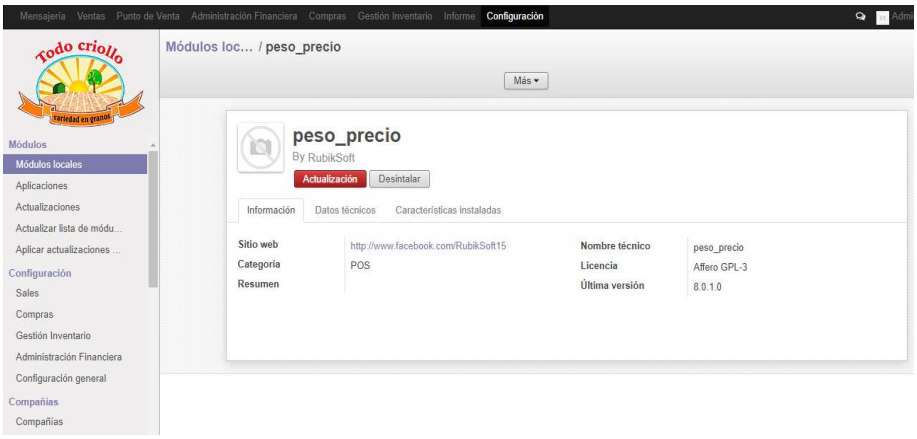


Ilustración 29 - Modulo Peso Precio
Fuente: Elaboración propia

Módulo de exportación de código Ean.

Le facilita al usuario la impresión de etiquetas para los productos, definiendo en la etiqueta el peso del producto para que los lectores de barras, entiendan que es una variante del producto, calculen correctamente los precio y controlen la salida de los productos por caja.

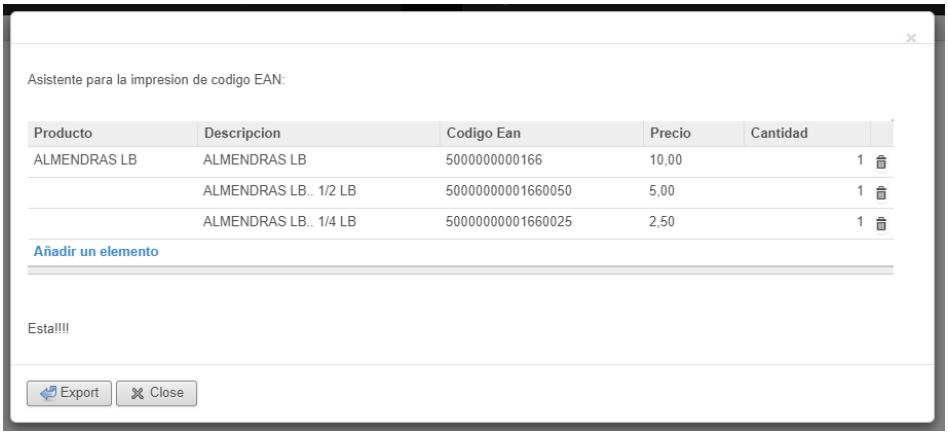


Ilustración 30 - Módulo Exportación Código EAN
Fuente: Elaboración propia

Módulo de contabilidad ecuatoriana

Permitirá el uso de todos los parámetros para el uso de la contabilidad en el país, con un plan de cuentas actualizado a la NIFF y exportar un XML que permita ser usado en el dimm para la declaración mensual.

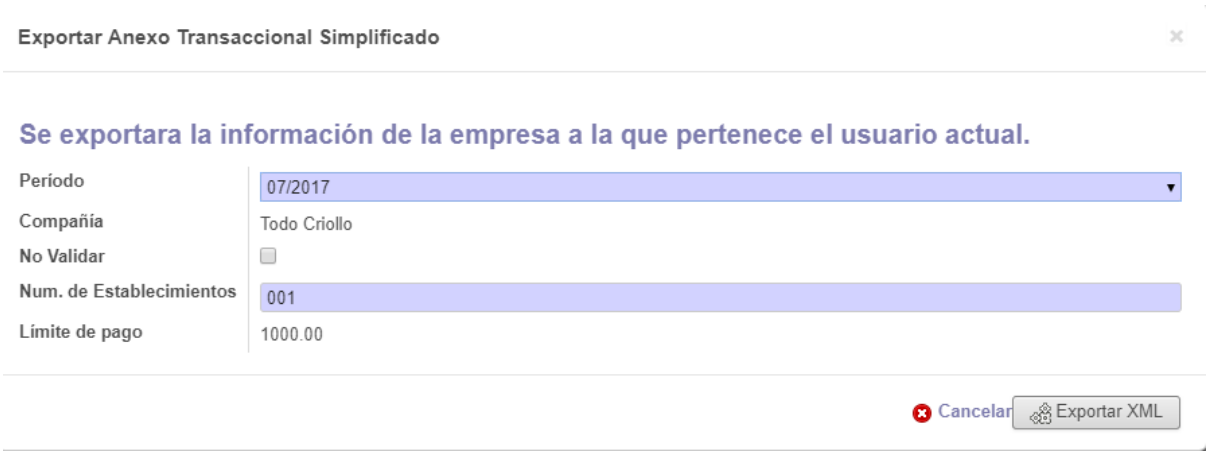


Ilustración 31 - Modulo Contabilidad ecuatoriana
Fuente: Elaboración propia

3.3.3.5. Sprint 5. Del 26 de junio al 07 de julio

3.3.3.5.1. Capacitaciones a los miembros de la empresa.

Plan de capacitación

La capacitación, es una medida que se aplica de manera organizada, para generar conocimientos hacia los empleados de la empresa en donde nos encontramos, esta nos ayudara a contribuir a la ejecución del proyecto para su mejor entendimiento por parte de los usuarios del sistema.

Con este plan de capacitación divido según los procesos que van a intervenir se capacitaron a siete personas, estas serán las encargadas de administrar el sistema en su totalidad.

En el anexo E se detallarán la constancia de las capacitaciones.

Tabla 14 - Plan de capacitaciones
Fuente: Elaboración propia

TODO CRIOLLO **Estructura del plan de capacitación**				
Área	Temas	Participantes	Fecha	Encargado
Ventas	Punto de venta, cobros, venta crédito, presupuestos, pedidos de ventas	Julia Véliz, Rafael Palacios, Julia Palacios	26 de junio	Bryan Cedeño
Bodega	Ajuste de inventario, alimentación de inventario, traspaso de bodega, creación de productos	Darwin Cedeño, Ramon Chávez	26 de junio	Jeffri Villacreses
Compras	Órdenes de compra, pago a proveedores,	Marisela Saltos	27 de junio	Jeffri Villacreses
Contabilidad	Asientos diarios, anexo transaccional, ajuste, impuestos, facturas a cliente y proveedores.	Genny Lucas, Marisela Saltos	28 junio	Bryan Cedeño
Empacado	Impresión de etiquetas, exportación de archivo xls	Mayra Véliz	28 junio	Bryan Cedeño

3.3.3.5.2. Campaña de publicidad.

Facebook.

Página Principal

Ilustración 32 - Campaña de publicidad en Facebook
Fuente: Elaboración propia

Publicaciones en campaña

Ilustración 33 - Publicación de campaña en Facebook
Fuente: Elaboración propia

Resultados de la campaña

Ilustración 34 - Resultados de campaña en Facebook
Fuente: Elaboración propia

Ilustración 35 - Representación gráfica de resultados
Fuente: Elaboración propia

Capitulo IV

Evaluación de resultados

4.1. Introducción

Los sistemas informáticos en las empresas deben ser capaces de responder ante cualquier situación, el monitoreo de estos sistemas ha instado a personas a trabajar en toda la infraestructura de TI para notificar problemas, prevenir incidencia y aprovecha los recursos de forma óptima, en general se podría decía cuantificar resultados. Obtenidos una vez estos resultados por medio de técnicas de recolección, procedimos a mostrarlos.

El capítulo se centra en la comparación de los tiempos obtenidos por medio de la recolección de información, demostrando así la eficiencia del sistema implantado, con mejoras a futuro para un mejor crecimiento de la empresa. Comparamos cuatro procesos principales de la empresa con la automatización en Odoo, procesos en sistemas de información anteriormente usados por la empresa y la forma manual que se puede emplear, obteniendo que los procesos en el sistema Odoo mejoraron exponencialmente en ocasiones los procesos descritos en el planteamiento del problema.

4.2. Seguimiento y Monitoreo de resultados

El efecto y resultado con mayor representatividad se concreta en la presente investigación y desarrollo de la propuesta para implementar un sistema ERP que atienda la gestión de los procesos

de negocio de una Pyme, caso de estudio Todo Criollo, representando así lograr una correcta administración en corto tiempo y con bajos recursos.

El uso de la metodología Scrum nos permite la implementación del Sistema de una forma integral y ordenada, logrando así un correcto desarrollo en las actividades planificadas.

Los módulos relacionados a la gestión de los procesos de negocio para la empresa, se deben establecer y adaptar de la mejor manera. Dado que el desarrollo de la metodología es incremental, por lo tanto, plantea que en la siguiente iteración se atienda otro proceso del negocio que agregue valor a la empresa; para ello se han integrado nuevas funcionalidades en el sistema que permite la ejecución y cumplimiento de los objetivos planteados.

Con los resultados obtenidos en el apartado anterior, podemos afirmar que efectivamente la utilización del Sistema de Planificación de Recursos Empresariales Odoo produce efectos positivos y resultados favorables al mejorar el desempeño en los procesos de negocio para la empresa.

4.2.1. Evaluación de resultados por observación.

Para la evaluación de la implementación del sistema de planificación empresarial, procedimos a determinar la eficiencia en los tiempos de ejecución en los procesos descritos con anterioridad por medio de técnicas de observación que nos permiten, tomar los datos y verificar cuanto

demoraría el proceso manual para realizar una tarea. Por medio de cuadros comparativos procedemos a detallar los resultados.

Administración contable

Tabla 15 - Detalle comparativo de Administración Contable
Fuente: Elaboración propia

Proceso	Usuario	Proceso Manual	Herramientas Tecnológicas	Proceso en Odoo
Anexo Transaccional	Genny Lucas	6 días	Sin uso	60 Seg.
Devoluciones	Marisela Saltos	800 Seg.	Sin uso	540 Seg.
Búsqueda de Retenciones	Genny Lucas	240 Seg.	Si uso	25 Seg.
Impresión de Cheques	Marisela Saltos	120 Seg.	30 Seg	20 Seg.
Declaración Mensual	Genny Lucas	3 horas	No aplica	800 Seg.

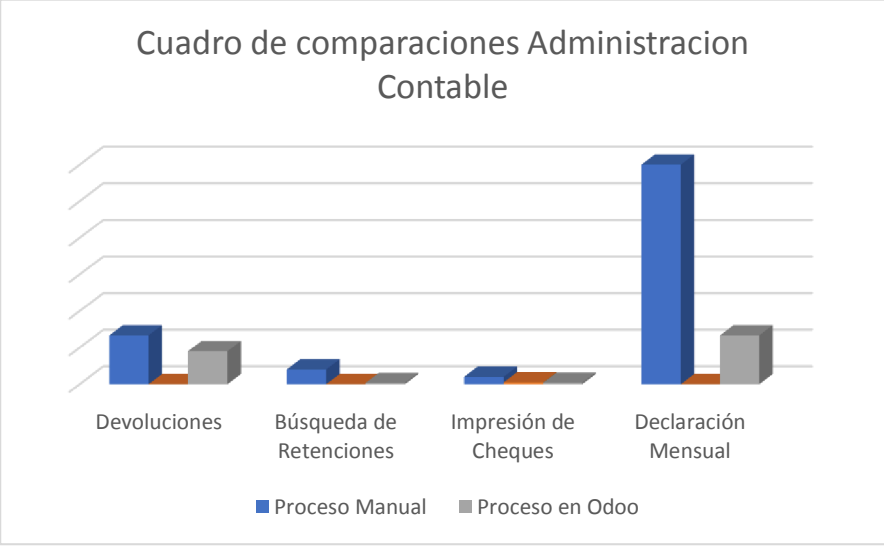


Ilustración 36 - Cuadro comparativo de Administración Contable
Fuente: Elaboración propia

Análisis. – Podemos observar en el gráfico y tabla, procesos en los que intervenimos con la implementación del sistema de planificación empresarial en la evaluación de estos procesos se consideró tres aristas para la comparación, procesos manuales, procesos en herramientas complementarias y procesos en Odoo, partiendo esta comparación del proceso de administración financiera. Los procesos manuales denotamos que poseen tiempos de esperas y ejecución largos como el caso del anexo transaccional donde la contadora tarda días para realizarlo, por falta de capacitación y carencia de instrumentos contables en las herramientas tecnológicas, mientras el módulo Odoo nos arrojó datos aceptables y una mejoría notoria en los tiempos de ejecución de ciertos procesos, como el anexo que demora un minuto en realizarlo.

Compras

Tabla 16 - Detalle comparativo de proceso de Compras
Fuente: Elaboración propia

Proceso	Usuario	Proceso Manual	Herramientas Tecnológicas	Proceso en Odoo
Órdenes de Compra	Marisela Saltos	No Aplica	600 Seg	450 Seg
Ingreso de Compra	Marisela Saltos	1200 Seg	No aplica	500 Seg
Cuentas por pagar de Proveedores	Marisela Saltos	No aplica	240 Seg	100 Seg

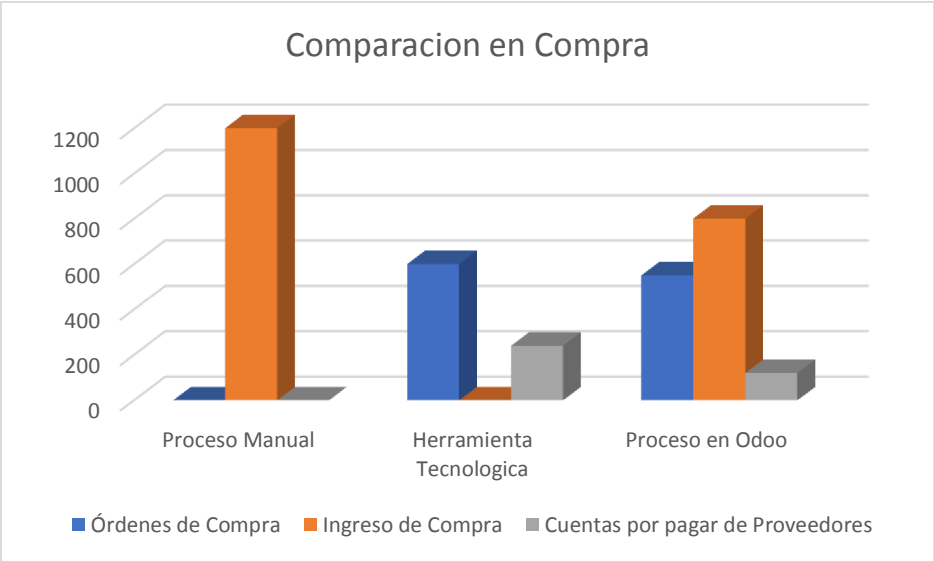


Ilustración 37 - Grafico comparativo de proceso de Compras
Fuente: Elaboración propia

Análisis. – En la comparación de los procesos en compra, pudimos ver déficit al momento del cobro de deudas, por el detalle de ciertas facturas de proveedores existe discordancia con el dinero ingresado, de manera manual este proceso y algunos componentes de herramientas tecnológicas genera confusión dando tiempos de ejecución de tareas de hasta 20 minutos en la realización de ingreso de compras, 10 minutos en proceso de orden de compra y hasta 4 minutos en las búsquedas de proveedores por pagar. Odoo nos arrojó resultados con tiempos menores al proceso manual y la herramienta tecnológica, se establecieron normas para el control del ingreso de las compras y el pago de facturas, con estas mejoras al proceso de compra hubo menos tiempo en la ejecución de tareas como órdenes de compra con un tiempo 7,5 minutos, el ingreso de compras con un tiempo de 8,33 minutos y la búsqueda de facturas por pagar con un tiempo de 1,8 minutos.

Ventas

Tabla 17 - Detalle comparativo de proceso de Ventas
Fuente: Elaboración propia

Proceso	Usuario	Proceso Manual	Proceso en Genesis	Proceso en Odoo
Punto de Venta	Julia Palacios	1200 Seg	700 Seg	400 Seg
Pedidos	Marisela Saltos	No Aplica	600 Seg	300 Seg
Cuentas por cobrar Clientes	Marisela Saltos	600 Seg	400 Seg	120 Seg

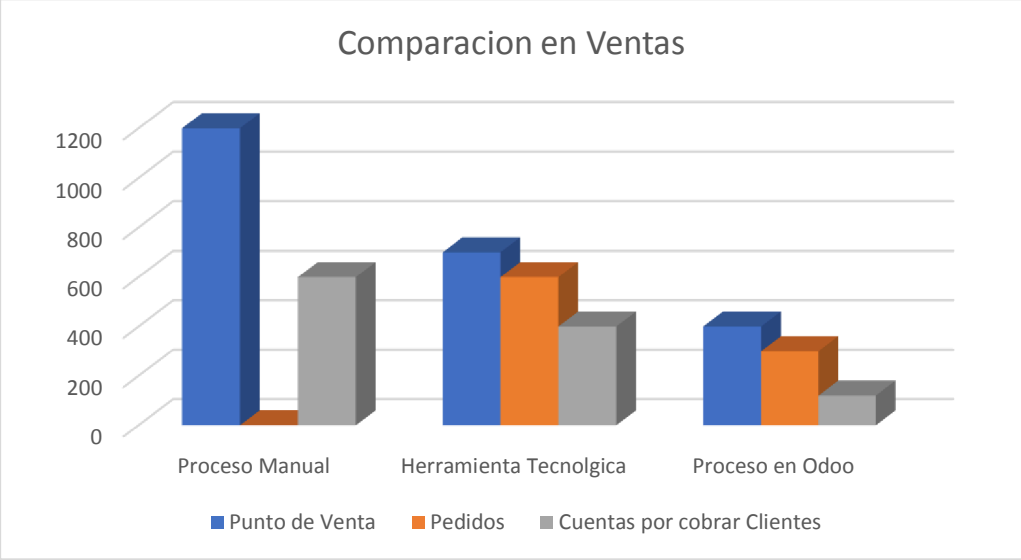


Ilustración 38 - Grafico comparativo de proceso de Ventas
Fuente: Elaboración propia

Análisis. – Las ventas en la empresa estuvieron enmarcadas en la desatención al cliente por falta de optimización en la caja de cobro por la demora en el momento de pasar por lectora cierto productos empacados por la empresa, teniendo un tiempo de ejecución de esta tarea de hasta 20 minutos de forma manual y hasta 11 minutos con herramienta tecnológica, la implementación del

sistema de planificación y el desarrollo de un módulo que permita sacar el código para poder imprimir las etiquetas y el uso del sistema Odoo nos dieron un resultado de 7 minutos promedio por venta, en los pedidos de clientes también se redujo hasta 5 minutos.

Con los resultados obtenidos luego de la comparación de como estaba antes y como está ahora los procesos, que por medio del sistema se automatizó y mejoró, nos da una disminución en los tiempos de ejecución de las tareas cotidianas de los empleados, aportando a la buena atención al cliente y aumentando la rentabilidad del negocio.

Conclusiones

- Se logro diagnosticar la situación de la empresa en temas de procesos de negocios, con la cual se elaboró la propuesta para dar solución al problema planteado.
- El uso de la metodología aplicada en el estudio permitió obtener las bases teóricas para alcanzar resultados contundentes sobre la implementación de sistemas de información empresariales y lo determinante fue el apoyo de los empleados de la empresa en la concepción del proyecto.
- Scrum como metodología de desarrollo es válido para la implementación de un ERP, sus virtudes en dividir tareas y roles, permitió direccionar las fases del proyecto para la entrega a tiempo de avances al final de cada iteración.
- Odoo como sistema de planificación empresarial, debido a su característica de flexibilidad y modularidad al momento de desarrollar, se adaptó a las necesidades descritas en las historias de usuarios recopiladas en la empresa.
- El uso de aplicativos móviles en la integración con el ERP, permite el acercamiento al cliente de manera más eficiente ayudándolo a conocer los productos ofrecidos por la empresa y contribuyendo a la campaña de social media realizada.

Recomendaciones

- Utilizar la metodología de desarrollo de software SCRUM, ya que cuenta con los principios y artefactos que permiten que el desarrollo sea interactivo, tener una mejora relación con el cliente final, y dividir las tareas entre los integrantes, estimar tareas importantes, realizar cambios una vez el proyecto por medio de la retroalimentación.

- El uso de software libre como framework de desarrollo, Odoo es una alternativa a lo convencional de emprender el desarrollo de un producto o sistema desde cero, se asume un desarrollo libre al usar ideas y mejorarlas para presentar un producto con calidad.

- Se recomienda realizar la utilización de herramientas que permitan tener la función como repositorio y control de versiones de código fuente como por ejemplo GitHub, mismo que permite realizar un trabajo colaborativo, y a la vez un adecuado manejo de las versiones de un proyecto.

Bibliografía

Balaguera, Y. D. (2013). Metodologías ágiles en el desarrollo de aplicaciones . *Universidad Tecnologica de Colombia*, 112-113.

Calvo, W. J. (29 de Noviembre de 2013). *Abcomweb*. Obtenido de Abcomweb: http://www.abcomweb.com/seo/que-significa-seo.php

Catt. (05 de Septiembre de 2011). *Informatica para ti*. Obtenido de Informatica para ti: http://informaticapara-tu.blogspot.com/2011/09/que-es-adwords.html

Comunitea. (06 de Mayo de 2016). *Comunitea*. Obtenido de Comunitea: http://comunitea.com/que-es-odoo/

Constitucion, A. (2008). *Constitución del Ecuador.* Quito.

Dataprix. (12 de Marzo de 2014). *Dataprix*. Obtenido de Dataprix: http://www.dataprix.com/articulo/erp/cual-origen-erp-invento-militar-software-imprescindible-las-empresas

Definicion Social Media. (06 de Mayo de 2015). *Genwords*. Obtenido de Genwords: https://www.genwords.com/blog/social-media-marketing#Social_Media_Marketing_que_es_este_fenomeno

González, A. (20 de Agosto de 2014). *Tu Programación*. Obtenido de Tu Programación: http://www.tuprogramacion.com/glosario/que-es-un-orm/

Lara Martinez , O. R. (02 de Diciembre de 2011). *Gestiopolis*. Obtenido de Gestiopolis: https://www.gestiopolis.com/erp-planificacion-de-recursos-empresariales/

López, Á., Quer, A., & Valdés, P. (2014). *Social Selling.* Madrid.

Manifiesto Agil. (13 de Noviembre de 2012). *Agile Manifiesto.* Obtenido de Agile Manifiesto: http://www.agilemanifesto

Marker, G. (24 de Abril de 2016). *Easytechnow*. Obtenido de Easytechnow: http://easytechnow.com/learn-technology/what-are-erp-systems/

Martín, A. D. (2013). *Proyectos ágiles con Scrum* . Buenos Aires: Kleer.

Méndez, C. (2010). *Diseño y Desarrollo del Proceso de Investigacion.* Mexico: Shalom.

Menzinsky, A., & Palacio, J. (2016). *Scrum Manager.* Valencia: Safe Creative.

Morales Mejía, C. A., & Auquilla Buñay, J. A. (2015). *Implementación del sistema ERP social en la Unidad Educativa Intercultural La Paz y el Despacho Parroquial de Cacha de la ciudad de Riobamba y reingeniería del sistema.* Quito.

Odoo. (06 de Diciembre de 2013). *Odoo.* Obtenido de Odoo: http://fundamentos-de-desarrollo-en-odoo.readthedocs.io/es/latest/capitulos/qweb-creando-vistas-kanban-reportes.html

OdooMrp. (02 de Junio de 2012). *OdooMrp.* Obtenido de OdooMrp: http://www.odoomrp.com/en_US/page/preguntas-frecuentes

Paredes Guerrero, C. M., & Cabrera Gallardo, U. A. (2012). *análisis, diseño, desarrollo e implementación de un erp(Enterprise resource planning) "Acsoft" de los módulos administrativo y contable para la empresa disprolim dedicada al sector industrial y químico.* Quito.

Plan del Buen Vivir. (2013). *Plan del Buen Vivir.* Ecuador.

Plan Nacional de Gobierno Electronico. (2014). *Plan Nacional del Gobierno Electronico.* Quito.

Plata, U. N. (20 de Enero de 2014). *unpl.* Obtenido de unpl: http://unlp.edu.ar/

Praxya. (13 de Octubre de 2013). *Praxya*. Obtenido de Praxya: http://praxya.com/soluciones/odoo-open-erp

Rivero, D. S. (2008). *Metodologia de la Investigacion.* Madrid: Shalom.

Romero, Y. F. (2012). Patrón Modelo-Vista-Controlador. *Telematica*, 2-4.

Sabino, C. (1996). *El proceso de investigacion.* Buenos Aires: Lumen-Humanitas.

Sampieri, R. H. (2010). *Metodologia de la Investagación.* Mexico: McGrawHill.

SantaMaría, D. F. (2015). *IMPLEMENTACIÓN Y REINGENIERÍA DEL SISTEMA ERP.* Quito: Universidad Central del Ecuador.

Scielo. (2016). *Adopción del XML.* Madrid: Solange Santos.

Solutum. (05 de Febrero de 2016). *Solutum*. Obtenido de Solutum: http://www.solutum.com/que-es-seo-sem-significado/

Tobar, A. N. (2011). *Implantación de una herramienta ERP software.* Imbabura.

Anexos

Anexo A Encuesta.

ENCUESTA (población)

Pregunta 1. ¿En qué rango de edad se encuentra?

15 a 30 31 a 64

Pregunta 2: ¿Cuál canal de comunicación es el que más usa usted?

Internet	
Televisión	
Radio	
Medios escritos	

Pregunta 3. ¿Qué sectores del Cantón Manta de usted frecuenta para realizar compras de víveres, granos y frutos secos?

Centro de Manta	
Nuevo Tarqui	
Los Esteros	
Otros	

Si Usted ha visitado el centro de Manta, conteste la siguiente pregunta.

Pregunta 4. ¿Cuáles de los siguientes comerciales frecuenta en el centro de Manta?

Supermercado Galli	
Comercial "El Arbolito"	
Supermercado TIA	
Otros	

Si Usted ha visitado el comercial "Todo Criollo", conteste las siguientes preguntas.

Pregunta 5. ¿Cuáles de las siguientes opciones usted considera que se puede aplicar en la empresa, para mejorar la atención al usuario?

Personal capacitado	
Mas cajas habilitadas	
Un mejor sistema de cobro	
Equipos nuevos	

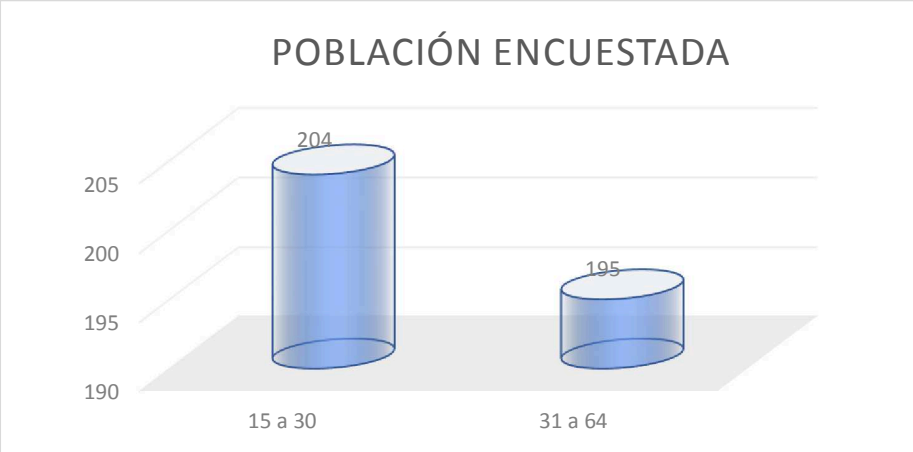
Pregunta 6. De los siguientes enunciados califique el servicio que brindan el comercial "Todo Criollo".

Excelente	
Bueno	
Regular	
Ineficiente	
Falta mejorar	

Anexo B resultado de encuestas.

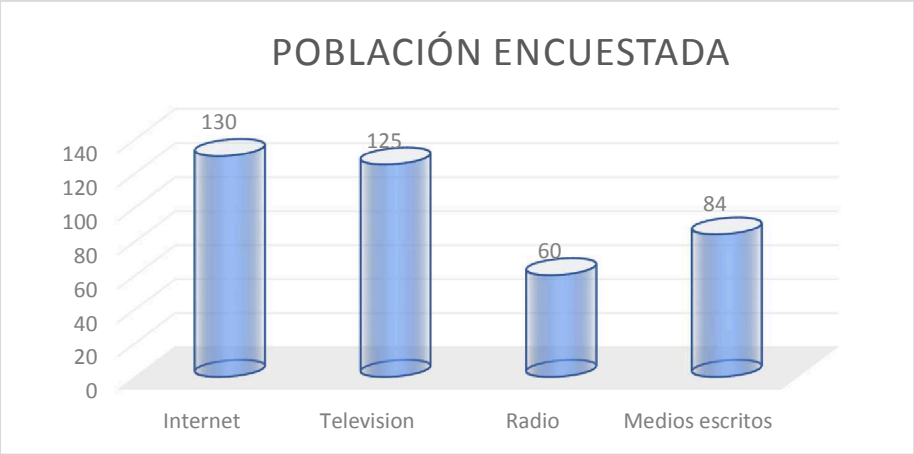
Pregunta # 1: ¿En qué rango de edad se encuentra?

VARIABLE	Población	
	Absoluta	*Relativa %*
15 a 30 años	204	51%
31 a 64 años	195	49%
Total	**399**	**100%**



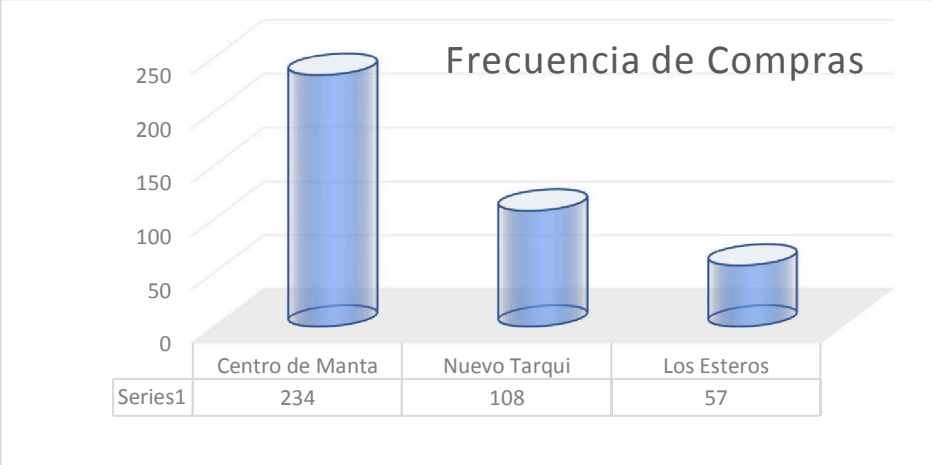
Pregunta # 2: ¿Cuál canal de comunicación es el que más usa usted?

VARIABLE	Población	
	Absoluta	*Relativa %*
Internet	130	33%
Televisión	125	31%
Radio	60	15%
Medios escritos	84	21%
Total	**399**	**100%**



Pregunta 3. ¿Qué sectores del Cantón Manta de usted frecuenta para realizar compras de víveres, granos y frutos secos?

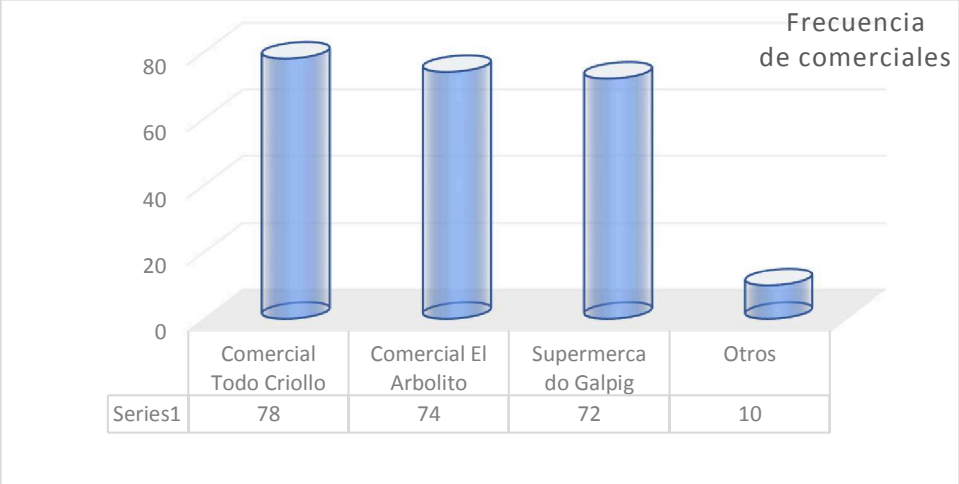
Cantón	Población	
	Absoluta	*Relativa %*
Centro de Manta	234	59%
Nuevo Tarqui	108	27%
Los Esteros	57	14%
Total	**399**	**100%**



Si Usted ha visitado el centro de Manta, conteste la siguiente pregunta.

Pregunta 4. ¿Cuáles de los siguientes comerciales frecuenta en el centro de Manta?

Cantón	Población	
	Absoluta	*Relativa %*
Comercial Todo Criollo	78	33%
Comercial "El Arbolito"	74	32%
Supermercado Galli	72	31%
Otros	10	4%
Total	**234**	**100%**

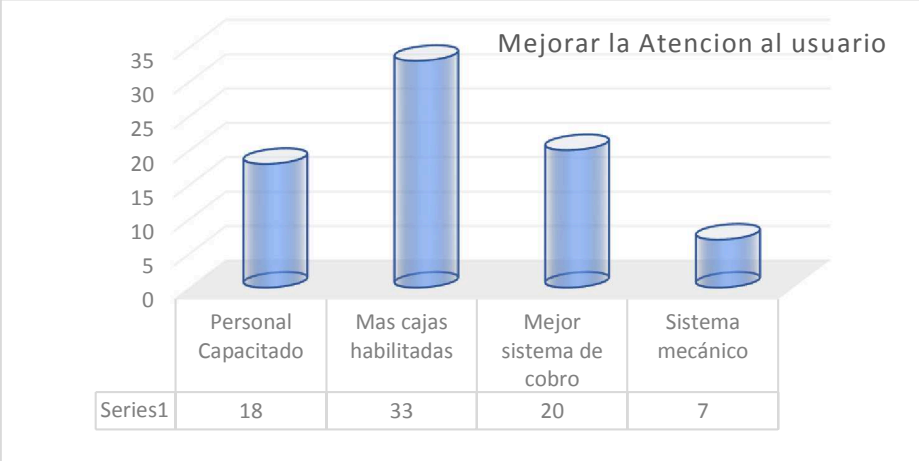


Si Usted ha visitado el comercial "Todo Criollo", conteste las siguientes preguntas.

Pregunta 5. ¿Cuáles de las siguientes opciones usted considera que se puede aplicar en la empresa, para mejorar la atención al usuario?

Cantón	Población	
	Absoluta	*Relativa %*
Personal Capacitado	18	23%
Mas cajas habilitadas	33	42%

Mejor sistema de cobro	20	26%
Equipos nuevos	7	9%
Total	**78**	**100%**



Pregunta 6. De los siguientes enunciados califique el servicio que brindan el comercial "Todo Criollo".

Cantón	**Población**	
	Absoluta	*Relativa %*
Excelente	3	4%
Bueno	6	8%
Ineficiente	21	27%
Regular	23	29%
Falta mejorar	25	32%
Total	**78**	**100%**

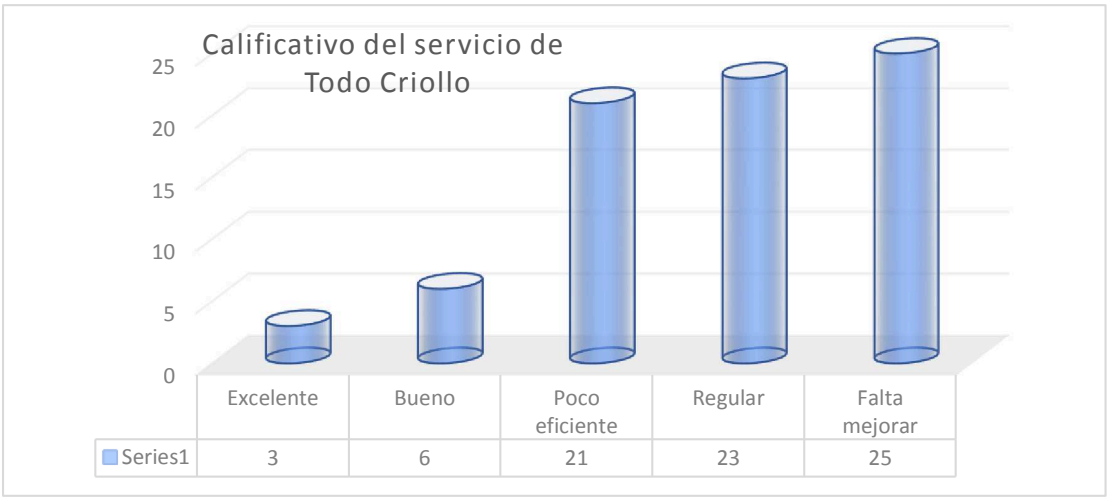


Ilustración 39 - Resultados pregunta 5 de encuesta

Anexo C Ficha de Observación

DATOS DE LA OBSERVACIÓN	
OBJETO: **OBJETIVO:**	
DESCRIPCIÓN DEL FENÓMENO:	**ANÁLISIS DEL FENÓMENO:**
CONCLUSIÓN:	
Responsabilidad: F: ……. Fecha: Manta enero 15 de 2017 Hora: 15h00	

Anexo D Historias de usuario.

HISTORIA DE USUARIO	
Numero: 2	**Nombre de la funcionalidad:** El sistema debe permitir exportar un archivo XML que funcione para el Sri
Usuario: Admistrador	
Prioridad en el negocio: Alta	**Riesgo de desarrollo:** Alta
Descripción: Al final del mes el contador podrá emitir un archivo XML con las compras y las ventas del establecimiento para la declaración mensual de impuestos	
Observaciones: trabajar con la contadora.	

HISTORIA DE USUARIO	
Numero: 3	**Nombre de la funcionalidad:** Se podrá exportar un archivo en Excel con la información para el programa Bartender que permitir imprimir las etiquetas para los productos.
Usuario: Admistrador	
Prioridad en el negocio: Baja	**Riesgo de desarrollo:** Baja
Descripción: El usuario podrá elegir los productos a imprimir con las diferentes formas de envase para el programa bartender	
Observaciones:	

HISTORIA DE USUARIO	
Numero: 4	**Nombre de la funcionalidad:** El ingreso al sistema deberá estar divido por roles para el control de la seguridad de la información
Usuario: Admistrador	**Riesgo de desarrollo:** Baja

Prioridad en el negocio: Baja	
Descripción: Se definirá los diferentes roles para la admisión del sistema a los empleados	
Observaciones:	

HISTORIA DE USUARIO	
Numero: 5	**Nombre de la funcionalidad:** En el producto se podrá introducir información sobre las variantes de los productos, el cual tendrá que poner peso y precio para determinar las etiquetas
Usuario: Admistrador, Usuario	
Prioridad en el negocio: Alta	**Riesgo de desarrollo:** Media
Descripción: Debido a la forma que el negocio se lleva en cuestiones de producción los productos vienen en diferentes variantes, por lo tanto, es necesario un control de estos y división correcta para el inventario.	
Observaciones:	

HISTORIA DE USUARIO	
Numero: 6	**Nombre de la funcionalidad:** El sistema podrá llevar la gestión de clientes y proveedores
Usuario: Admistrador, Usuario	
Prioridad en el negocio: Alta	**Riesgo de desarrollo:** Media
Descripción: La implementación de Odoo ya cuenta con este módulo por lo cual solo se debe configurar y hacer la transferencia de información del sistema anterior	
Observaciones:	

HISTORIA DE USUARIO	
Numero: 7	**Nombre de la funcionalidad:** El sistema deberá tener los impuestos al día de acuerdo a la normativa NIFF
Usuario: Admistrador, Usuario	
Prioridad en el negocio: Alta	**Riesgo de desarrollo:** Baja
Descripción: El módulo de contabilidad, se modificó por lo cual con un módulo aparte se podrá actualizar todos los impuestos de acuerdo a las normativas ecuatorianas.	
Observaciones:	

HISTORIA DE USUARIO	
Numero: 8	**Nombre de la funcionalidad:** El sistema debe estar acorde a las funciones necesaria para el punto de venta que el cliente desea
Usuario: Admistrador, Usuario	
Prioridad en el negocio: Alta	**Riesgo de desarrollo:** Media
Descripción: El módulo de punto de venta se le harán las modificaciones necesarias según el cliente para poder leer los códigos que se crearan por medio de los otros módulos.	
Observaciones:	

HISTORIA DE USUARIO	
Numero: 9	**Nombre de la funcionalidad:** La impresión deberá estar acorde a las facturas del sistema
Usuario: Admistrador, Usuario	
Prioridad en el negocio: Alta	**Riesgo de desarrollo:** Media
Descripción: La factura personalizada de la empresa deberá estar igual que el sistema anterior.	
Observaciones:	

Anexo E.

Diagrama número 2.

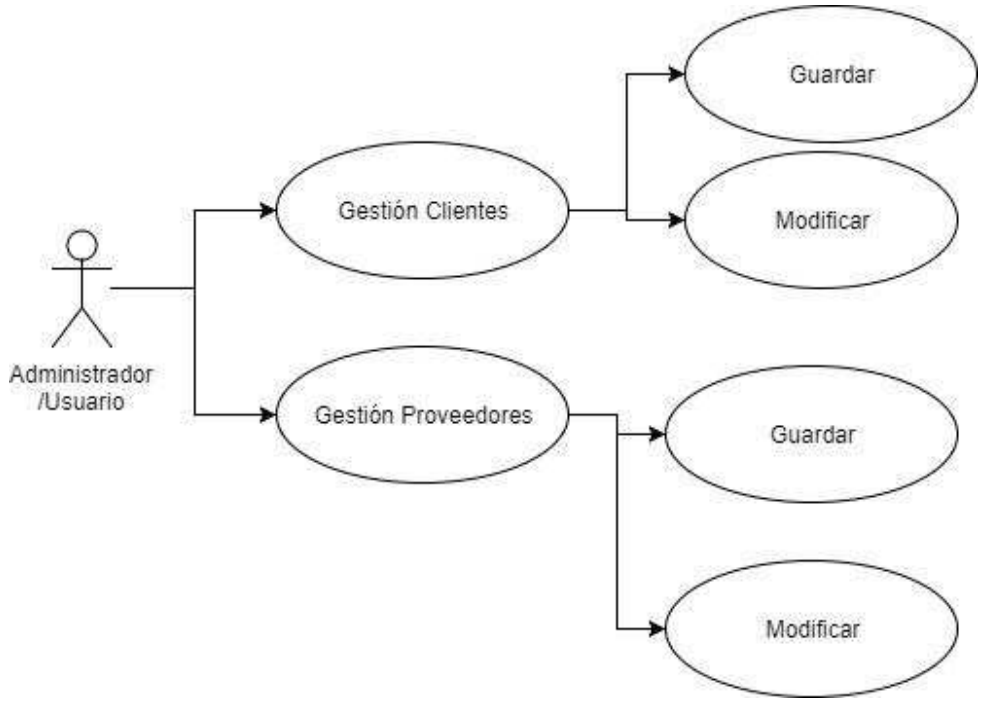


Diagrama número 3.

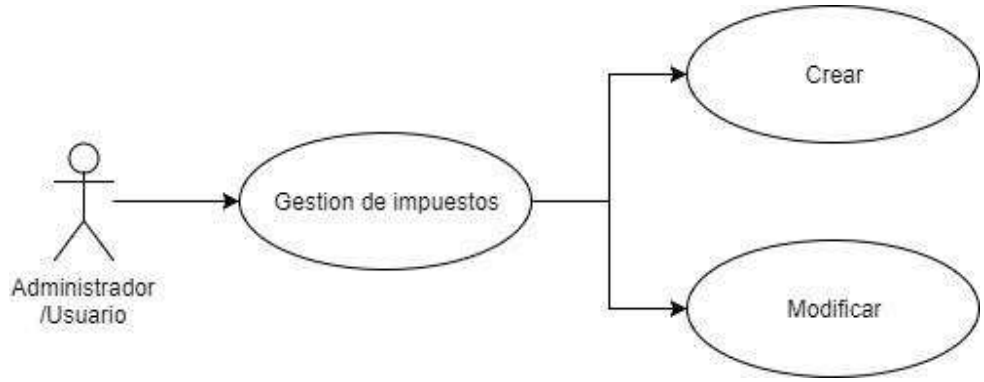


Diagrama número 4.

Anexo F.

Diagrama de Secuencia 2.

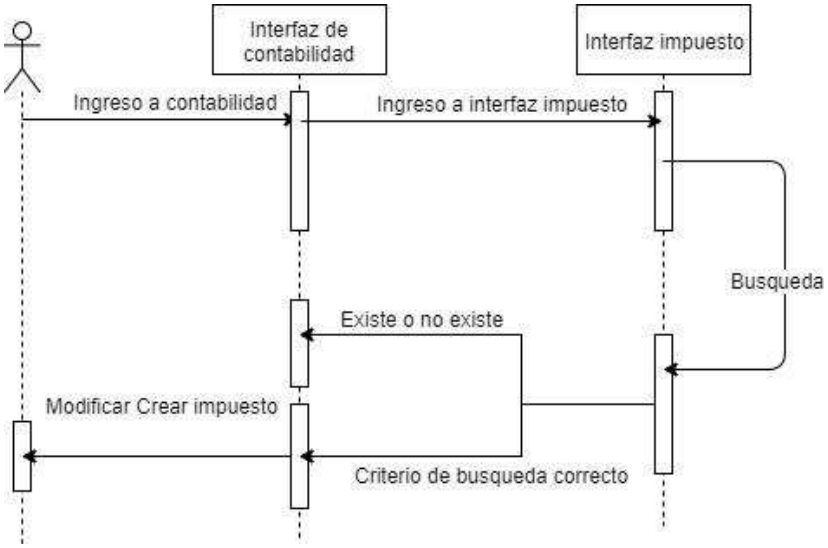
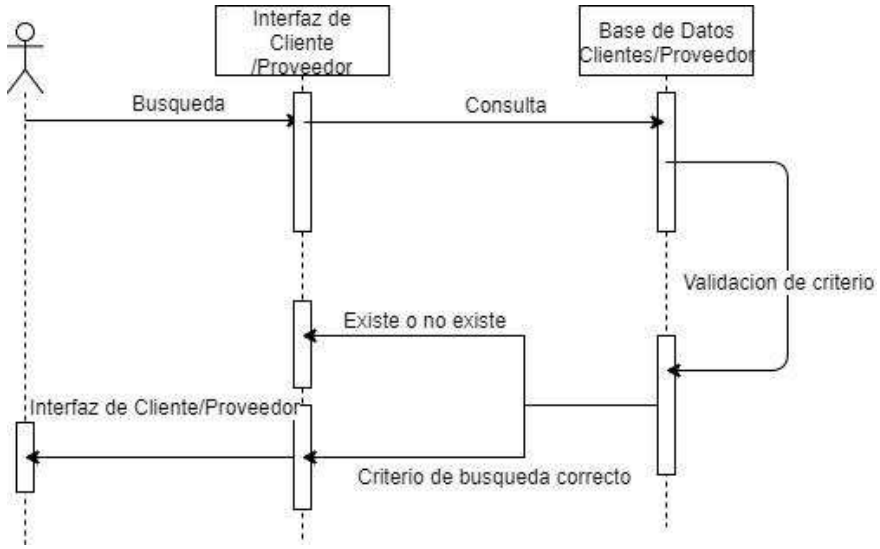


Diagrama de Secuencia 3.



Anexo G.

REGISTRO DE CAPACITACIÓN

FECHA	ACTIVIDADES	RESPONSABLES	PERSONAS CAPACITADAS	FIRMA DE PARTICIPANTES
26 JUNIO 2017	PRESENTACIÓN Y MANEJO DEL AREA DE VENTAS.	BRYAN REDEÑO	JULIA VELIZ RAFAEL PALACIOS SULIA PALACIOS	
26 JUNIO 2017	CAPACITACION DE FUNCIONALIDADES DEL PUNTO DE VENTA.	BRYAN LEDEÑO	RAFAEL PALACIOS JULIA PALACIOS JULIA VELIZ	
26 JUNIO 2017	DESCRIPCIÓN Y MANEJO DEL PROCESO DE VENTAS A CRÉDITO.	BRYAN CEDEÑO	RAFAEL PALACIOS JULIA VELIZ JULIA PALACIOS	
26 JUNIO 2017	FUNCIONES DE PRESUPUESTO Y PEDIDOS DE VENTAS	BRYAN CEDEÑO	RAFAEL PALACIOS JULIA VELIZ JULIA PALACIOS	
26 JUNIO 2017				

Rafael Palacios Mero
TODO CRIOLLO
0994780661

REGISTRO DE CAPACITACIÓN

FECHA	ACTIVIDADES	RESPONSABLES	PERSONAS CAPACITADAS	FIRMA DE PARTICIPANTES
27 Junio 2017	Presentación de funcionalidades y manejo del área de compra.	Jeffri Villacreses	Mariselo Saltos	
27 Junio 2017	Órdenes de Compra	Jeffri Villacreses	Mariselo Saltos	
27 Junio 2017	Pago a proveedores.	Jeffri Villacreses	Mariselo Saltos.	

Rafael Palacios Mero
TODO CRIOLLO
0994780661

REGISTRO DE CAPACITACIÓN

FECHA	ACTIVIDADES	RESPONSABLES	PERSONAS CAPACITADAS	FIRMA DE PARTICIPANTES
28 JUNIO 2017	PRESENTACIÓN Y MANEJO DEL AREA DE CONTABILIDAD	BRYAN CEDEÑO	MARISELA SALTOS GENNY LUCAS	
28 JUNIO 2017	ASIENTOS DIARIOS	BRYAN CEDEÑO	GENNY LUCAS MARISELA SALTOS	
28 JUNIO 2017	GENERAR ANEXO TRANSACCIONAL	BRYAN CEDEÑO	GENNY LUCAS MARISELA SALTOS	
28 JUNIO 2017	AJUSTES DE IMPUESTOS	BRYAN CEDEÑO	GENNY LUCAS MARISELA SALTOS	
28 JUNIO 2017	FACTURAS A CLIENTE Y PROVEEDORES	BRYAN CEDEÑO	GENNY LUCAS MARISELA SALTOS	

Rafael Palacios Mero
TODO CRIOLLO
0994780661

REGISTRO DE CAPACITACIÓN

FECHA	ACTIVIDADES	RESPONSABLES	PERSONAS CAPACITADAS	FIRMA DE PARTICIPANTES
28 Junio 2017	Presentación. de funcionalidades del listado de producto.	Bryan Cedeño	Mayra Véliz	Mayra Véliz Garcia.
28 Junio 2017	Generar listado de productos con respectivo código de barra.	Bryan Cedeño	Mayra Véliz	Mayra Véliz Garcia.
28 Junio 2017	Impresión de etiquetas.	Bryan Cedeño	Mayra Véliz	Mayra Véliz Garcia.
28 Junio 2017	Exportación de archivo xls.	Bryan Cedeño	Mayra Véliz	Mayra Véliz Garcia.

Rafael Palacios Mero
TODO CRIOLLO
0994780661

REGISTRO DE CAPACITACIÓN

FECHA	ACTIVIDADES	RESPONSABLES	PERSONAS CAPACITADAS	FIRMA DE PARTICIPANTES
26 Junio 2017	Presentación correspondiente al área de Bodega.	Jeffri Villacreses	Darwin Cedeño Ramón Chávez	
26 Junio 2017	Realizar ajuste de inventario.	Jeffri Villacreses	Darwin Cedeño Ramón Chávez	
26 Junio 2017	Alimentación de inventario.	Jeffri Villacreses	Darwin Cedeño Ramón Chávez	
26 Junio 2017	Traspaso de Bodega.	Jeffri Villacreses	Darwin Cedeño Ramón Chávez	
26 Junio 2017	Creación de Productos.	Jeffri Villacreses	Darwin Cedeño Ramón Chávez.	

Rafael Palacios Mero
TODO CRIOLLO
0994780661

Todo Criollo

MANUAL DEL PROGRAMADOR SISTEMA ERP INFORMÁTICO PARA LA GESTIÓN DE INVENTARIO, COMPRA, VENTA Y CONTABILIDAD

Índice

Comenzando con Odoo 5

Configurar un equipo como servidor Odoo 5

Disposiciones para un equipo Debian 5

Creando una cuenta de usuario para Odoo 6

Instalar Odoo desde el código fuente 6

Inicializar una base de datos nueva en Odoo 7

Gestionar la base de datos 7

Más opciones de configuración del servidor 6

Archivos de configuración del servidor Odoo 6

Cambiar el puerto de escucha..... 6

Registro..... 7

Desarrollar desde la estación de trabajo 7

Usar un editor de texto Linux..... 7

Instalar y configurar Samba 7

Habilitar las herramientas técnicas..... 8

Activar las Características Técnicas 8

Activar el modo de Desarrollo 8

Instalar módulos de terceras partes..... 0

Encontrar módulos de la comunidad 0

Configurar la ruta de complementos 0

Actualizar la lista de módulos 10

Construyendo tu primera aplicación con Odoo 10

Entender las aplicaciones y los módulos . 10

Modificar un módulo existente..... 12

Crear un módulo nuevo 12

Agregar el módulo a la ruta de complementos 13

Instalar el módulo nuevo 14

Actualizar un módulo..... 14

Crear un modelo de aplicación..... 15

Agregar entradas al menú 0

Crear vistas - formulario, árbol y búsqueda 17

Creando una vista formulario.....17

Formatear como un documento de negocio 17

Agregar botones de acción.....18

Organizar formularios usando grupos 18

La vista de formulario completa 18

Agregar vistas de lista y búsqueda 19

Agregar la lógica de negocio.....19

Configurando la seguridad en el control de acceso 20

Reglas de acceso de nivel de fila.....21

Agregar un ícono al módulo.....22

Agregar la capacidad de compartir con otros a la aplicación To-Do.....22

Camino a seguir para las características colaborativas 22

Ampliando el modelo de tareas por hacer 23

Agregar campos a un modelo.....23

Modificar los campos existentes 23

Modificar los métodos del modelo.....24

Ampliar las vistas.....24

Ampliando mas vistas de árbol y búsqueda 26

Más sobre el uso de la herencia para ampliar los modelos.....26

Copiar características usando herencia por prototipo 27

Integrar Modelos usando herencia delegada 27

Usar la herencia para agregar características redes sociales 28

Modificar datos.....28

Ampliando las reglas de registro.....29

Modelos – Estructura de los Datos de la Aplicación 30
Organizar las características de las aplicaciones en módulos..... 30
Introducción al módulo todo_ui 30
Crear modelos 31
Atributos del modelo..... 31
Modelos y clases Python 31
Modelos transitorios y abstractos 32
Inspeccionar modelos existentes 0
Crear campos 0
Tipos básicos de campos 0
Atributos de campo comunes 35
Nombres de campo reservados..... 36
Relaciones entre modelos..... 36
Relaciones muchos a uno 37
Relaciones muchos a muchos..... 37
Relaciones inversas de uno a muchos..... 38
Relaciones jerárquicas..... 38
Hacer referencia a campos usando relaciones dinámicas 39
Campos calculados..... 39
Buscar y escribir en campos calculados .. 40
Guardar campos calculados..... 40
Campos relacionados 40
Restricciones del Modelo 41
Vistas – Diseñar la Interfaz 41
Acciones de ventana..... 42
Opciones de menú 42
Contexto y dominio..... 43
Contexto de sesión 43
Expresiones de dominio 43
Vistas de Formulario 45
Vistas de negocio 45
La barra de estado del encabezado 45
El flujo de negocio 46
Título y subtítulo.....46
Etiquetas y campos 47
Botones inteligentes 47
Organizar el contenido en formulario 48
Cuaderno con pestañas.....48
Elementos de la vista 48
Botones 48
Campos 49
Campos relacionales 49
Widgets de campo.....50
Eventos on-change 50
Vistas dinámicas 50
Vistas de lista.....51
Vistas de búsqueda.....51
Otros tipos de vista 52
Vistas de Calendario 52
Vistas de Gantt.....53
Vistas de Gráfico 53
QweB - Creando vistas Kanban y Reportes 53
Iniciándose con el tablero Kanban 54
Vistas Kanban.....54
Diseña vistas Kanban.....0
Prioridad y estado Kanban 59
Acciones en las vistas Kanban 60
Agrengano contenido dinámico Qweb 61
Renderizado Condicional con t-if 61
DISEÑO DE LA BASE DE DATOS DE LOS MODULOS MODIFICADOS.....63

DESCRIPCIÓN DE TABLAS DE MODULOS MODIFICADOS:..... 64

CODIGO FUENTE DE MODULO DESARROLLADOS..... 65

Comenzando con Odoo

Se aprenderá a configurar sistemas Debian o Ubuntu para alojar las instancias del servidor de desarrollo, y como instalar Odoo desde el código fuente en GitHub. Luego aprenderá a configurar archivos compartidos con Samba, permitiendo trabajar con archivos de Odoo desde una estación de trabajo con cualquier sistema operativo.

Odoo está desarrollado usando el lenguaje de programación Python y usa PostgreSQL como base de datos para almacenar datos, estos son los requisitos principales para trabajar con Odoo. Para ejecutar Odoo desde el código fuente, es necesario instalar las librerías Python de las cuales depende. Luego el código fuente de Odoo debe descargarse desde GitHub y ejecutado desde el código fuente. Aunque es posible descargar un Zip o tarball, es mejor obtener el código fuente usando GitHub, así además tendremos Odoo instalado en nuestro equipo.

Configurar un equipo como servidor Odoo

Preferimos usar sistemas Debian/Ubuntu para el servidor Odoo, aunque puede trabajar desde el sistema operativo de su preferencia, sea Windows, Macintosh, o Linux.

Odoo puede ser ejecutado en una gran variedad de sistemas operativos, entonces ¿por qué elegir Debian por encima de otros sistemas operativos? Debido a que Odoo es desarrollado principalmente para sistemas Debian/Ubuntu, el soporte para Odoo es mejor. Por lo tanto será más fácil encontrar ayuda y recursos adicionales si se trabaja con Debian/Ubuntu.

También son las plataformas más usadas por las personas que desarrollan aplicaciones, y donde se dan a conocer más implementaciones. Por esta razón, inevitablemente, se espera que las desarrolladoras y los desarrolladores de Odoo se sientan a gusto con esta plataforma. Incluso quienes tiene una historial de trabajo con Windows, es importante que tengan algún conocimiento sobre estas plataformas.

En este capítulo, se aprenderá a configurar y trabajar con Odoo sobre un sistema Debian, usando únicamente la línea de comandos. Para quienes están acostumbrados a sistemas Windows, se describirá como configurar una máquina virtual para alojar un servidor Odoo. Adicionalmente, las técnicas aprendidas servirán para gestionar servidores Odoo en la nube donde el único acceso será a través de una **Shell Segura (SSH).**

Disposiciones para un equipo Debian

Como se explicó antes, será necesario un equipo con Debian para alojar nuestro servidor Odoo versión 8.0. Si estos son sus primeros pasos con Linux, le gustará saber que Ubuntu es una distribución Linux basada en Debian, por lo tanto son muy similares

Si ya está ejecutando Ubuntu u otra distribución basada en Debian, todo está listo para comenzar; ésta máquina también puede ser usada para alojar Odoo.

Para los sistemas operativos Windows y Macintosh, es posible tener Python, PostgreSQL, y todas las dependencias instaladas, y luego ejecutar Odoo desde el código fuente de forma nativa.

Sin embargo, esto puede ser un gran reto, por lo que nuestra recomendación es usar una máquina virtual ejecutando Debian o Ubuntu Server. Puede usar su software de virtualización preferido para hacer funcionar Debian en una máquina virtual. Si necesita alguna ayuda, aquí hay algunos consejos: en lo que se refiere a software de virtualización, tiene muchas opciones, como Microsoft Hyper-V (disponible para algunas versiones de Windows), Oracle VirtualBox, o VMWare Player (o VMWare Fusion para Macintosh). VMWare Player es probablemente el más fácil de usar, y puede descargarse gratuitamente enhttps://my.vmware.com/web/vmware/downloads

Con relación a la imagen Linux a usar, Ubuntu Server es más amigable para las usuarias y usuarios para instalar que Debian. Si está comenzando con Linux, es recomendable que use una distribución lista para usar. TurnKey Linux provee imágenes fáciles de usar, preinstaladas en distintos formatos, incluyendo ISO. El formato ISO funcionara con cualquier software de virtualización de su preferencia, o incluso en cualquier equipo actual. Una buena opción sería una imagen LAPP, que puede hallarse en http://www.turnkeylinux.org/lapp.

Una vez instalado el sistema e iniciado, debería ser capaz de ingresar en la línea de comando.

Si ingresa usado root, su primera tarea será crear un usuario para ser usado en el trabajo cotidiano, ya que es considerada una mala práctica trabajar como root. Particularmente, el servidor Odoo se rehusará a ejecutarse si está usando root.

Creando una cuenta de usuario para Odoo

Primero, asegúrese que sudo este instalado. Su usuario de trabajo lo necesitará. Si ha accedido como root ejecute los siguientes comandos:

```
$ apt-get update & apt-get upgrade # Instalar actualizaciones del sistema
$ apt-get install sudo # Asegurarse que 'sudo' está instalada
```

Los siguientes comandos crearán un usuario Odoo:

```
$ useradd -m -g sudo -s /bin/bash odoo # Cre a un usuario 'Odoo' con poderes sudo
$ passwd odoo # Solicita y configura una cont raseña para el nuevo usuario
```

Puede cambiar odoo por cualquier nombre que desee. La opción -m crea el directorio home. El -g sudo agrega al nuevo usuario a la lista de usuarios sudo, por lo tanto podrá ejecutar comandos como root, y -s /bin/bash configura la línea de comando predeterminada a bash, la cual es más amigable de usar que la fijada por omisión estándar sh.

Ahora puede acceder con el nuevo usuario y configurar Odoo.

Instalar Odoo desde el código fuente

Los paquetes de Odoo listos para instalar pueden ser encontrados en nightly.odoo.com, disponibles para Windows (.exe), Debian (.deb), CentOS (.rpm), y código fuente (.tar.gz).

Como desarrolladoras y desarrolladores, preferimos hacer la instalación directamente desde el repositorio GitHub. Esto nos permitirá tener más control sobre las sucesivas versiones y actualizaciones.

Para mantener el orden de las cosas, se trabaja en el directorio /odoo-dev que se encuentra en su directorio /home. A lo largo del libro, se asume que este es el lugar donde está instalado el servidor Odoo.

Si está usando Ubuntu, probablemente no necesite esto ya que el proceso de instalación le habrá guiado en la creación de un usuario personal.

Primero, asegúrese que ha accedido con el usuario creado anteriormente, o durante el proceso de instalación, y no como root. Asumiendo que su usuario es odoo, puede confirmar esto con el siguiente comando:

```
$ whoami
odoo
$ echo $HOME
/home/odoo
```

Ahora es posible usar este script. Muestra como instalar Odoo desde el código fuente en un sistema Debian:

```
$ sudo apt-get update & sudo apt-get upgrad e # Instala las actualizaciones del sistema
$ sudo apt-get install git # Instala Git
$ mkdir ~/odoo-dev # Crear el directorio de t rabajo
$ cd ~/odoo-dev # Ingresar en el directorio de trabajo
$ git clone https://github.com/odoo/odoo.git -b 8.0 # Obtiene el código fuente de Odoo
$ ./odoo/odoo.py setup_deps # Instala las dep endencias del sistema Odoo
$ ./odoo/odoo.py setup_pg # Instala PostgreS QL y el usuario administrador para un usuar io Unix
```

Al finalizar, Odoo estará listo para ser usado. El símbolo ~ es un atajo para su directorio raíz (por ejemplo, /home/odoo). La opción git -b 8.0 explícitamente solicita descargar la rama 8.0 de Odoo. En el momento de escribir éste libro, esto es redundante, ya que 8.0 es la rama predeterminada, pero ésto puede cambiar, lo que hará más flexible lo aquí descrito.

Para iniciar una instancia del servidor Odoo, simplemente ejecute odoo.py

```
$ ~/odoo-dev/odoo/odoo.py
```

De forma predeterminada, las instancias de Odoo escuchan a través del puerto 8069, si apuntamos en nuestro navegador ahttp://<server-address>:8069 se llegará a la instancia de Odoo. Cuando se accede por primera vez, se mostrará un asistente para crear

una nueva base de datos, como se muestra en la siguiente imagen:

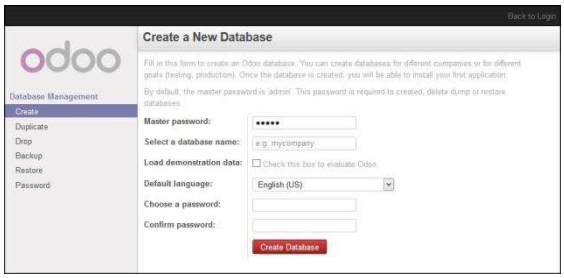


Gráfico 1.1 - Vista Crear una Nueva Base de datos

Pero aprenderá como inicializar bases de datos nuevas desde la línea de comando, ahora presione *Ctrl* + *C* para detener el servidor y volver a la línea de comandos.

Inicializar una base de datos nueva en Odoo

Para poder crear una base de datos nueva, su usuario debe ser un superusuario de PostgreSQL. Lo siguiente hace esto por usted ./odoo.py setup_pg; de lo contrario use el siguiente comando para crear un superusuario PostgreSQL para el usuario Unix actual:

```
$ sudo createuser --superuser $(whoami)
```

Para crear una base de datos nueva use el comando **createdb**. Cree la base de datos **v8dev**:

```
$ createdb v8dev
```

Para inicializar ésta base de datos con el esquema de datos de Odoo debe ejecutar Odoo en la base de datos vacía usando la opción **-d**:

```
$ ~/odoo-dev/odoo/odoo.py -d v8dev
```

Tomará unos minutos inicializar la base de datos v8dev, y terminará con un mensaje de log INFO Modules loaded. Luego el servidor estará listo para atender las peticiones del cliente.

Por defecto, éste método inicializará la base de datos con los datos de demostración, lo cual frecuentemente es útil en bases de datos de desarrollo. Para inicializar una base de datos sin los datos de demostración, agregue la siguiente opción al comando anterior: --without-demo-data=all.

Para mostrar la pantalla de acceso abra en un navegador web http://<server-name>:8069. Si no conoce el nombre de su servidor, escriba el comando hostname en la terminal para averiguarlo, o el comando ifconfig para conocer la dirección IP.

Si está alojando Odoo en una máquina virtual probablemente necesite hacer algunas configuraciones de red para poder usarlo como servidor. La solución más simple es cambiar el tipo de red de la VM de NAT a Bridged. Con esto, en vez de compartir la dirección IP del equipo, la VM huésped tendrá su propia dirección IP. También es posible usar NAT, pero esto requiere que configure el enrutamiento de puerto, así su sistema sabrá que algunos puertos, como el 8069, deben ser controlados por la VM. En caso de algún problema, con suerte estos detalles puedan ayudarle a encontrar ayuda en la documentación del software de virtualización de su preferencia.

La cuenta de usuario predeterminada es admin con la contraseña admin. Una vez acceda se mostrará el menú **Configuración**, revelando los módulos instalados. Elimine el filtro de **Instalado** y podrá ver e instalar cualquiera de los módulos oficiales.

En cualquier momento que desee detener la instancia del servidor Odoo y volver a la línea de comando, presione *Ctrl* + *C*. En consola, presiona la tecla de flecha Arriba para mostrar el comando anterior ejecutado, esta es una forma rápida de iniciar Odoo con las mismas opciones. Notará que *Ctrl* + *C* seguido de la flecha Arriba y *Enter* es una combinación frecuentemente usada para re-iniciar el servidor Odoo durante el desarrollo.

Gestionar la base de datos

Ha visto como crear e inicializar bases de datos nuevas en Odoo desde la línea de comando.

Existen más comandos que valen la pena conocer para gestionar bases de datos.

Ya sabe como usar el comando createdb para crear una base de datos vacía, pero también puede crear una base de datos copiando una existente, usando la opción --template.

Asegúrese que su instancia de Odoo este detenida y no tenga otra conexión abierta con la base de datos v8dev creada anteriormente, y ejecute:

```
$ createdb --template=v8dev v8test
```

De hecho, cada vez que se crea una base de datos, es usada una plantilla. Si no se especifica ninguna, se usa una predefinida llamada template1.

Para listar las bases de datos existentes en su sistema use la utilidad psql de PostgreSQL con la opción -l:

```
$ psql -l
```

Al ejecutar esto se debe listar las dos bases de datos creadas hasta los momentos: v8dev y v8test. La lista también mostrará la codificación usada en cada base de datos. La codificación predeterminada es UTF8, la cual es necesaria para las bases de datos Odoo.

Para eliminar una base de datos que ya no necesite (o necesita crear nuevamente), use el comando dropdb:

```
$ dropdb v8test
```

Ahora ya conoce lo básico para trabajar con varias bases de datos. Para aprender más sobre PostgresSQL, puede encontrar la documentación oficial en http://www.postgresql.org/docs/

Más opciones de configuración del servidor

El servidor Odoo soporta unas pocas opciones más. Es posible verificar todas las opciones disponibles con la opción --help:

```
$ ./odoo.py --help
```

Vale la pena tener una idea general de las más importantes.

Archivos de configuración del servidor Odoo

La mayoría de las opciones pueden ser guardadas en un archivo de configuración. De forma predeterminada, Odoo usará el archivo .openerp-serverrc en su directorio home. Convenientemente, existe una opción --save para guardar la instancia actual de configuración dentro de ese archivo:

```
$ ~/odoo-dev/odoo/odoo.py --save --stop-after
-init # guarda la configuración en archivo
```

Aquí también se usa la opción --stop-after-init, para que el servidor se detenga al finalizar las acciones. Ésta opción es usada frecuentemente para ejecutar pruebas y solicitar la ejecución de actualización de un módulo para verificar que se instala correctamente.

Ahora se puede inspeccionar lo que se guardó en este archivo de configuración:

```
$ more ~/.openerp_serverrc # mostrar el arch
ivo de configuración
```

Esto mostrará todas las opciones de configuración disponibles con sus valores predeterminados. La edición se hará efectiva la próxima vez que inicie una instancia de Odoo. Escriba q para salir y retornar a la línea de comandos.

También es posible seleccionar un archivo específico de configuración, usando la opción --conf=<filepath>. Los archivos de configuración no necesitan tener todas las opciones de configuración que ha visto hasta ahora. Solo es necesario que estén aquellas opciones que cambian los valores predeterminados.

Cambiar el puerto de escucha

El comando --xmlrpc-server=<port> permite cambiar el puerto predeterminado 8069 desde donde la instancia del servidor escucha las peticiones. Esto puede ser usado para ejecutar más de una instancia al mismo tiempo, en el mismo servidor.

Intentemos esto. Abra dos ventanas de la terminal. En la primera ejecute:

```
$ ~/odoo-dev/odoo.py --xmlrpc-port=8070
```

y en la otra ejecute:

```
$ ~/odoo-dev/odoo.py --xmlrpc-port=8071
```

Y allí lo tiene: dos instancias de Odoo en el mismo servidor escuchando a través de diferentes puertos. Las dos instancias pueden ser usadas en la misma o en diferentes base de datos. Y ambas pueden ejecutar versiones iguales o diferentes de Odoo.

Registro

La opción --log-level permite configurar el nivel de detalle del registro. Esto puede ser muy útil para entender lo que esta pasando en el servidor. Por ejemplo, para habilitar el nivel de registro de depuración utilice: --log-level=debug

Los siguientes niveles de registro pueden ser particularmente interesantes: - debug_sql para inspeccionar el SQL generado por el servidor - debug_rpc para detallar las peticiones recibidas por el servidor - debug_rpc para detallar las respuestas enviadas por el servidor

La salida del registro es enviada de forma predeterminada a la salida estándar (la terminal), pero puede ser dirigida a un archivo de registro con la opción --logfile=<filepath>.

Finalmente, la opción --debug llamará al depurador Python (pdb) cuando aparezca una excepción. Es útil hacer un análisis post-mortem de un error del servidor. Note que esto no tiene ningún efecto en el nivel de detalle del registro. Se pueden encontrar más detalles sobre los comandos del depurador de Python aquí:https://docs.python.org/2/library/pdb.html#debugger-commands.

Desarrollar desde la estación de trabajo

Puede ejecutar Odoo con un sistema Debian/Ubuntu, en una máquina virtual local o en un servidor remoto. Pero posiblemente prefiera hacer el trabajo de desarrollo en su estación de trabajo personal, usando su editor de texto o IDE favorito.

Éste puede ser el caso para las personas que desarrollan en estaciones de trabajo con Windows. Pero puede también ser el caso para las personas que usan Linux y necesitan trabajar en un servidor Odoo desde una red local.

Una solución para esto es habilitar el uso compartido de archivos en el servidor Odoo, así los archivos son fáciles de editar desde su estación de trabajo. Para las operaciones del servidor Odoo, como reiniciar el servidor, es posible usar un intérprete de comando SSH (como PUTTY en Windows) junto a su editor favorito.

Usar un editor de texto Linux

Tarde o temprano, será necesario editar archivos desde la línea de comandos. En muchos sistemas Debian el editor de texto predeterminado es vi. Si no se siente a gusto con éste, puede usar una alternativa más amigable. En sistemas Ubuntu el editor de texto predeterminado es nano. Puede que prefiera usar éste ya que es más fácil de usar. En caso que no esté disponible en su servidor, puede instalarlo con:

```
$ sudo apt-get install nano
```

En las siguientes secciones se asumirá como el editor de preferencia. Si prefiere cualquier otro editor, siéntase libre de adaptar los comandos de acuerdo a su elección.

Instalar y configurar Samba

El proyecto Samba proporciona a Linux servicios para compartir archivos compatibles con sistemas Microsoft Windows. Se puede instalar en el servidor Debian/Ubuntu con:

```
$ sudo apt-get install samba samba-common-bin
```

El paquete samba instala el servicio para compartir archivos y el paquete samba-common-bin es necesario para la herramientasmbpasswd. De forma predeterminada las usuarias y usuarios autorizados para acceder a los archivos compartidos necesitan ser registrados. Es necesario registrar el usuario odoo y asignarle una contraseña para su acceso a los archivos compartidos:

```
$ sudo smbpasswd -a odoo
```

Después de esto el usuario odoo podrá acceder a un recurso compartido de archivos para su directorio home, pero será de solo lectura. Se requiere el acceso a escritura, así que es necesario editar los archivos de configuración de Samba para cambiar eso:

```
$ sudo nano /etc/samba/smb.conf
```

En el archivo de configuración, busque la sección [homes]. Edite las líneas de configuración para que sean iguales a los siguientes ajustes:

```
[homes]
   comment = Home Directories
   browseable = yes
   read only = no
   create mask = 0640
   directory mask = 0750
```

Gráfico 1.2 - Características Técnicas de Odoo

Activar las Características Técnicas

Las Características Técnicas proporcionan herramientas avanzadas de configuración del servidor.

Estas están deshabilitadas de forma predeterminada, y para habilitarlas, es necesario acceder con el usuario Administrador. En el menú **Configuración**, seleccione **Usuarios** y edite el usuario Administrador. En la pestaña **Derechos de Acceso**, encontrará una casilla de selección de **Características Técnicas**. Seleccione esa casilla y guarde los cambios.

Ahora es necesario recargar la página en el navegador web. Deberá poder ver en el menú **Configuraciones** una nueva sección **Técnico** que da acceso a lo interno del servidor Odoo.

Para que estos cambios en la configuración tengan efecto, reinicie el servicio:

```
$ sudo /etc/init.d/smbd restart
```

Habilitar las herramientas técnicas

Odoo incluye algunas herramientas que son muy útiles para las personas que desarrollan, y haremos uso de estas a lo largo del libro. Estas son las Características Técnicas y el Modo de Desarrollo.

Estas están deshabilitadas de forma predeterminada, así que aprenderemos como habilitarlas.

La opción del menú **Técnico** permite inspeccionar y editar todas las configuraciones de Odoo almacenadas en la base de datos, desde la interfaz de usuario, a la seguridad y otros parámetros del sistema. Aprenderá mas sobre esto a lo largo del libro.

Activar el modo de Desarrollo

El modo de Desarrollo habilita una caja de selección cerca de la parte superior de la ventana Odoo, haciendo accesible algunas opciones de configuración avanzadas en toda la aplicación. También deshabilita la modificación del código JavaScript y CSS usado por el cliente web, haciendo mas fácil la depuración del comportamiento del lado del cliente.

Para habilitarlo, abra el menú desplegable en la esquina superior derecha de la ventana del navegador, en el nombre de usuario, y seleccione la opción **Acerca de Odoo**. En la ventana de dialogo **Acerca de**, haga clic sobre el botón **Activar modo desarrollador** en la esquina superior derecha.

Gráfico 1.3 - Activar Modo de Desarrollo en Odoo

Luego de esto, verá una caja de selección **Depurar Vista** en la parte superior izquierda del área actual del formulario.

Instalar módulos de terceras partes

Hacer que nuevos módulos estén disponibles en una instancia de Odoo para que puedan ser instalados es algo que puede resultar confuso para las personas nuevas. Pero no necesariamente tiene que ser así, así que a continuación se desmitificará esta suposición.

Encontrar módulos de la comunidad

Existen muchos módulos para Odoo disponibles en Internet. El sitio web https://www.odoo.com/apps es un catalogo de módulos que pueden ser descargados e instalados. La Odoo Community Association (OCA) coordina las contribuciones de la comunidad y mantiene unos pocos repositorios en GitHub, en https://github.com/OCA.

Para agregar un módulo a la instalación de Odoo puede simplemente copiarlo dentro del directorio de complementos, junto a los módulos oficiales. En este caso, el directorio de complementos está en ~/odoo-dev/odoo/addons/. Ésta puede que no sea la mejor opción para Ud., debido a que su instalación está basada en una versión controlada por el repositorio, y querrá tenerla sincronizada con el repositorio de GitHub.

Afortunadamente, es posible usar ubicaciones adicionales para los módulos, por lo que se puede tener los módulos personalizados en un directorio diferente, sin mezclarlos con los complementos oficiales.

Como ejemplo, se descargará el proyecto department de OCA y sus módulos se harán disponibles en la instalación de Odoo. Éste proyecto es un conjunto de módulos muy simples que agregan un campo Departamento en muchos formularios, como en el de Proyectos u Oportunidades de CRM.

Para obtener el código fuente desde GitHub:

```
$ cd ~/odoo-dev
$ git clone https://github.com/OCA/department.git -b 8.0
```

Se usó la opción -b para asegurar que se descargan los módulos de la versión 8.0.

Pero debido a que en el momento de escribir esto la versión 8.0 en la rama predeterminada del proyecto la opción -b podría haber sido omitida.

Luego, se tendrá un directorio /department nuevo junto al directorio /odoo, que contendrá los módulos. Ahora es necesario hacer saber a Odoo sobre este nuevo directorio.

Configurar la ruta de complementos

El servidor Odoo tiene una opción llamada addons-path que define donde buscar los módulo. De forma predeterminada este apunta al directorio /addons del servidor Odoo que se esta ejecutando.

Afortunadamente, es posible asignar no uno, sino una lista de directorios donde se pueden encontrar los módulos. Esto permite mantener los módulos personalizados en un directorio diferente, sin mezclarlos con los complementos oficiales. Se ejecutará el servidor con una ruta de complemento incluyendo el nuevo directorio de módulos:

```
$ cd ~/odoo-dev/odoo
```

```
$ ./odoo.py -d v8dev --addons-path="/department,./addons"
```

Si se observa con cuidado el registro del servidor notará una línea reportando la ruta de los complementos en uso: **INFO ? Openerp: addons paths:** (...). Confirmando que la instancia contiene nuestro directorio department.

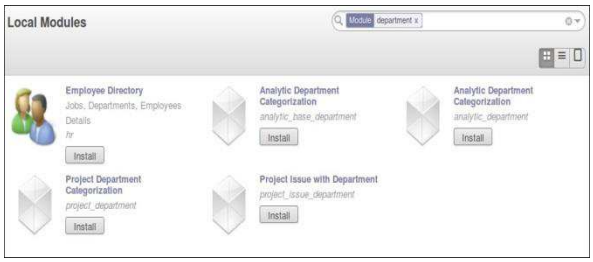
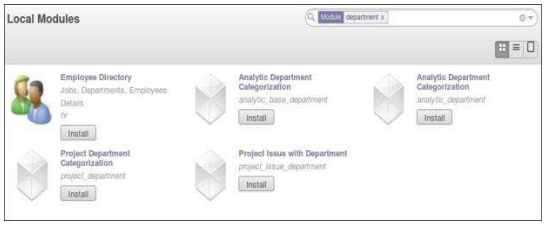


Gráfico 1.4 - Confirmar que la instancia Odoo reconoce el directorio 'department'

Actualizar la lista de módulos

Es necesario pedirle a Odoo que actualice su lista de módulos antes que estos módulos nuevos estén disponibles para ser instalados.

Para esto es necesario habilitar el menú **Técnico**, debido a que esta provee la opción de menú **Actualizar Lista de Módulos**. Esta puede ser encontrada en la sección **Módulos** en el menú **Configuración**.



Luego de ejecutar la actualización de la lista de módulos se puede confirmar que los módulos nuevos están disponibles para ser instalados. En la lista de **Módulos Locales**, quite el filtro de Aplicaciones en línea y busque department. Debería poder ver los nuevos módulos disponibles.

Construyendo tu primera aplicación con Odoo

Desarrollar en Odoo la mayoría de las veces significa crear nuestros propios módulos. En este capítulo, se creará la primera aplicación con Odoo, y se aprenderán los pasos necesarios para habilitarlas e instalarlas en Odoo.

Inspirados del notable proyecto todomvc.com, se desarrollará una simple aplicación para el registro de cosas por hacer. Deberá permitir agregar nuevas tareas, marcarlas como culminadas, y finalmente borrar de la lista todas las tareas finalizadas.

Aprenderá como Odoo sigue una arquitectura MVC, y recorrerá las siguientes capas durante la implementación de la aplicación: - El **modelo**, define la estructura de los datos. - La **vista**, describe la interfaz con el usuario o la usuaria. - El **controlador**, soporta la lógica de negocio de la aplicación.

La capa modelo es definida por objetos Python cuyos datos son almacenados en una base de datos PostgreSQL. El mapeo de la base de datos es gestionado automáticamente por Odoo, y el mecanismo responsable por esto es el **modelo objeto relacional, (ORM - object relational model)**.

La capa vista describe la interfaz con el usuario o la usuaria. Las vistas son definidas usando XML, las cuales son usadas por el marco de trabajo (framework) del cliente web para generar vistas HTML de datos.

Las vistas del cliente web ejecutan acciones de datos persistentes a través de la interacción con el servidor ORM. Estas pueden ser operaciones básicas como escribir o eliminar, pero pueden también invocar métodos definidos en los objetos Python del ORM, ejecutando lógica de negocio más compleja. A esto es a lo que nos referimos cuando se habla de la capa modelo.

Entender las aplicaciones y los módulos

Es común escuchar hablar sobre los módulos y las aplicaciones en Odoo. Pero, ¿Cual es exactamente la diferencia entre un módulo y una aplicación? Los **módulos** son bloques para la construcción de las aplicaciones en Odoo. Un módulo puede agregar o modificar características en Odoo. Esto es soportado por un directorio que contiene un archivo de manifiesto o descriptor (llamado __openerp__.py) y el resto de los archivos que implementan sus características. A veces, los módulos pueden ser llamados "add-ons" (complementos). Las **aplicaciones** no son diferentes de los módulos regulares, pero funcionalmente, éstas proporcionan una característica central, alrededor de la cual otros módulos agregan características u opciones. Estas proveen los elementos base para un área funcional, como contabilidad o RRHH, sobre las cuales otros módulos agregan características. Por esto son resaltadas en el menú Apps de Odoo.

Modificar un módulo existente

En el ejemplo que sigue a continuación, crearemos un módulo nuevo con tan pocas dependencias como sea posible.

Sin embargo, este no es el caso típico. Lo más frecuente serán situaciones donde las modificaciones y extensiones son necesarias en un módulo existente para ajustarlo a casos de uso específicos.

La regla de oro dice que no debemos cambiar módulos existentes modificándolos directamente. Esto es considerado una mala práctica. Especialmente cierto para los módulos oficiales proporcionados por Odoo. Hacer esto no permitirá una clara separación entre el módulo original y nuestras modificaciones, y hace difícil la actualización.

Por el contrario, debemos crear módulos nuevos que sean aplicados encima de los módulos que queremos modificar, e implementar esos cambios. Esta es una de las principales fortalezas de Odoo: provee mecanismos de "herencia" que permiten a los módulos personalizados extender los módulos existentes, bien sean oficiales o de la comunidad. La herencia el posible en todos los niveles, modelo de datos, lógica de negocio, e interfaz con el usuario o usuaria.

Ahora, crearemos un módulo nuevo completo, sin extender ningún módulo existente, para enfocarnos en las diferentes partes y pasos involucrados en la creación de un módulo. Solo daremos una breve mirada a cada parte, ya que cada una será estudiada en detalle en los siguientes capítulos. Una vez estemos a gusto con la creación de un módulo nuevo, podremos sumergirnos dentro de los mecanismos de herencia, los cuales serán estudiados en el siguiente capítulo.

Crear un módulo nuevo

Nuestro módulo será una aplicación muy simple para gestionar las tareas por hacer. Estas tareas tendrán un único campo de texto, para la descripción, y una casilla de verificación para marcarlas como culminadas. También tendremos un botón para limpiar la lista de tareas de todas aquellas finalizadas.

Estas especificaciones son muy simples, pero a medida que avancemos en el libro iremos agregando gradualmente nuevas características, para hacer la aplicación más interesante.

Basta de charla, comencemos a escribir código y crear nuestro nuevo módulo.

Siguiendo las instrucciones del *Capítulo 1, Comenzando con Odoo*, debemos tener el servidor Odoo en /odoo-dev/odoo/. Para mantener las cosas ordenadas, crearemos un directorio junto a este para guardar nuestros propios módulos:

```
$ mkdir ~/odoo-dev/custom-addons
```

Un módulo en Odoo es un directorio que contiene un archivo descriptor __openerp__.py. Esto es una herencia de cuando Odoo se llamaba OpenERP, y en el futuro se espera se convierta en __odoo__.py. Es necesario que pueda ser importado desde Python, por lo que debe tener un archivo __init__.py.

El nombre del directorio del módulo será su nombre técnico. Usaremos todo_app para el nombre. El nombre técnico debe ser un identificador Python valido: debe comenzar con una letra y puede contener letras, números y el carácter especial guión bajo. Los siguientes comandos crean el directorio del módulo y el archivo vacío __init__.py dentro de este:

```
$ mkdir ~/odoo-dev/custom-addons/todo_app
```

```
$ touch ~/odoo-dev/custom-addons/todo_app/__init__.py
```

Luego necesitamos crear el archivo descriptor. Debe contener únicamente un diccionario Python y puede contener alrededor de una docena de atributos, de los cuales solo el atributo **name** es obligatorio. Son recomendados los atributos **description**, para una descripción más larga, y **author**. Ahora agregamos un archivo **__openerp__.py** junto al archivo **__init__.py** con el siguiente contenido:

```
{
    'name': 'To-Do Application',
    'description': 'Manage your personal Tasks with this module.',
    'author': 'Daniel Reis',
    'depends': ['mail'],
    'application': True,
}
```

El atributo depends puede tener una lista de otros módulos requeridos. Odoo los instalará automáticamente cuando este módulo sea instalado. No es un atributo obligatorio pero se recomienda tenerlo siempre. Si no es requerida alguna dependencia en particular, debería existir alguna dependencia a un módulo base especial. Debe tener cuidado de asegurarse que todas las dependencias sean explícitamente fijadas aquí, de otra forma el módulo podría fallar al instalar una base de datos vacía (debido a dependencias insatisfechas) o tener errores en la carga, si otros módulos necesarios son cargados después.

Para nuestra aplicación, queremos que dependa del módulo **mail** debido a que este agrega el menú **Mensajería** en la parte superior de la ventana, y queremos incluir nuestro nuevo menú de opciones allí.

Para precisar, escogimos pocas claves del descriptor, pero en el mundo real es recomendable usar claves adicionales, ya que estas son relevantes para la app-store de Odoo:

- summary, muestra un subtitulo del módulo.
- version, de forma predeterminada, es 1.0. Se debe seguir las reglas de versionamiento semántico (para más detalles versemver.org).
- license, de forma predeterminada es AGPL-3.
- website, es una URL para encontrar más información sobre el módulo. Esta puede servir a las personas a encontrar documentación, informar sobre errores o hacer sugerencias.
- category, es la categoría funcional del módulo, la cual de forma predeterminada es Sin Categoría. La lista de las categorías existentes puede encontrarse en el formato de Grupos (Configuraciones | Usuarios | menú Grupos), en la lista desplegable del campo Aplicación.

Estos descriptores también están disponibles: - installable, de forma predeterminada es True, pero puede ser fijada Falsepara deshabilitar el módulo. - auto_install, si esta fijada en True este módulo es automáticamente instalado si todas las dependencias han sido instaladas. Esto es usado en módulos asociados.

Desde Odoo 8.0, en vez de la clave description podemos usar un archivo README.rst o README.md en el directorio raíz del módulo.

Agregar el módulo a la ruta de complementos

Ahora que tenemos un módulo nuevo, incluso si es muy simple, queremos que esté disponible en Odoo. Para esto, debemos asegurarnos que el directorio que contiene el módulo sea parte de la ruta de complementos addons. Y luego tenemos que actualizar la lista de módulos de Odoo.

Ambas operaciones han sido explicadas en detalle en el capítulo anterior, pero a continuación presentamos un resumen de lo necesario.

Nos posicionamos dentro del directorio de trabajo e iniciamos el servidor con la configuración de la ruta de complementos o addons:

```
$ cd ~/odoo-dev
$ odoo/odoo.py -d v8dev --addons-path="custom-addons,odoo/addons" --save
```

La opción --save guarda la configuración usada en un archivo de configuración. Esto evita repetirlo cada vez que el servidor es iniciado: simplemente ejecute ./odoo.py y serán ejecutadas las últimas opciones guardadas.

Mira detenidamente en el registro del servidor. Debería haber una línea **INFO ? openerp: addons paths:** (...), y debería incluir nuestro directorio custom-addons.

Recuerde incluir cualquier otro directorio que pueda estar usando. Por ejemplo, si siguió las instrucciones del último capítulo para instalar el repositorio department, puede querer incluirlo y usar la opción:

```
--addons-path="custom-addons,department,
odoo/addons"
```

Ahora hagamos que Odoo sepa de los módulos nuevos que hemos incluido.

Para esto, En la sección Módulos del menú Configuración, seleccione la opción Actualizar lista de módulos. Esto actualizará la lista de módulos agregando cualquier módulo incluido desde la última actualización de la lista. Recuerde que necesitamos habilitar las Características Técnicas para que esta opción sea visible. Esto se logra seleccionando la caja de verificación de Características técnicas para nuestra cuenta de usuario.

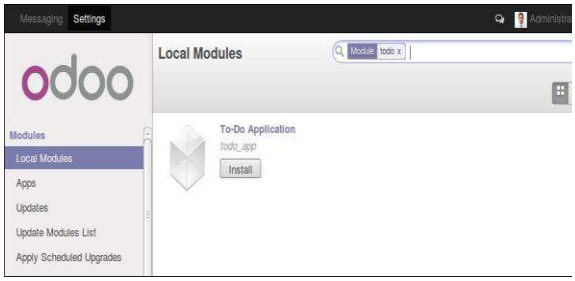


Gráfico 2.1 - Instalar nuevo módulo 'todo_app'

Haga clic en el botón **Instalar** y listo!

Actualizar un módulo

El desarrollo de un módulo es un proceso iterativo, y puede querer que los cambios hechos en los archivos fuente sean aplicados y estén visibles en Odoo.

En la mayoría de los casos esto es hecho a través de la actualización del módulo: busque el módulo en la lista de Módulos Locales y, ya que está instalado, debe poder ver el botón Actualizar.

De cualquier forma, cuando los cambios realizados son en el código Python, la actualización puede no tener ningún efecto. En este caso es necesario reiniciar la aplicación en el servidor.

En algunos casos, si el módulo ha sido modificado tanto en los archivos de datos como en el código Python, pueden ser necesarias ambas operaciones. Este es un punto común de confusión para las personas que se inician en el desarrollo con Odoo.

Instalar el módulo nuevo

La opción **Módulos locales** nos muestra la lista de módulos disponibles. De forma predeterminada solo muestra los módulos de**Aplicaciones en línea**. Debido a que creamos un módulo de aplicación no es necesario remover este filtro. Escriba "todo" en la campo de búsqueda y debe ver nuestro módulo nuevo, listo para ser instalado.

Pero afortunadamente, existe una mejor forma. La forma más simple y rápida para hacer efectivos todos los cambios en nuestro módulo es detener (*Ctrl* + *C*) y reiniciar el proceso del servidor que requiere que nuestros módulos sean actualizados en la base de datos de trabajo.

Para hacer que el servidor inicie la actualización del módulo todo_app en la base de datos v8dev, usaremos:

```
$ ./odoo.py -d v8dev -u todo_app
```

La opción -u (o --update en su forma larga) requiere la opción -d y acepta una lista separada por comas de módulos para actualizar. Por ejemplo, podemos usar: -u todo_app,mail.

En el momento en que necesite actualizar un módulo en proceso de desarrollo a lo largo del libro, la manera mas segura de hacerlo es ir a una ventana de terminal donde se este ejecutando Odoo, detener el servidor, y reiniciarlo con el comando visto anteriormente. Usualmente será suficiente con presionar la

tecla de flecha arriba, esto debería devolver el último comando usado para iniciar el servidor.

Desafortunadamente, la actualización de la lista de módulos y la desinstalación son acciones que no están disponibles a través de la línea de comandos. Esto debe ser realizado a través de la interfaz web, en el menú Configuraciones.

Crear un modelo de aplicación

Ahora que Odoo sabe sobre la disponibilidad de nuestro módulo nuevo, comencemos a agregarle un modelo simple.

Los modelos describen los objetos de negocio, como una oportunidad, una orden de venta, o un socio (cliente, proveedor, etc). Un modelo tiene una lista de atributos y también puede definir su negocio específico.

Los modelos son implementados usando clases Python derivadas de una plantilla de clase de Odoo. Estos son traducidos directamente a objetos de base de datos, y Odoo se encarga de esto automáticamente cuando el módulo es instalado o actualizado.

Algunas personas consideran como buena práctica mantener los archivos Python para los modelos dentro de un subdirectorio. Por simplicidad no seguiremos esta sugerencia, así que vamos a crear un archivo todo_model.py en el directorio raíz del módulo todo_app.

Agregar el siguiente contenido:

```
#-*- coding: utf-8 -*-
from openerp import models, fields

class TodoTask(models.Model):
    _name = 'todo.task'
    name = fields.Char('Description', required
=True)
    is_done = fields.Boolean('Done?')
    active = fields.Boolean('Active?', default=T
rue)
```

La primera línea es un marcador especial que le dice al interprete de Python que ese archivo es UTF-8, por lo que puede manejar y esperarse caracteres non-ASCII. No usaremos ninguno, pero es mas seguro usarlo.

La segunda línea hace que estén disponibles los modelos y los objetos campos del núcleo de Odoo.

la tercera línea declara nuestro nuevo modelo. Es una clase derivada de models.Model. La siguiente línea fija el atributo _namedefiniendo el identificador que será usado por Odoo para referirse a este modelo. Note que el nombre real de la clase Python no es significativo para los otros módulos de Odoo. El valor de _name es lo que será usado como identificador.

Observe que éstas y las siguientes líneas tienen una sangría. Si no conoce muy bien Python debe saber que esto es sumamente importante: la sangría define un bloque de código anidado, por lo tanto estas cuatro líneas deben tener la misma sangría.

Las últimas tres líneas definen los campos del modelo. Vale la pena señalar que name y active son nombres de campos especiales. De forma predeterminada Odoo usara el campo name como el título del registro cuando sea referenciado desde otros modelos. El campo active es usado para desactivar registros, y de forma predeterminada solo los registros activos son mostrados. Lo usaremos para quitar las tareas finalizadas sin eliminarlas definitivamente de la base de datos.

Todavía, este archivo, no es usado por el módulo. Debemos decirle a Odoo que lo cargue con el módulo en el archivo __init__.py. Editemos el archivo para agregar la siguiente línea:

```
from . import todo_model
```

Esto es todo. para que nuestros cambios tengan efecto el módulo debe ser actualizado. Encuentre la aplicación To-Do en Módulos Locales y haga clic en el botón Actualizar.

Ahora podemos revisar el modelo recién creado en el menú Técnico. Vaya a Estructura de la Base de Datos | Modelos y busque el modelo todo.task en la lista. Luego haga clic en este para ver su definición:

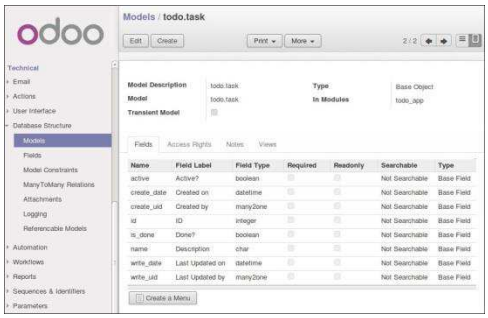


Gráfico 2.2 - Vista de Estructura de la Base de Datos de módulo 'todo_app'

Si no hubo ningún problema, esto nos confirmará que el modelo y nuestros campos fueron creados. Si hizo algunos cambios y no son reflejados, intente reiniciar el servidor, como fue descrito anteriormente, para obligar que todo el código Python sea cargado nuevamente.

También podemos ver algunos campos adicionales que no declaramos. Estos son cinco campos reservados que Odoo agrega automáticamente a cualquier modelo. Son los siguientes: - id: Este es el identificador único para cada registro en un modelo en particular. - create_date y create_uid: Estos nos indican cuando el registro fue creado y quien lo creó, respectivamente. - write_date y write_uid: Estos nos indican cuando fue la última vez que el registro fue modificado y quien lo modificó, respectivamente.

Agregar entradas al menú

Ahora que tenemos un modelo en el cual almacenar nuestros datos, hagamos que este disponible en la interfaz con el usuario y la usuaria.

Todo lo que necesitamos hacer es agregar una opción de menú para abrir el modelo de "To-do Task" para que pueda ser usado. Esto es realizado usando un archivo XML. Igual que en el caso de los modelos, algunas personas consideran como una buena practica mantener las definiciones de vistas en en un subdirectorio separado.

Crearemos un archivo nuevo todo_view.xml en el directorio raíz del módulo, y este tendrá la declaración de un ítem de menú y la acción ejecutada por este:

```
<?xml version="1.0" encoding="UTF-8"?>
    <openerp>
        <data>
            <!-- Action to open To-do Task list -->
            <act_window
                id="action_todo_task"
                name="To-do Task"
                res_model="todo.task"
                view_mode="tree,form"
            />
            <!-- Menu item to open To-do Task lis
t -->
            <menuitem
                id="menu_todo_task"
                name="To-Do Tasks"
                parent="mail.mail_feeds"
                sequence="20"
                action="action_todo_task"
            />
        </data>
    </openerp>
```

La interfaz con el usuario y usuaria, incluidas las opciones del menú y las acciones, son almacenadas en tablas de la base de datos. El archivo XML es un archivo de datos usado para cargar esas definiciones dentro de la base de datos cuando el módulo es instalado o actualizado. Esto es un archivo de datos de Odoo, que describe dos registros para ser agregados a Odoo: - El elemento <act_window> define una Acción de Ventana del lado del cliente para abrir el modelo todo.task definido en el archivo Python, con las vistas de árbol y formulario habilitadas, en ese orden. - El elemento <menuitem> define un ítem de menú bajo el menú Mensajería (identificado por mail.mail_feeds), llamando a la acción action_todo_task, que fue definida anteriormente. el atributo sequence nos deja fijar el orden de las opciones del menú.

Ahora necesitamos decirle al módulo que use el nuevo archivo de datos XML. Esto es hecho en el archivo __openerp__.pyusando el atributo data. Este define la lista de archivos que son cargados por el módulo. Agregue este atributo al diccionario del descriptor:

```
'data' : ['todo_view.xml'],
```

Ahora necesitamos actualizar nuevamente el módulo para que estos cambios tengan efecto.

Gráfico 2.3 - Agregar módulo 'todo_app' al menú de Odoo

Si hace clic en ella se abrirá un formulario generado automáticamente para nuestro modelo, permitiendo agregar y modificar los registros.

Las vistas deben ser definidas por los modelos para ser expuestas a los usuarios y las usuarias, aunque Odoo es lo suficientemente amable para hacerlo automáticamente si no queremos, entonces podemos trabajar con nuestro modelo, sin tener ningún formulario o vistas definidas aún.

Hasta ahora vamos bien. Mejoremos ahora nuestra interfaz gráfica. Intente las mejoras graduales que son mostradas en las secciones siguientes, haciendo actualizaciones frecuentes del módulo, y no tenga miedo de experimentar.

Crear vistas - formulario, árbol y búsqueda

Como hemos visto, si ninguna vista es definida, Odoo automáticamente generará vistas básicas para que puedas continuar. Pero seguramente le gustará definir sus propias vistas del módulo, así que eso es lo que haremos.

Odoo soporta varios tipos de vistas, pero las tres principales son: list (lista, también llamada árbol), form (formulario), ysearch (búsqueda). Agregaremos un ejemplo de cada una a nuestro módulo.

Todas las vistas son almacenadas en la base de datos, en el modelo ir.model.view. Para agregar una vista en un módulo, declaramos un elemento <record> describiendo la vista en un archivo XML que será cargado dentro de la base de datos cuando el modelo sea instalado.

Creando una vista formulario

Vaya al menú Mensajería y debe poder ver nuestro nueva opción disponible.

Edite el XML que recién hemos creado para agregar el elemento <record> después de la apertura de la etiqueta <data>:

```
<record id="view_form_todo_task" model
="ir.ui.view">
  <field name="name">To-do Task Form</f
ield>
  <field name="model">todo.task</field>
  <field name="arch" type="xml">
    <form string="To-do Task">
      <field name="name"/>
      <field name="is_done"/>
      <field name="active" readonly="1"/>
    </form>
  </field>
</record>
```

Esto agregará un registro al modelo ir.ui.view con el identificador view_form_todo_task. Para el modelo la vista es todo.task y nombrada To-do Task Form. El nombre es solo para información, no tiene que ser único, pero debe permitir identificar fácilmente a que registro se refiere.

El atributo más importante es arch, que contiene la definición de la vista. Aquí decimos que es un formulario, y que contiene tres campos, y que decidimos hacer al campo active de solo lectura.

Formatear como un documento de negocio

Lo anterior proporciona una vista de formulario básica, pero podemos hacer algunos cambios para mejorar su apariencia. Para los modelos de documentos Odoo tiene un estilo de presentación que asemeja una hoja de papel. El formulario contiene dos elementos: una <head>, que contiene botones de acción, y un <sheet>, que contiene los campos de datos:

```
<form>
```

```
    <header>
      <!-- Buttons go here-->
    </header>
    <sheet>
      <!-- Content goes here: -->
      <field name="name"/>
      <field name="is_done"/>
    </sheet>
</form>
```

Agregar botones de acción

Los formularios pueden tener botones que ejecuten acciones. Estos son capaces de desencadenar acciones de flujo de trabajo, ejecutar Acciones de Ventana, como abrir otro formulario, o ejecutar funciones Python definidas en el modelo.

Estos pueden ser colocados en cualquier parte dentro de un formulario, pero para formularios con estilo de documentos, el sitio recomendado es en la sección <header>.

Para nuestra aplicación, agregaremos dos botones para ejecutar métodos del modelo todo.task:

```
<header>
    <button name="do_toggle_done" type="o
bject" string="Toggle Done" class="oe_highl
ight" />
    <button name="do_clear_done" type="obj
ect" string="Clear All Done" />
</header>
```

Los atributos básicos para un botón son: string con el texto que se muestra en el botón, type que hace referencia al tipo de acción que ejecuta, y name que es el identificador para esa acción. El atributo class puede aplicar estilos CSS, como un HTML común.

Organizar formularios usando grupos

La etiqueta <group> permite organizar el contenido del formulario. Colocando los elementos <group> dentro de un elemento <group> crea una disposición de dos columnas dentro del grupo externo. Se recomienda que los elementos Group tengan un nombre para hacer más fácil su extensión en otros módulos.

Usaremos esto para mejorar la organización de nuestro contenido. Cambiemos el contenido de <sheet> de nuestro formulario:

```
<sheet>
    <group name="group_top">
      <group name="group_left">
        <field name="name"/>
      </group>
      <group name="group_right">
        <field name="is_done"/>
        <field name="active" readonly="1"/>
      </group>
    </group>
</sheet>
```

La vista de formulario completa

En este momento, nuestro registro en todo_view.xml para la vista de formulario de todo.task debería lucir así:

```
<record id="view_form_todo_task" model="
ir.ui.view">
    <field name="name">To-do Task Form</f
ield>
    <field name="model">todo.task</field>
    <field name="arch" type="xml">
      <form>
        <header>
          <button name="do_toggle_done" ty
pe="object" string="Toggle Done" class="oe
_highlight" />
          <button name="do_clear_done" ty
pe="object" string="Clear All Done" />
        </header>
        <sheet>
          <group name="group_top">
            <group name="group_left">
              <field name="name"/>
            </group>
            <group name="group_right">
              <field name="is_done"/>
              <field name="active" readonly
="1" />
            </group>
          </group>
        </sheet>
      </form>
    </field>
</record>
```

Recuerde que para que los cambios tengan efecto en la base de datos de Odoo, es necesario actualizar el módulo. Para ver los cambio en el

cliente web, es necesario volver a cargar el formulario: haciendo nuevamente clic en la opción de menú que abre el formulario, o volviendo a cargar la página en el navegador (*F5* en la mayoría de los navegadores).

Ahora, agreguemos la lógica de negocio para las acciones de los botones.

Agregar vistas de lista y búsqueda

Cuando un modelo se visualiza como una lista, se esta usando una vista <tree> Las vistas de árbol son capaces de mostrar líneas organizadas por jerarquía, pero la mayoría de las veces son usadas para desplegar listas planas.

Podemos agregar la siguiente definición de una vista de árbol a todo_view.xml:

```
<record id="view_tree_todo_task" model="ir.ui.view">
   <field name="name">To-do Task Tree</field>
   <field name="model">todo.task</field>
   <field name="arch" type="xml">
      <tree colors="gray:is_done==True">
         <field name="name"/>
         <field name="is_done"/>
      </tree>
   </field>
</record>
```

Hemos definido una lista con solo dos columnas, name y is_done. También agregamos un toque extra: las líneas para las tareas finalizadas (is_done==True) son mostradas en color gris.

En la parte superior derecha de la lista Odoo muestra una campo de búsqueda. Los campos de búsqueda predefinidos y los filtros disponibles pueden ser predeterminados por una vista <search>.

Como lo hicimos anteriormente, agregaremos esto a **todo_view.xml**:

```
<record id="view_filter_todo_task" model="ir.ui.view">
   <field name="name">To-do Task Filter</field>
   <field name="model">todo.task</field>
   <field name="arch" type="xml">
      <search>
         <field name="name"/>
         <filter string="Not Done" domain="[('is_done','=',False)]"/>
         <filter string="Done" domain="[('is_done','!=',False)]"/>
      </search>
   </field>
</record>
```

Los elementos <field> definen campos que también son buscados cuando se escribe en el campo de búsqueda. Los elementos <filter> agregan condiciones predefinidas de filtro, usando la sintaxis de dominio que puede ser seleccionada por el usuario o la usuaria con un clic.

Agregar la lógica de negocio

Ahora agregaremos lógica a nuestros botones. Edite el archivo Python todo_model.py para agregar a la clase los métodos llamados por los botones.

Usaremos la API nueva introducida en Odoo 8.0. Para compatibilidad con versiones anteriores, de forma predeterminada Odoo espera la API anterior, por lo tanto para crear métodos usando la API nueva se necesitan en ellos decoradores Python. Primero necesitamos una declaración import al principio del archivo:

```
from openerp import models, fields, api
```

La acción del botón **Toggle Done** es bastante simple: solo cambia de estado (marca o desmarca) la señal **Is Done?**. La forma más simple para agregar la lógica a un registro, es usar el decorador @api.one. Aquí self representara un registro. Si la acción es llamada para un conjunto de registros, la API gestionara esto lanzando el método para cada uno de los registros.

Dentro de la clase TodoTask agregue:

```
@api.one def do_toggle_done(self):
   self.is_done = not self.is_done
   return True
```

Como puede observar, simplemente modifica el campo is_done, invirtiendo su valor. Luego los métodos pueden ser llamados desde el lado del client y siempre deben devolver algo. Si no devuelven nada, las llamadas del cliente usando el protocolo XMLRPC no funcionará. Si no

tenemos nada que devolver, la práctica común es simplemente devolver True.

Después de esto, si reiniciamos el servidor Odoo para cargar nuevamente el archivo Python, el botón **Toggle Done** debe funcionar.

Para el botón **Clear All Done** queremos ir un poco más lejos. Este debe buscar todos los registros activos que estén finalizados, y desactivarlos. Se supone que los botones de formulario solo actúan sobre los registros seleccionados, pero para mantener las cosas simples haremos un poco de trampa, y también actuará sobre los demás botones:

```
@api.multi def do_clear_done(self):
    done_recs = self.search([('is_done', '=', Tru
e)])
    done_recs.write({'active': False})
    return True
```

En los métodos decorados con @api.multi el self representa un conjunto de registros. Puede contener un único registro, cuando se usa desde un formulario, o muchos registros, cuando se usa desde la vista de lista. Ignoraremos el conjunto de registros de self y construiremos nuestro propio conjunto done_recs que contiene todas la tareas marcadas como finalizadas. Luego fijamos la señal activa como False, en todas ellas.

El search es un método de la API que devuelve los registros que cumplen con algunas condiciones. Estas condiciones son escritas en un dominio, esto es una lista de tríos. Exploraremos con mayor detalle los dominios más adelante.

Gráfico 2.4 - Lista controles de acceso de Odoo

Aquí podemos ver la ACL para el modelo mail.mail. Este indica, por grupo, las acciones permitidas en los registros.

Esta información debe ser provista por el modelo, usando un archivo de datos para cargar las líneas dentro del modelo ir.model.access. Daremos acceso completo al modelo al grupo empleado. Empleado es el grupo básico de acceso, casi todos pertenecen a este grupo.

Esto es realizado usualmente usando un archivo CSV llamado security/ir.model.access.csv. Los modelos generan identificadores

El método write fija los valores de todos los elementos en el conjunto de una vez. Los valores a escribir son definidos usando un diccionario. Usar write aquí es más eficiente que iterar a través de un conjunto de registros para asignar el valor uno por uno.

Note que @api.one no es lo más eficiente para estas acciones, ya que se ejecutará para cada uno de los registros seleccionados. La @api.multi se asegura que nuestro código sea ejecutado una sola vez incluso si hay más de un registro seleccionado. Esto puede pasar si una opción es agregada a la vista de lista.

Configurando la seguridad en el control de acceso

Debe haber notado, desde que cargamos nuestro módulo, un mensaje de alerta en en registro del servidor: **The model todo.task has no access rules, consider adding one**.

El mensaje es muy claro: nuestro modelo nuevo no tiene reglas de acceso, por lo tanto puede ser usado por cualquiera, no solo por el administrador. Como súper usuario el admin ignora las reglas de acceso, por ello somos capaces de usar el formulario sin errores. Pero debemos arreglar esto antes que otros usuarios puedan usarlo.

Para tener una muestra de la información requerida para agregar reglas de acceso a un modelo, use el cliente web y diríjase a: Configuración | Técnico | Seguridad | Lista controles de acceso.

automáticamente: para todo.task el identificador es model_todo_task. Los grupos también tienen identificadores fijados por los modelos que los crean. El grupo empleado es creado por el módulo base y tiene el identificador base.group_user. El nombre de la línea es solo informativo y es mejor si es único. Los módulos raíz usando una cadena separada por puntos con el nombre del modelo y el nombre del grupo. Siguiendo esta convención usaremos todo.task.user.

Ahora que tenemos todo lo que necesitamos saber, vamos a agregar el archivo nuevo con el siguiente contenido:

```
id,name,model_id:id,group_id:id,perm_read,
perm_write,perm_create,perm_unlink
access_todo_task_group_user,todo.task.user,
model_todo_task,base.group_user,1,1,1,1
```

No debemos olvidar agregar la referencia a este archivo nuevo en el atributo "data" del descriptor en __openerp__.py, de la siguiente manera:

```
'data': [
    'todo_view.xml',
    'security/ir.model.access.csv',
],
```

Como se hizo anteriormente, actualice el módulo para que estos cambios tengan efecto. El mensaje de advertencia debería desaparecer, y puede confirmar que los permisos sean correctos accediendo con la cuenta de usuario demo (la contraseña es también demo) e intentar ejecutar la característica de "to-do tasks".

Reglas de acceso de nivel de fila

Odoo es un sistema multi-usuario, y queremos que la aplicación **to-do task** sea privada para cada usuario. Afortunadamente, Odoo soporta reglas de acceso de nivel de fila. En el menú **Técnico** pueden encontrarse en la opción **Reglas de Registro**, junto a la **Lista de Control de Acceso**. Las reglas de registro son definidas en el modelo ir.rule. Como es costumbre, necesitamos un nombre distintivo. También necesitamos el modelo en el cual operan y el dominio para forzar la restricción de acceso. El filtro de dominio usa la misma sintaxis de dominio mencionada anteriormente, y usado a lo largo de Odoo.

Finalmente, las reglas pueden ser globales (el campo global es fijado a True) o solo para grupos particulares de seguridad. En nuestro caso, puede ser una regla global, pero para ilustrar el caso más común, la haremos como una regla específica para un grupo, aplicada solo al grupo empleados.

Debemos crear un archivo security/todo_access_rules.xml con el siguiente contenido:

```
<?xml version="1.0" encoding="utf-8"?>
  <openerp>
    <data noupdate="1">
      <record id="todo_task_user_rule" mo
del="ir.rule">
        <field name="name">ToDo Tasks
only for owner</field>
        <field name="model_id" ref="mod
el_todo_task"/>
        <field name="domain_force">
          [('create_uid','=',user.id)]
        </field>
        <field name="groups" eval="[(4,ref
('base.group_user'))]"/>
      </record>
    </data>
  </openerp>
```

Nota el atributo noupdate="1". Esto significa que estos datos no serán actualizados en las actualizaciones del módulo. Esto permitirá que sea personalizada mas adelante, debido a que las actualizaciones del módulo no destruirán los cambios realizados. Pero ten en cuenta que esto será así mientras se esté desarrollando, por lo tanto es probable que quieras fijar noupdate="0" durante el desarrollo, hasta que estés feliz con el archivo de datos.

En el campo groups también encontraras una expresión especial. Es un campo de relación uno a muchos, y tienen una sintaxis especial para operar con ellos. En este caso la tupla (4,x) indica agregar x a los registros, y x es una referencia al grupo empleados, identificado por base.group_user.

Como se hizo anteriormente, debemos agregar el archivo a __openerp__.py antes que pueda ser cargado al módulo:

```
'data':  [
  'todo_view.xml',
  'security/ir.model.access.csv',
```

```
    'security/todo_access_rules.xml',
],
```

Agregar un ícono al módulo

Nuestro módulo se ve genial. ¿Por qué no añadir un ícono para que se vea aún mejor?. Para esto solo debemos agregar al módulo el archivo static/description/icon.png con el ícono que usaremos.

Los siguientes comandos agregan un ícono copiado del módulo raíz Notes:

```
$ mkdir -p ~/odoo-dev/custom-addons/todo_app/static/description
$ cd ~/odoo-dev/custom-addons/todo_app/static/description
$ cp ../odoo/addons/note/static/description/icon.png ./
```

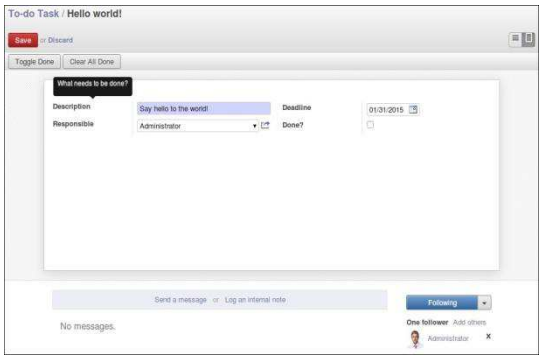


Gráfico 3.1 - Nuevo módulo para la aplicación To-Do

Camino a seguir para las características colaborativas

Aquí esta nuestro plan de trabajo para implementar la extensión de funcionalidades: - Agregar campos al modelo **Task**, como el usuario quien posee la tarea. - Modificar la lógica de negocio para operar solo en la tarea actual del usuario, en vez de todas las tareas disponibles para ser vistas por el usuario. - Agregar los campos necesarios a las vistas. - Agregar características de redes sociales: el muro de mensajes y los seguidores.

Comenzaremos creando la estructura básica para el módulo junto al módulo todo_app. Siguiendo el ejemplo de instalación del Capítulo 1, nuestros módulos estarán alojados en ~/odoo-dev/custom-addons/:

Ahora, si actualizamos la lista de módulos, nuestro módulo debe mostrarse con el ícono nuevo.

Agregar la capacidad de compartir con otros a la aplicación To-Do

Nuestra aplicación To-Do actualmente permite a los usuarios y las usuarias gestionar de forma privada sus tareas por hacer. ¿No sería grandioso llevar nuestra aplicación a otro nivel agregando características colaborativas y de redes sociales? Seriamos capaces de compartir las tareas y discutirlas con otras personas.

Haremos esto con un módulo nuevo para ampliar la funcionalidad de la aplicación To-Do creada anteriormente y agregar estas características nuevas. Esto es lo que esperamos lograr al final de este capítulo:

```
$ mkdir ~/odoo-dev/custom-addons/todo_user
$ touch ~/odoo-dev/custom-addons/todo_user/__init__.py
```

Ahora creamos el archivo __openerp__.py, con el siguiente código:

```
{
    'name': 'Multiuser To-Do',
    'description': 'Extend the To-Do app to multiuser.',
    'author': 'Daniel Reis',
    'depends': ['todo_app'],
}
```

No hemos hecho esto, pero incluir las claves "summary" y "category" puede ser importante cuando se publican módulos en la tienda de aplicaciones en línea de Odoo.

Ahora, podemos instalarlo. Debe ser suficiente con solo actualizar el **Lista de módulos** desde

el menú **Configuración**, encuentre el módulo nuevo en la lista de **Módulos locales** y haga clic en el botón Instalar. Para instrucciones más detalladas sobre como encontrar e instalar un módulo puede volver al Capítulo 1.

Ahora, comencemos a agregar las nuevas características.

Ampliando el modelo de tareas por hacer

Los modelos nuevos son definidos a través de las clases Python. Ampliarlos también es hecho a través de las clases Python, pero usando un mecanismo específico de Odoo.

Para aplicar un modelo usamos una clase Python con un atributo __inherit. Este identifica el modelo que será ampliado. La clase nueva hereda todas las características del modelo padre, y solo necesitamos declarar las modificaciones que queremos introducir.

De hecho, los modelos de Odoo existen fuera de nuestro módulo particular, en un registro central. Podemos referirnos a este registro como la piscina, y puede ser accedido desde los métodos del modelo usando self.env[<model name>]. Por ejemplo, para referirnos al modelo res.partner escribiremos self.env['res.partner'].

Para modificar un modelo de Odoo obtenemos una referencia a la clase de registro y luego ejecutamos los cambios en ella. Esto significa que esas modificaciones también estarán disponibles en cualquier otro lado donde el modelo sea usado.

En la secuencia de carga del módulo, durante un reinicio del servidor, las modificaciones solo serán visibles en los modelos cargados después. Así que, la secuencia de carga es importante y debemos asegurarnos que las dependencias del módulo están fijadas correctamente.

Agregar campos a un modelo

Ampliaremos el modelo todo.task para agregar un par de campos: el usuario responsable de la tarea, y la fecha de vencimiento.

Cree un archivo todo_task.py nuevo y declare una clase que extienda al modelo original:

```
#-*- coding: utf-8 -*-
from openerp import models, fields, api
class TodoTask(models.Model):
    _inherit = 'todo.task'
    user_id = fields.Many2one('res.users', 'Responsible')
    date_deadline = fields.Date('Deadline')
```

El nombre de la clase es local para este archivo Python, y en general es irrelevante para los otros módulos. El atributo _inheritde la clase es la clave aquí: esta le dice a Odoo que esta clase hereda el modelo todo.task. Note la ausencia del atributo _name. Este no es necesario porque ya es heredado desde el modelo padre.

Las siguientes dos líneas son declaraciones de campos comunes. El user_id representa un usuario desde el modelo Users, res.users. Es un campo de Many2one equivalente a una clave foránea en el argot de base de datos. El date_deadline es un simple campo de fecha. En el *Capítulo 5*, explicaremos con mas detalle los tipos de campos disponibles en Odoo.

Aun nos falta agregar al archivo __init__.py la declaración "import" para incluirlo en el módulo:

```
from . import todo_task
```

Para tener los campos nuevos agregados a la tabla de la base de datos soportada por el modelo, necesitamos ejecutar una actualización al módulo. Si todo sale como es esperado, debería poder ver los campos nuevos cuando revise el modelo todo.task, en el menú **Técnico**, **Estructura de base de datos | Modelos**.

Modificar los campos existentes

Como puede ver, agregar campos nuevos a un modelo existente es bastante directo. Desde Odoo 8, es posible modificar atributos en campos existentes. Esto es hecho agregando un campo con el mismo nombre, y configurando los valores solo para los atributos que serán modificados.

Por ejemplo, para agregar un comentario de ayuda a un campo name, podríamos agregar esta línea en el archivo todo_task.py:

```
name = fields.Char(help="What needs to be done?")
```

Si actualizamos el módulo, vamos a un formulario de tareas por hacer, y posicionamos el ratón sobre el campo **Descripción**, aparecerá el mensaje de texto escrito en el código anterior.

Modificar los métodos del modelo

La herencia también funciona en la lógica de negocio. Agregar métodos nuevos es simple: solo declare las funciones dentro de la clase heredada.

Para ampliar la lógica existente, un método puede ser sobreescrito declarando otro método con el mismo nombre, y el método nuevo reemplazará al anterior. Pero este puede extender el código de la clase heredada, usando la palabra clave de Python super() para llamar al método padre.

Es mejor evitar cambiar la función distintiva del método (esto es, mantener los mismos argumentos) para asegurarnos que las llamadas a este sigan funcionando adecuadamente. En caso que necesite agregar parámetros adicionales, hágalos opcionales (con un valor predeterminado).

La acción original de Clear All Done ya no es apropiada para nuestro módulos de tareas compartidas, ya que borra todas las tareas sin importar a quien le pertenecen. Necesitamos modificarla para que borre solo las tareas del usuario actual.

Para esto, se sobreescribirá el método original con una nueva versión que primero encuentre las tareas completadas del usuario actual, y luego las desactive:

```
@api.multi
def do_clear_done(self):
    domain = [('is_done', '=', True), '|', ('user_id', '=', self.env.uid), ('user_id', '=', False)]
    done_recs = self.search(domain)
    done_recs.write({'active': False})
    return True
```

Primero se listan los registros finalizados sobre los cuales se usa el método search con un filtro de búsqueda. El filtro de búsqueda sigue una sintaxis especial de Odoo referida como domain.

El filtro "domain" usado es definido en la primera instrucción: es una lista de condiciones, donde cada condición es una tupla.

Estas condiciones son unidas implícitamente con un operador AND (& en la sintaxis de dominio). Para agregar una operación OR se usa una "tubería" (|) en el lugar de la tupla, y afectara las siguientes dos condiciones. Ahondaremos más sobre este tema en el *Capítulo 6*.

El dominio usado aquí filtra todas las tareas finalizadas('is_done', '=', True) que también tengan al usuario actual como responsable ('user_id','=',self.env.uid) o no tengan fijado un usuario ('user_id', '=', False).

Lo que acabamos de hacer fue sobrescribir completamente el método padre, reemplazándolo con una implementación nueva.

Pero esto no es lo que usualmente querremos hacer. En vez de esto, ampliaremos la lógica actual y agregaremos operaciones adicionales. De lo contrario podemos dañar operaciones existentes. La lógica existente es insertada dentro de un método sobrescrito usando el comando super() de Python para llamar a la versión padre del método.

Veamos un ejemplo de esto: podemos escribir una versión mejor de do_toggle_done() que solo ejecute la acción sobre las Tareas asignadas a nuestro usuario:

```
@api.one
def do_toggle_done(self):
    if self.user_id != self.env.user:
        raise Exception('Only the responsible can do this!')
    else:
        return super(TodoTask, self).do_toggle_done()
```

Estas son las técnicas básicas para sobrescribir y ampliar la lógica de negocio definida en las clases del modelo. Veremos ahora como extender las vistas de la interfaz con las usuarias y usuarios.

Ampliar las vistas

Vistas de formulario, listas y búsqueda son definidas usando las estructuras de arco de XML. Para ampliar las vistas necesitamos una

manera de modificar este XML. Esto significa localizar los elementos XML y luego introducir modificaciones en esos puntos. Las vistas heredadas permiten esto.

Una vista heredada se ve así:

```
<record id="view_form_todo_task_inherited" model="ir.ui.view">
   <field name="name">Todo Task form – User extension</field>
   <field name="model">todo.task</field>
   <field name="inherit_id" ref="todo_app.view_form_todo_task"/>
   <field name="arch" type="xml">
      <!-- ...match and extend elements here! ... -->
   </field>
</record>
```

El campo inherit_id identifica la vista que será ampliada, a través de la referencia de su identificador externo usando el atributo especial ref. Los identificadores externos serán discutidos con mayor detalle en el *Capítulo 4*.

La forma natural de localizar los elementos XML es usando expresiones XPath. Por ejemplo, tomando la vista que fue definida en el capítulo anterior, la expresión XPath para localizar el elemento <field name="is_done">es //field[@name]='is_done'. Esta expresión encuentra un elemento field con un atributo name igual a is_done. Puede encontrar mayor información sobre XPath en: https://docs.python.org/2/library/xml.etree.elementtree.html#xpath-support.

Tener atributos "name" en los elementos es importante porque los hace mucho más fácil de seleccionar como puntos de extensión. Una vez que el punto de extensión es localizado, puede ser modificado o puede tener elementos XML agregados cerca de él.

Como un ejemplo práctico, para agregar el campo date_deadline antes del campo is_done, debemos escribir en arch:

```
<xpath expr="//field[@name]='is_done'" position="before">
   <field name="date_deadline" />
</xpath>
```

Afortunadamente Odoo proporciona una notación simplificada para eso, así que la mayoría de las veces podemos omitir la sintaxis XPath. En vez del elemento "xpath" anterior podemos usar el tipo de elementos que queramos localizar y su atributo distintivo.

Lo anterior también puede ser escrito como:

```
<field name="is_done" position="before">
   <field name="date_deadline" />
</field>`
```

Agregar campos nuevos, cerca de campos existentes es hecho frecuentemente, por lo tanto la etiqueta <field> es usada frecuentemente como el localizador. Pero cualquier otra etiqueta puede ser usada: <sheet>, <group>, , entre otras. El atributo name es generalmente la mejor opción para hacer coincidir elementos, pero a veces, podemos necesitar usar string(el texto mostrado en un "label") o la clase CSS del elemento.

El atributo de posición usado con el elemento localizador es opcional, y puede tener los siguientes valores: - after: Este es agregado al elemento padre, después del nodo de coincidencia. - before: Este es agregado al elemento padre, antes del nodo de coincidencia. - inside (el valor predeterminado): Este es anexado al contenido del nodo de coincidencia. - replace: Este reemplaza el nodo de coincidencia. Si es usado con un contenido vacío, borra un elemento. - attributes: Este modifica los atributos XML del elemento de coincidencia (más detalles luego de esta lista).

La posición del atributo nos permite modificar los atributos del elemento de coincidencia. Esto es hecho usando los elementos<attribute name="attr-name"> con los valores del atributo nuevo.

En el formulario de Tareas, tenemos el campo **Active**, pero tenerlo visible no es muy útil. Quizás podamos esconderlo de la usuaria y el usuario. Esto puede ser realizado configurando su atributo invisible:

```
<field name="active" position="attributes">
   <attribute name="invisible">1<attribute/>
</field>
```

Configurar el atributo invisible para esconder un elemento es una buena alternativa para usar el localizador de reemplazo para eliminar nodos. Debería evitarse la eliminación, ya que puede

dañar las extensiones de modelos que pueden depender del nodo eliminado.

Finalmente, podemos poner todo junto, agregar los campos nuevos, y obtener la siguiente vista heredada completa para ampliar el formulario de tareas por hacer:

```
<record id="view_form_todo_task_inherited" model="ir.ui.view">
   <field name="name">Todo Task form – User extension</field>
   <field name="model">todo.task</field>
   <field name="inherit_id" ref="todo_app.view_form_todo_task"/>
   <field name="arch" type="xml">
      <field name="name" position="after">
         <field name="user_id" />
      </field>
      <field name="is_done" position="before">
         <field name="date_deadline" />
      </field>
      <field name="name" position="attributes">
         <attribute name="string">I have to…</attribute>
      </field>
   </field>
</record>
```

Esto debe ser agregado al archivo todo_view.xml en nuestro módulo, dentro de las etiquetas <openerp> y <data>, como fue mostrado en el capítulo anterior.

No podemos olvidar agregar el atributo datos al archivo descriptor __openerp__.py:

```
'data': ['todo_view.xml'],
```

Ampliando mas vistas de árbol y búsqueda

Las extensiones de las vistas de árbol y búsqueda son también definidas usando la estructura XML arch, y pueden ser ampliadas de la misma manera que las vistas de formulario. Seguidamente mostramos un ejemplo de la ampliación de vistas de lista y búsqueda.

Para la vista de lista, queremos agregar el campo usuario:

```
<record id="view_tree_todo_task_inherited" model="ir.ui.view">
   <field name="name">Todo Task tree – User extension</field>
   <field name="model">todo.task</field>
   <field name="inherit_id" ref="todo_app.view_tree_todo_task"/>
   <field name="arch" type="xml">
      <field name="name" position="after">
         <field name="user_id" />
      </field>
   </field>
</record>
```

Para la vista de búsqueda, agregaremos una búsqueda por usuario, y filtros predefinidos para las tareas propias del usuario y tareas no asignadas a alguien.

```
<record id="view_filter_todo_task_inherited" model="ir.ui.view">
   <field name="name">Todo Task tree – User extension</field>
   <field name="model">todo.task</field>
   <field name="inherit_id" ref="todo_app.view_filter_todo_task"/>
   <field name="arch" type="xml">
      <field name="name" position="after">
         <field name="user_id" />
         <filter name="filter_my_tasks" string="My Tasks" domain="[('user_id','in',[uid,False])]" />
         <filter name="filter_not_assigned" string="Not Assigned" domain="[('user_id','=',False)]" />
      </field>
   </field>
</record>
```

No se preocupe demasiado por la sintaxis específica de las vistas. Describiremos esto con más detalle en el *Capítulo 6*.

Más sobre el uso de la herencia para ampliar los modelos

Hemos visto lo básico en lo que se refiere a la ampliación de modelos "in place", lo cual es la forma más frecuente de uso de la herencia. Pero la herencia usando el atributo _inherit tiene mayores capacidades, como la mezcla de clases.

También tenemos disponible el método de herencia delegada, usando el atributo _inherits. Esto permite a un modelo contener otros modelos de forma transparente a la vista,

mientras por detrás de escena cada modelo gestiona sus propios datos.

Exploremos esas posibilidades en más detalle.

Copiar características usando herencia por prototipo

El método que usamos anteriormente para ampliar el modelo solo usa el atributo _inherit. Definimos una clase que hereda el modelo todo.task, y le agregamos algunas características. La clase _name no fue fijada explícitamente; implícitamente fue también todo.task.

Pero usando el atributo _name nos permitió crear una mezcla de clases (mixin), incorporándolo al modelo que queremos ampliar. Aquí mostramos un ejemplo:

```
from openerp import models
class TodoTask(models.Model):
    _name = 'todo.task'
    _inherit = 'mail.thread'
```

Esto amplia el modelo todo.task copiando las características del modelo mail.thread. El modelo mail.thread implementa la mensajería de Odoo y la función de seguidores, y es reusable, por lo tanto es fácil agregar esas características a cualquier modelo.

Copiar significa que los métodos y los campos heredados estarán disponibles en el modelo heredero. Para los campos significa que estos serán creados y almacenados en las tablas de la base de datos del modelo objetivo. Los registros de datos del modelo original (heredado) y el nuevo modelo (heredero) son conservados sin relación entre ellos. Solo son compartidas las definiciones.

Estas mezclas son usadas frecuentemente como modelos abstractos, como el mail.thread usado en el ejemplo. Los modelos abstractos son como los modelos regulares excepto que no es creada ninguna representación de ellos en la base de datos. Actúan como plantillas, describen campos y la lógica para ser reusadas en modelos regulares.

Los campos que definen solo serán creados en aquellos modelos regulares que hereden de ellos. En un momento discutiremos en detalle como usar eso para agregar mail.thread y sus características de redes sociales a nuestro módulo. En la práctica cuando se usan las mezclas rara vez heredamos de modelos regulares, porque esto puede causar duplicación de las mismas estructuras de datos.

Odoo proporciona un mecanismo de herencia delegada, el cual impide la duplicación de estructuras de datos, por lo que es usualmente usada cuando se hereda de modelos regulares. Veamos esto con mayor detalle.

Integrar Modelos usando herencia delegada

La herencia delegada es el método de extensión de modelos usado con menos frecuencia, pero puede proporcionar soluciones muy convenientes. Es usada a través del atributo _inherits (note la 's' adicional) con un mapeo de diccionario de modelos heredados con campos relacionados a él.

Un buen ejemplo de esto es el modelo estándar Users, res.users, que tiene un modelo Partner anidado:

```
from openerp import models, fields

class User(models.Model):
    _name = 'res.users'
    _inherits = {'res.partner': 'partner_id'}
    partner_id = fields.Many2one('res.partner'
)
```

Con la herencia delegada el modelos res.users integra el modelo heredado res.partner, por lo tanto cuando un usuario (User) nuevo es creado, un socio (Partner) también es creado y se mantiene una referencia a este a través del campo partner_id de User. Es similar al concepto de polimorfismo en la programación orientada a objetos.

Todos los campos del modelo heredado, Partner, están disponibles como si fueran campos de User, a través del mecanismo de delegación. Por ejemplo, el nombre del socio y los campos de dirección son expuestos como campos de User, pero de hecho son almacenados en el modelo Partner enlazado, y no ocurre ninguna duplicación de la estructura de datos.

La ventaja de esto, comparada a la herencia por prototipo, es que no hay necesidad de repetir la

estructura de datos en muchas tablas, como las direcciones. Cualquier modelo que necesite incluir un dirección puede delegar esto a un modelo Partner vinculado. Y si son introducidas algunas modificaciones en los campos de dirección del socio o validaciones, estas estarán disponibles inmediatamente para todos los modelos que vinculen con él!

Usar la herencia para agregar características redes sociales

El módulo de red social (nombre técnico mail) proporciona la pizarra de mensajes que se encuentra en la parte inferior de muchos formularios, también llamado Charla Abierta (Open Chatter), los seguidores se presentan junto a la lógica relativa a mensajes y notificaciones. Esto es algo que vamos a querer agregar con frecuencia a nuestros modelos, así que aprendamos como hacerlo.

Las características de mensajería de red social son proporcionadas por el modelo mail.thread del modelo mail. Para agregarlo a un módulo personalizado necesitamos:

Que el módulo dependa de mail.

Que la clase herede de mail.thread.

Tener agregados a la vista de formulario los widgets Followers (seguidores)
y Threads (hilos).

Opcionalmente, configurar las reglas de registro para seguidores.

Sigamos esta lista de verificación:

En relación a #1, debido a que nuestro módulo ampliado depende de todo_app, el cual a su vez depende de mail, la dependencia de mail esta implícita, por lo tanto no se requiere ninguna acción.

En relación a #2, la herencia a mail.thread es hecha usando el atributo _inherit. Pero nuestra clase ampliada de tareas por hacer ya está usando el atributo _inherit.

Afortunadamente, también puede aceptar una lista de modelos desde los cuales heredar, así que podemos usar esto para hacer que incluya la herencia a mail.thread:

```
_name = 'todo.task'
_inherit = ['todo.task', 'mail.thread']
```

El modelo mail.thread es un modelo abstracto. Los modelos abstractos son como los modelos regulares excepto que no tienen una representación en la base de datos; no se crean tablas para ellos. Los modelos abstractos no están destinados a ser usados directamente. Pero se espera que sean usados en la mezcla de clases, como acabamos de hacer.

Podemos pensar en los modelos abstractos como plantillas con características listas para usar. Para crear una clase abstracta solo necesitamos usar modelos abstractos. AbstractModel en vez de models.Model.

Para la número *#3*, queremos agregar el widget de red social en la parte inferior del formulario. Podemos reusar la vista heredada que recién creamos, view_form_todo_task_inherited, y agregar esto dentro de arch:

```
<sheet position="after">
   <div class="oe_chatter">
      <field name="message_follower_ids" wi
dget="mail_followers" />
      <field name="message_ids" widget="ma
il_thread" />
   </div>
</sheet>
```

Los dos campos que hemos agregado aquí no han sido declarados explícitamente, pero son provistos por el modelo mail.thread.

El paso final es fijar las reglas de los registros de seguidores, esto solo es necesario si nuestro modelo tiene implementadas reglas de registro que limitan el acceso a otros usuarios. En este caso, necesitamos asegurarnos que los seguidores para cada registro tengan al menos acceso de lectura.

Tenemos reglas de registro en nuestro modelo de tareas por hacer así que necesitamos abordar esto, y es lo que haremos en la siguiente sección.

Modificar datos

A diferencia de las vistas, los registros de datos no tienen una estructura de arco XML y no pueden ser ampliados usando expresiones

XPath. Pero aún pueden ser modificados reemplazando valores en sus campos.

El elemento <record id="x" model="y"> está realizando una operación de inserción o actualización en un modelo: si x no existe, es creada; de otra forma, es actualizada / escrita.

Debido a que los registros en otros módulos pueden ser accedidos usando un identificador <model>.<identifier>, es perfectamente legal para nuestro módulo sobrescribir algo que fue escrito antes por otro módulo.

Como ejemplo, cambiemos la opción de menú creada por el módulo todo_app en "My To Do". Para esto agregamos lo siguiente al archivo todo_user/todo_view.xml:

```
<!-- Modify menu item -->
<record id="todo_app.menu_todo_task" model="ir.ui.menu">
   <field name="name">My To-Do</field>
</record>
<!-- Action to open To-Do Task list -->
<record model="ir.actions.act_window" id="todo_app.action_todo_task">
   <field name="context">
      {'search_default_filter_my_tasks': True}
   </field>
</record>
```

Ampliando las reglas de registro

La aplicación Tareas-por-Hacer incluye una regla de registro para asegurar que cada tarea sea solo visible para el usuario que la ha creado. Pero ahora, con la adición de las características sociales, necesitamos que los seguidores de la tarea también tengan acceso. El modelo de red social no maneja esto por si solo.

Ahora las tareas también pueden tener usuarios asignados a ellas, por lo tanto tiene más sentido tener reglas de acceso que funcionen para el usuario responsable en vez del usuario que creo la tarea.

El plan será el mismo que para la opción de menú:
sobrescribir todo_app.todo_task_user_rule para modificar el campodomain_force a un valor nuevo.

Desafortunadamente, esto no funcionará esta vez. Recuerde que el <data no_update="1"> que usamos anteriormente en el archivo XML de las reglas de seguridad: previene las operaciones posteriores de escritura.

Debido a que las actualizaciones del registro no están permitidas, necesitamos una solución alterna. Este será borrar el registro y agregar un reemplazo para este en nuestro módulo.

Para mantener las cosas organizadas, crearemos un archivo security/todo_access_rules.xml y agregaremos lo siguiente:

```
<?xml version="1.0" encoding="utf-8"?>
   <openerp>
      <data noupdate="1">
         <delete model="ir.rule" search="[('id'=',ref('todo_app.todo_task_user_rule'))]" />
         <record id="todo_task_per_user_rule" model="ir.rule">
            <field name="name">ToDo Tasks only for owner</field>
            <field name="model_id" ref="model_todo_task"/>
            <field name="groups" eval="[(4, ref('base.group_user'))]"/>
            <field name="domain_force">
               ['|', ('user_id','in', [user.id,False]), ('message_follower_ids','in',[user.partner_id.id])]
            </field>
         </record>
      </data>
   </openerp>
```

Esto encuentra y elimina la regla de registro todo_task_user_rule del
módulo todo_app, y crea una nueva regla de registrotodo_task_per_user. El filtro de dominio que usamos ahora hace la tarea visible para el usuario responsable user_id, para todo el mundo si el usuario responsable no ha sido definido (igual a False), y para todos los seguidores. La regla se ejecutará en un contexto donde el usuario este disponible y represente la sesión del usuario actual. Los seguidores son socios, no objetos User, así que en vez de user_id, necesitamos usar user.partner_id.id.

Como de costumbre, no debemos olvidar agregar el archivo nuevo al archivo descriptor __openerp__.py en el atributo "data":

```
'data': ['todo_view.xml', 'security/todo_access_rules.xml'],
```

Note que en la actualización de módulos, el elemento **<delete>** arrojará un mensaje de advertencia, porque el registro que será eliminado no existe más. Esto no es un error y la actualización se realizará con éxito, así que no es necesario preocuparse por esto.

Modelos – Estructura de los Datos de la Aplicación

En los capítulos anteriores, vimos un resumen de extremo a extremo sobre la creación de módulos nuevos para Odoo. En el Capítulo 2, se construyo una aplicación totalmente nueva, y en el Capítulo 3, exploramos la herencia y como usarla para crear un módulo de extensión para nuestra aplicación. En el Capítulo 4, discutimos como agregar datos iniciales y de demostración a nuestros módulos.

En estos resúmenes, tocamos todas las capas que componen el desarrollo de aplicaciones "backend" para Odoo. Ahora, en los siguientes capítulos, es hora de explicar con más detalle todas estas capas que conforman una aplicación: modelos, vistas, y lógica de negocio.

En este capítulo, aprenderá como diseñar las estructuras de datos que soportan una aplicación, y como representar las relaciones entre ellas.

Organizar las características de las aplicaciones en módulos

Como hicimos anteriormente, usaremos un ejemplo para ayudar a explicar los conceptos. Una de las mejores cosas de Odoo es tener la capacidad de tomar una aplicación o módulo existente y agregar, sobre este, las características que necesite. ¡Así que continuaremos mejorando nuestros módulos to-do, y pronto formaran una aplicación completa!

Es una buena práctica dividir las aplicaciones Odoo en varios módulos pequeños, cada uno responsable de una característica específica. Esto reduce la complejidad general y hace el mantenimiento y la actualización más fácil.

El problema de tener que instalar todos esos módulos individuales puede ser resuelto proporcionando un módulo de la aplicación que empaquete todas esas características, a través de sus dependencias. Para ilustrar este enfoque implementaremos las características adicionales usando módulos to-do nuevos.

Introducción al módulo todo_ui

En el capítulo anterior, primero creamos una aplicación para tareas por hacer personales, y luego la ampliamos para que las tareas por hacer pudieran ser compartidas con otras personas.

Ahora queremos llevar a nuestra aplicación a otro nivel agregándole una pizarra kanban y otras mejoras en la interfaz. La pizarra kanban nos permitirá organizar las tareas en columnas, de acuerdo a sus estados, como En Espera, Lista, Iniciada o Culminada.

Comenzaremos agregando la estructura de datos para permitir esa visión. Necesitamos agregar los estados y sería bueno si añadimos soporte para las etiquetas, permitiendo que las tareas estén organizadas por categoría.

La primera cosa que tenemos que comprender es como nuestra data estará estructurada para que podamos diseñar los Modelos que la soportan. Ya tenemos la entidad central: las tareas por hacer. Cada tarea estará en un estado, y las tareas pueden tener una o más etiquetas. Esto significa que necesitaremos agregar dos modelos adicionales, y tendrán estas relaciones:

- Cada tarea tiene un estado, y puede haber muchas tareas en un estado.
- Cada tarea puede tener muchas etiquetas, y cada etiqueta puede estar en muchas tareas.

Esto significa que la tarea tiene una relación muchos a uno con los estados, y una relación muchos a muchos con las etiquetas. Por otra parte, las relaciones inversas son: los estados tiene una relación uno a muchos con las tareas y las etiquetas tienen una relación muchos a muchos con las tareas.

Comenzaremos creando el módulo nuevo todo_ui y agregaremos los estados y los modelos de etiquetas.

Hemos estado usado el directorio ~/odoo-dev/custom-addons/ para alojar nuestros módulos. Para crear el módulo nuevo junto a los existentes, podemos usar estos comandos en la terminal:

```
$ cd ~/odoo-dev/custom-addons
```

```
$ mkdir todo_ui
$ cd todo_ui
$ touch __openerp__.py
$ touch todo_model.py
$ echo "from . Import todo_model" > __init_
_.py
```

- Luego, debemos editar el archivo manifiesto __openerp__.py con este contenido:

```
{
  'name': 'User interface improvements to th
e To-Do app',
  'description': 'User friendly features.',
  'author': 'Daniel Reis',
  'depends': ['todo_app']
}
```

Note que dependemos de todo_app y no de todo_user. En general, es buena idea mantener los módulos tan independientes como sea posible. Cuando un módulo aguas arriba es modificado, puede impactar todos los demás módulos que directa o indirectamente dependen de el. Es mejor si podemos mantener al mínimo el número de dependencias, e igualmente evitar concentrar un gran número de dependencias, como: todo_ui → todo_user → todo_app en este caso.

Ahora podemos instalar el módulo en nuestra base de datos de trabajo y comenzar con los modelos.

Crear modelos

Para que las tareas por hacer tengan una pizarra kanban, necesitamos estados. Los estados son columnas de la pizarra, y cada tarea se ajustará a una de esas columnas.

Agreguemos el siguiente código al archivo todo_ui/todo_model.py:

```
#-*- coding: utf-8 -*-
from openerp import models, fields, api

class Tag(models.Model):
   _name = 'todo.task.tag'
   name = fields.Char('Name', 40, translate=
True)

class Stage(models.Model):
   _name = 'todo.task.stage'
   _order = 'sequence,name'
   _rec_name = 'name'  # predeterminado
   _table = 'todo_task_stage' # predeterminad
o
   name = fields.Char('Name', 40, translate=
True)
   sequence = fields.Integer('Sequence')
```

- Aquí, creamos los dos Modelos nuevos a los cuales haremos referencia en las tareas por hacer.
- Enfocándonos en los estados de las tareas, tenemos una clase Python, Stage, basada en la clase models.Model, que define un modelo nuevo, todo.task.stage. También definimos dos campos, "name" y "sequence". Podemos ver algunos atributos del modelo, (con el grión bajo, _, como prefijo) esto es nuevo para nosotros. Demos le una mirada más profunda.

Atributos del modelo

- Las clases del modelo pueden tener atributos adicionales usados para controlar alguno de sus comportamientos:
- _name: Este es el identificador interno para el modelo que estamos creando.
- _order: Este fija el orden que será usado cuando se navega por los registros del modelo. Es una cadena de texto que es usada como una clausula SQL "order by", así que puede ser cualquier cosa permitida.
- _rec_name: Este indica el campo a usar como descripción del registro cuando se hace referencia a él desde campos relacionados, como una relación muchos a uno. De forma predeterminada usa el campo name, el cual esta frecuentemente presente en los modelos. Pero este atributo nos permite usar cualquier otro campo para este propósito.
- _table: Este es el nombre de la tabla de la base de datos que soporta el modelo. Usualmente, se deja para que sea calculado automáticamente, y es el nombre del modelo con el carácter de piso bajo (_) que reemplaza a los puntos. Pero puede ser configurado para indicar un nombre de tabla específico.

Para completar, también podemos tener atributos inherit e _inherits, como explicamos en el Capítulo 3.

Modelos y clases Python

- Los modelos de Odoo son representados por las clases Python. En el código precedente, tenemos una clase Python llamada Stage, basada en la clase models.Model, usada para definir el modelo nuevo todo.task.stage.
- Los modelos de Odoo son mantenidos en un registro central, también denominado como piscina - pool - en las versiones anteriores. Es un diccionario que mantiene las referencias de todas las clases de modelos disponibles en la instancia, a las cuales se les puede hacer referencia por el nombre del modelo. Específicamente, el código en un método del modelo puede usar self.env['x] o self.env.get('x') para obtener la referencia a la clase que representa el modelo x.
- Puede observar que los nombres del modelo son importantes ya que son la llave para acceder al registro. La convención para los nombres de modelo es usar una lista de palabras en minúscula unidas con puntos, como todo.task.stage. Otros ejemplos pueden verse en los módulos raíz de Odoo project.project, project.task o project.task.type.
- Debemos usar la forma singular: todo.task en vez de todo.tasks. Por cuestiones históricas se pueden encontrar módulos raíz, que no sigan dicha convención, como res.users, pero no es la norma.
- Los nombres de modelo deben ser únicos. Debido a esto, la primera palabra deberá corresponder a la aplicación principal con la cual esta relacionada el módulo. En nuestro ejemplo, es "todo". De los módulos raíz tenemos, por ejemplo, project, crm, o sale.

Por otra parte, las clases Python, son locales para el archivo Python en la cual son declaradas. El identificador usado en ellas es solo significativo para el código en ese archivo.

Debido a esto, no se requiere que los identificadores de clase tengan como prefijo a la aplicación principal a la cual están relacionados. Por ejemplo, no hay problema en llamar simplemente Stage a nuestra clase para el modelo todo.task.stage. No hay riesgo de colisión con otras posibles clases con el mismo nombre en otros módulos.

Se pueden usar dos convenciones diferentes para los identificadores de clase: **snake_case** o **CamelCase**.
Históricamente, el código Odoo ha usado el snake_case, y es aún muy frecuente encontrar clases que usan esa convención. Pero la tendencia actual en usar CamelCase, debido a que es el estándar definido para Python por la convenciones de codificación PEP8. Puede haber notado que estamos usando esta última forma.

Modelos transitorios y abstractos

En el código precedente, y en la vasta mayoría de los modelos Odoo, las clases están basadas en el clase models.Model. Este tipo de modelos tienen bases de datos persistentes: las tablas de las bases de datos son creadas para ellos y sus registros son almacenados hasta que son borrados explícitamente.

Pero Odoo proporciona otros dos tipos de modelo: modelos Transitorios y Abstractos.

Los **modelos transitorios** están basados en la clase models.TransientModel y son usados para interacción tipo asistente con el usuario y la usuaria. Sus datos son aún almacenados en la base de datos, pero se espera que sea temporal. Un proceso de reciclaje limpia periódicamente los datos viejos de esas tablas.

Los **modelos abstractos** están basados en la clase models.AbstractModel y no tienen almacén vinculado a ellos. Actúan como una característica de re-uso configurada para ser mezclada con otros modelos. Esto es hecho usando las capacidades de herencia de Odoo.

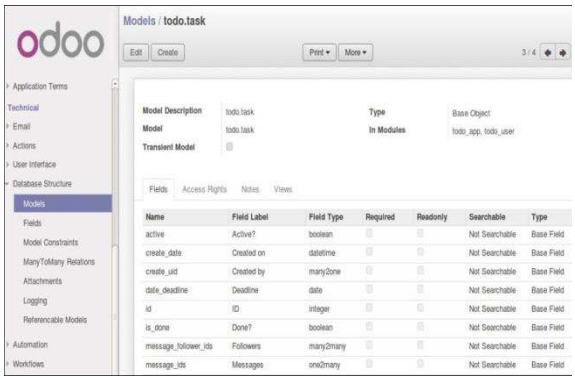


Grafico 5.1 - Vista de la estructura de base de datos del modelo todo.task

Inspeccionar modelos existentes

La información sobre los modelos y los campos creados con clases Python esta disponible a través de la interfaz. En el menú principal de Configuración, seleccione la opción de menú Técnico | Estructura de base de datos | Modelos. Allí, encontrará la lista de todos los modelos disponibles en la base de datos. Al hacer clic en un modelo de la lista se abrirá un formulario con sus detalles.

Esta es una buena herramienta para inspeccionar la estructura de un Modelo, ya que se tiene en un solo lugar el resultado de todas las adiciones que pueden venir de diferentes módulos. En este caso, como puede observar en el campo **En los módulos**, en la parte superior derecha, las definiciones de todo.task vienen de los módulos todo_app y todo_user.

En el área inferior, tenemos disponibles algunas etiquetas informativas: una referencia rápida de los Campos del modelo, los Derechos de Acceso concedidos, y también lista las Vistas disponibles para este modelo.

Podemos encontrar el Identificador Externo del modelo, activando el **Menú de Desarrollo** y accediendo a la opción **Ver metadatos**. Estos son generados automáticamente pero bastante predecibles: para el modelo todo.task, el Identificador Externo es model_todo_task.

Crear campos

Después de crear un modelo nuevo, el siguiente paso es agregar los campos. Vamos a explorar diferentes tipos de campos disponibles en Odoo.

Tipos básicos de campos

Ahora tenemos un modelo Stage y vamos a ampliarlo para agregar algunos campos adicionales. Debemos editar el archivo todo_ui/todo_model.py, removiendo algunos atributos innecesarios incluidos antes con propósitos descriptivos:

```
class Stage(models.Model):
    _name  = 'todo.task.stage'
    _order = 'sequence,name'

    # Campos de cadena de caracteres:
    name  = fields.Char('Name',40)
    desc  = fields.Text('Description')
    state  =  fields.Selection([('draft','New'),('op
en','Started'), ('done','Closed')],'State')
    docs  = fields.Html('Documentation')

    # Campos numéricos:
    sequence      = fields.Integer('Sequence')
    perc_complete = fields.Float('% Complete'
,(3,2))

    # Campos de fecha:
    date_effective = fields.Date('Effective Date'
)
    date_changed   = fields.Datetime('Last Cha
nged')

    # Otros campos:
    fold  = fields.Boolean('Folded?')
    image = fields.Binary('Image')
```

Aquí tenemos un ejemplo de tipos de campos no relacionales disponibles en Odoo, con los argumentos básicos esperados por cada función. Para la mayoría, el primer argumento es el título del campo, que corresponde al atributo palabra clave de cadena. Es un argumento opcional, pero se recomienda colocarlo. De lo contrario, sera generado automáticamente un título por el nombre del campo.

Existe una convención para los campos de fecha que usa date como prefijo para el nombre. Por ejemplo, deberíamos usar date_effective en vez de effective_date. Esto también puede aplicarse a otros campos, como “amount_”, “price_” o “qty_”.

Algunos otros argumentos están disponibles para la mayoría de los tipos de campo:

Char, acepta un segundo argumento opcional, "size", que corresponde al tamaño máximo del texto. Es recomendable usarlo solo si se tiene una buena razón.

Text, se diferencia de Char en que puede albergar texto de varias líneas, pero espera los mismos argumentos.

Selecction, es una lista de selección desplegable. El primer argumento es la lista de opciones seleccionables y el segundo es la cadena de título. La lista de selección es una tupla ('value', 'Title') para el valor almacenado en la base de datos y la cadena de descripción correspondiente. Cuando se amplía a través de la herencia, el argumento selection_add puede ser usado para agregar opciones a la lista de selección existente.

Html, es almacenado como un campo de texto, pero tiene un manejo específico para presentar el contenido HTML en la interfaz.

Integer, solo espera un argumento de cadena de texto para el campo de título.

Float, tiene un argumento opcional, una tupla (x,y) con los campos de precisión: 'x' como el número total de dígitos; 'y' representa los dígitos decimales.

Date y Datetime, estos datos son almacenados en formato UTC. Se realizan conversiones automáticas, basadas en las preferencias del usuario o la usuaria, disponibles a través del contexto de la sesión de usuario. Esto es discutido con mayor detalle en el Capítulo 6.

Boolean, solo espera sea fijado el campo de título, incluso si es opcional.

Binary también espera este único argumento.

Además de estos, también existen los campos relacionales, los cuales serán introducidos en este mismo capítulo. Pero por ahora, hay mucho que aprender sobre los tipos de campos y sus atributos.

Atributos de campo comunes

Los campos también tienen un conjunto de atributos los cuales podemos usar, y los explicaremos aquí con más detalle:

- string, es el título del campo, usado como su etiqueta en la UI. La mayoría de las veces no es usado como palabra clave, ya que puede ser fijado como un argumento de posición.
- default, fija un valor predefinido para el campo. Puede ser un valor estático o uno fijado anticipadamente, pudiendo ser una referencia a una función o una expresión lambda.
- size, aplica solo para los campos Char, y pueden fijar el tamaño máximo permitido.
- translate, aplica para los campos de texto, Char, Text y Html, y hacen que los campos puedan ser traducidos: puede tener varios valores para diferentes idiomas.
- help, proporciona el texto de ayuda desplegable mostrado a los usuarios y usuarias.
- readonly = True, hace que el campo no pueda ser editado en la interfaz.
- required = True, hace que el campo sea obligatorio.
- index = True, creara un índice en la base de datos para el campo.
- copy = False, hace que el campo sea ignorado cuando se usa la función Copiar. Los campos no relacionados de forma predeterminada pueden ser copiados.
- groups, permite limitar la visibilidad y el acceso a los campos solo a determinados grupos. Es una lista de cadenas de texto separadas por comas, que contiene los ID XML del grupo de seguridad.
- states, espera un diccionario para los atributos de la UI dependiendo de los valores de estado del campo. Por ejemplo:states={'done':[('readonly', True)]}. Los atributos que pueden ser usados son, "readonly", "required" e "invisible".

Para completar, a veces son usados dos atributos más cuando se actualiza entre versiones principales de Odoo:

- deprecated = True, registra un mensaje de alerta en cualquier momento que el campo sea usado.

- oldname = 'field', es usado cuando un campo es re-nombrado en una versión nueva, permitiendo que la data en el campo viejo sea copiada automáticamente dentro del campo nuevo.

Nombres de campo reservados

- Unos cuantos nombres de campo estan reservados para ser usados por el ORM:
- id, es un número generado automáticamente que identifica de forma única a cada registro, y es usado como clave primaria en la base de datos. Es agregado automáticamente a cada modelo.
- Los siguientes campos son creados automáticamente en los modelos nuevos, a menos que sea fijado el atributo _log_access=False:
- create_uid, para el usuario que crea el registro.
- created_date, para la fecha y la hora en que el registro es creado.
- write_uid, para el último usuario que modifica el registro.
- write_date, para la última fecha y hora en que el registro fue modificado.
- Esta información esta disponible desde el cliente web, usando el menú de Desarrollo y seleccionando la opción Ver metadatos.
- Hay algunos efectos integrados que esperan nombres de campo específicos. Debemos evitar usarlos para otros propósitos que aquellos para los que fueron creados. Algunos de ellos incluso están reservados y no pueden ser usados para ningún otro propósito:
- name, es usado de forma predeterminada como el nombre del registro que será mostrado. Usualmente es un Char, pero se permiten otros tipos de campos. Puede ser sobre escrito configurando el atributo _rec_name del modelo.
- active (tipo Boolean), permite desactivar registros. Registros con active==False serán excluidos automáticamente de las consultas. Para acceder a ellos debe ser agregada la condición ('active','=', False) al dominio de búsqueda o agregar'active_test':False al contexto actual.
- sequence (tipo Integer), si esta presente en una vista de lista, permite definir manualmente el orden de los registros. Para funcionar correctamente debe estar también presente en el _order del modelo.
- state (tipo Selection), representa los estados básicos del ciclo de vida del registro, y puede ser usado por el atributo "field" del estado para modificar de forma dinámica la vista: algunos campos de formulario pueden ser de solo lectura, requeridos o invisibles en estados específicos del registro.
- parent_id, parent_left, y parent_right; tienen significado especial para las relaciones jerárquicas padre/hijo. En un momento las discutiremos con mayor detalle.
- Hasta ahora hemos discutido los valores escalares de los campos. Pero una buena parte de una estructura de datos de la aplicación es sobre la descripción de relaciones entre entidades. Veamos algo sobre esto ahora.

Relaciones entre modelos

- Viendo nuestro diseño del módulo, tenemos estas relaciones:
- Cada tarea tiene un estado – esta es una relación muchos a uno, también conocida como una clave foránea. La relación inversa es de uno a muchos, que significa que cada estado puede tener muchas tareas.
- Cada tarea puede tener muchas etiquetas – esta es una relación muchos a muchos. La relación inversa, obviamente, es también una relación muchos a muchos, debido a que cada etiqueta puede también tener muchas tareas.
- Agreguemos los campos de relación correspondientes al archivo todo_ui/todo_model.py:

```
class TodoTask(models.Model):
    _inherit = 'todo.task'
    stage_id = fields.Many2one('todo.task.stage', 'Stage')
    tag_ids = fields.Many2many('todo.task.tag', string='Tags')
```

- El código anterior muestra la sintaxis básica para estos campos. Configurando el modelo relacionado y el campo de título. La convención para los nombres de campo relacionales es agregar a los nombres de campos _id o _ids, para las relaciones de uno y muchos, respectivamente.

- Como ejercicio puede intentar agregar en los modelos relacionados, las relaciones inversas correspondientes: La relación inversa de Many2one es un campo One2many en los estados: cada estado puede tener muchas tareas. Deberíamos agregar este campo a la clase Stage. La relación inversa de Many2many es también un campo Many2many en las etiquetas: cada etiqueta puede ser usada en muchas tareas.
- Veamos con mayor detalle las definiciones de los campos relacionales.

Relaciones muchos a uno

- Many2one, acepta dos argumentos de posición: el modelo relacionado (que corresponde al argumento de palabra clave del comodel) y la cadena de título. Este crea un campo en la tabla de la base de datos con una clave foránea a la tabla relacionada.
- Algunos nombres adicionales de argumentos también están disponibles para ser usados con estos tipos de campo:
- ondelete, define lo que pasa cuando el registro relacionado es eliminado. De forma predeterminada esta fijado como null, lo que significa que al ser eliminado el registro relacionado se fija a un valor vacío. Otros valores posibles son "restrict", que arroja un error que previene la eliminación, y "cascade" que también elimina este registro.
- context y domain, son significativos para las vistas del cliente. Pueden ser configurados en el modelo para ser usados de forma predeterminada en cualquier vista donde sea usado el campo. Estos serán explicados con más detalle en el Capítulo 6.
- auto_join = True, permite que el ORM use uniones SQL haciendo búsquedas usando esta relación. De forma predeterminada esto esta fijado como False para reforzar las reglas de seguridad. Si son usadas uniones, las reglas de seguridad serán pasadas por alto, y el usuario podrá tener acceso a los registros relacionados que las reglas de seguridad no le permitirían, pero las consultas SQL serán más eficientes y se ejecutarán con mayor rapidez.

Relaciones muchos a muchos

La forma mas simple de la relación Many2many acepta un argumento para el modelo relacionado, y es recomendable también proporcionar el argumento de cadena con el título del campo.

En el nivel de base de datos, esto no agrega ninguna columna a las tablas existentes. Por el contrario, automáticamente crea una tabla nueva de relación de solo dos campos con las claves foráneas de las tablas relacionadas. El nombre de la tabla de relación es el nombre de ambas tablas unidos por un símbolo de guión bajo (_) con _rel anexado.

Estas configuraciones predeterminadas pueden ser sobre escritas manualmente. Una forma de hacerlo es usar la forma larga para la definición del campo:

```
# TodoTask class: Task <-> relación Tag (for
ma larga):
tag_ids = fields.Many2many( 'todo.task.tag',
# modelo relacionado
                    'todo_task_tag_rel', # nombr
e de la tabla de relación
                    'task_id', # campo para "este
" registro
                    'tag_id', # campo para "otro
" registro
                     string='Tasks')
```

Note que los argumentos adicionales son opcionales. Podemos simplemente fijar el nombre para la tabla de relación y dejar que los nombres de los campos usen la configuración predeterminada.

Si prefiere, puede usar la forma larga usando los argumentos de palabra clave:

```
# TodoTask class: Task  <-> relación Tag (for
ma larga):
tag_ids = fields.Many2many(comodel_name=
'todo.task.tag', # modelo relacionado
                    relation='todo_task_tag_rel',
# nombre de la tabla de relación
                    column1='task_id',  # campo
para "este" registro
                    column2='tag_id', # campo p
ara "otro" registro
                    string='Tasks')
```

Como los campos muchos a uno, los campos muchos a muchos también soportan los atributos de palabra clave de dominio y contexto.

En algunas raras ocasiones tendremos que usar estas formas largas para sobre escribir las configuraciones automáticas predeterminadas, en particular, cuando los modelos relacionados tengan nombres largos o cuando necesitemos una segunda relación muchos a muchos entre los mismos modelos.

Lo inverso a la relación Many2many es también un campo Many2many. Si también agregamos un campo Many2many a las etiquetas, Odoo infiere que esta relación de muchos a muchos es la inversa a la del modelo de tareas.

La relación inversa entre tareas y etiquetas puede ser implementada así:

```
# class Tag(models.Model): #
    _name = 'todo.task.tag'

    #Tag class relación a Tasks:
    task_ids = fields.Many2many( 'todo.task',
# modelo relacionado
                        string='Tasks')
```

Relaciones inversas de uno a muchos

La inversa de Many2many puede ser agregada al otro extremo de la relación. Esto no tiene un impacto real en la estructura de la base de datos, pero nos permite navegar fácilmente desde "un" lado a "muchos" lados de los registros. Un caso típico es la relación entre un encabezado de un documento y sus líneas.

En nuestro ejemplo, con una relación inversa One2many en estados, fácilmente podemos listar todas las tareas que se encuentran en un estado. Para agregar esta relación inversa a los estados, agregue el código mostrado a continuación:

```
# class Stage(models.Model): #
    _name = 'todo.task.stage'

    #Stage class relación con Tasks:
    tasks = fields.One2many('todo.task',# mod
elo relacionado
                    'stage_id',# campo para "est
e" en el modelo relacionado
                    'Tasks in this stage')
```

One2many acepta tres argumentos de posición: el modelo relacionado, el nombre del campo en aquel modelo que referencia este registro, y la cadena de título. Los dos primeros corresponden a los argumentos comodel_name e inverse_name.

Los parámetros adicionales disponibles son los mismos que para el muchos a uno: contexto, dominio, ondelete (aquí actúa en el lado "muchos" de la relación), y auto_join.

Relaciones jerárquicas

Las relaciones padre-hijo pueden ser representadas usando una relación Many2one al mismo modelo, para dejar que cada registro haga referencia a su padre. Y la inversa One2many hace más fácil para un padre mantener el registro de sus hijos.

Odoo también provee soporte mejorado para estas estructuras de datos jerárquicas: navegación más rápida a través de árboles hermanos, y búsquedas más simples con el operador child_of en las expresiones de dominio.

Para habilitar esas características debemos configurar el atributo _parent_store y agregar los campos de ayuda: parent_lefty parent_right. Tenga en cuenta que estas operaciones adicionales traen como consecuencia penalizaciones en materia de almacenamiento y ejecución, así que es mejor usarlo cuando se espere ejecutar más lecturas que escrituras, como es el caso de un árbol de categorías.

Revisando el modelo de etiquetas definido en el archivo todo_ui/todo_model.py, ahora editaremos para que luzca así:

```
class Tags(models.Model):
    _name        = 'todo.task.tag'
    _parent_store = True
    #_parent_name  = 'parent_id'
    name = fields.Char('Name')
    parent_id     = fields.Many2one('todo.task.t
ag','Parent Tag', ondelete='restrict')
    parent_left   = fields.Integer('Parent Left',
index=True)
    parent_right  = fields.Integer('Parent Righ
t', index=True)
```

Aquí tenemos un modelo básico, con un campos parent_id que referencia al registro padre, y el atributo adicional _parent_store para agregar soporte a búsquedas jerárquicas.

Se espera que el campo que hace referencia al padre sea nombrado parent_id. Pero puede usarse cualquier otro nombre declarándolo con el atributo _parent_name.

También, es conveniente agregar un campo con el hijo directo del registro:

```
child_ids = fields.One2many('todo.task.tag', 'parent_id', 'Child Tags')
```

Hacer referencia a campos usando relaciones dinámicas

Hasta ahora, los campos de relación que hemos visto puede solamente hacer referencia a un modelo. El tipo de campo Reference no tiene esta limitación y admite relaciones dinámicas: el mismo campo es capaz de hacer referencia a más de un modelo.

Podemos usarlo para agregar un campo, "Refers to", a Tareas por Hacer que pueda hacer referencia a un User o un Partner:

```
# class TodoTask(models.Model):
    refers_to = fields.Reference([('res.user', 'User'),('res.partner', 'Partner')], 'Refers to')
```

Puede observar que la definición del campo es similar al campo Selection, pero aquí la lista de selección contiene los modelos que pueden ser usados. En la interfaz, el usuario o la usuaria seleccionará un modelo de la lista, y luego elegirá un registro de ese modelo.

Esto puede ser llevado a otro nivel de flexibilidad: existe una tabla de configuración de Modelos Referenciables para configurar los modelos que pueden ser usados en campos Reference. Esta disponible en el menú **Configuración** | **Técnico** | **Estructuras de base de datos**. Cuando se crea un campo como este podemos ajustarlo para que use cualquier modelo registrado allí, con la ayuda de la función referencable_models() en el módulo openerp.addons.res.res_request. En la versión 8 de Odoo, todavía se usa la versión antigua de la API, así que necesitamos empaquetarlo para usarlo con la API nueva:

```
from openerp.addons.base.res import res_request

def referencable_models(self):
    return res_request.referencable_model(self, self.env.cr, self.env.uid, context=self.env.context)
```

Usando el código anterior, la versión revisada del campo "Refers to" sera así:

```
# class TodoTask(models.Model):
    refers_to = fields.Reference(referencable_models, 'Refers to')
```

Campos calculados

Los campos pueden tener valores calculados por una función, en vez de simplemente leer un valor almacenado en una base de datos. Un campo calculado es declarado como un campo regular, pero tiene el argumento "compute" adicional con el nombre de la función que se usará para calcularlo.

En la mayoría de los casos los campos calculados involucran alguna lógica de negocio, por lo tanto este tema se desarrollara con más profundidad en el Capítulo 7. Igual podemos explicarlo aquí, pero manteniendo la lógica de negocio lo más simple posible.

Trabajamos en un ejemplo: los estados tienen un campo "fold". Agregaremos a las tareas un campo calculado con la marca "Folded?" para el estado correspondiente.

Debemos editar el modelo TodoTask en el archivo todo_ui/todo_model.py para agregar lo siguiente:

```
# class TodoTask(models.Model):
    stage_fold = fields.Boolean('Stage Folded?', compute='_compute_stage_fold')
    @api.one
    @api.depends('stage_id.fold')

def _compute_stage_fold(self):
    self.stage_fold = self.stage_id.fold
```

El código anterior agrega un campo nuevo stage_fold y el método _compute_stage_fold que sera usado para calcular el campo. El nombre de la función es pasado como una cadena, pero también es posible pasarla como una referencia obligatoria (el identificador de la función son comillas).

Debido a que estamos usando el decorador @api.one, self tendrá un solo registro. Si en vez de esto usamos @api.multi,

representara un conjunto de registros y nuestro código necesitará gestionar la iteración sobre cada registro.

El @api.depends es necesario si el calculo usa otros campos: le dice al servidor cuando re-calcular valores almacenados o en cache. Este acepta uno o mas nombres de campo como argumento y la notación de puntos puede ser usada para seguir las relaciones de campo.

Se espera que la función de calculo asigne un valor al campo o campos a calcular. Si no lo hace, arrojara un error. Debido a que self es un objeto de registro, nuestro calculo es simplemente para obtener el campo "Folded?" usando self.stage_id.fold. El resultado es conseguido asignando ese valor (escribiéndolo) en el campo calculado, self.stage_fold.

No trabajaremos aún en las vistas para este módulo, pero puede hacer una edición rápida al formulario de tareas para confirmar si el campo calculado esta funcionando como es esperado: usando el menú de **Desarrollo** escoja la opción **Editar Vista** y agregue el campo directamente en el XML del formulario. No se preocupe: será reemplazado por una vista limpia del módulo en la próxima actualización.

Buscar y escribir en campos calculados

El campo calculado que acabamos de crear puede ser leído, pero no se puede realizar una búsqueda ni escribir en el. Esto puede ser habilitado proporcionando funciones especiales para esto. A lo largo de la función de calculo también podemos colocar una función de búsqueda, que implemente la lógica de búsqueda, y la función inversa, que implemente la lógica de escritura.

Para hacer esto, nuestra declaración de campo calculado se convertirá en esto:

```
# class TodoTask(models.Model):
    stage_fold = fields.Boolean
        string   = 'Stage Folded?',
        compute  ='_compute_stage_fold',
                # store=False) # predeterminado
        search   ='_search_stage_fold',
        inverse  ='_write_stage_fold')
```

Las funciones soportadas son:

```
def _search_stage_fold(self, operator, value):
    return [('stage_id.fold', operator, value)]

def _write_stage_fold(self):
    self.stage_id.fold = self.stage_fold
```

La función de búsqueda es llamada en cuanto es encontrada en este campo una condición (campo, operador, valor) dentro de una expresión de dominio de búsqueda.

La función inversa realiza la lógica reversa del cálculo, para hallar el valor que sera escrito en el campo de origen. En nuestro ejemplo, es solo escribir en stage_id.fold.

Guardar campos calculados

Los valores de los campos calculados también pueden ser almacenados en la base de datos, configurando "store" a "True" en su definición. Estos serán calculados cuando cualquiera de sus dependencias cambie. Debido a que los valores ahora estarán almacenados, pueden ser buscados como un campo regular, entonces no es necesaria una función de búsqueda.

Campos relacionados

Los campos calculados que implementamos en la sección anterior son un caso especial que puede ser gestionado automáticamente por Odoo. El mismo efecto puede ser logrado usando campos Relacionados. Estos hacen disponibles, de forma directa en un módulo, los campos que pertenecen a un modelo relacionado, que son accesibles usando la notación de puntos. Esto posibilita su uso en los casos en que la notación de puntos no pueda usarse, como los formularos de UI.

Para crear un campo relacionado, declaramos un campo del tipo necesario, como en los campos calculados regulares, y en vez de calcularlo, usamos el atributo "related" indicando la cadena de notación por puntos para alcanzar el campo deseado.

Las tareas por hacer están organizadas en estados personalizables y a su vez esto forma un mapa en los estados básicos. Los pondremos disponibles en las tareas, y usaremos esto para la lógica del lado del cliente en la próximo capítulo.

Agregaremos un campo calculado en el modelo tarea, similar a como hicimos a "stage_fold", pero ahora usando un campo "Related":

```
# class TodoTask(models.Model):
    stage_state = fields.Selection(related='stage
_id.state', string='Stage State')
```

Detrás del escenario, los campos "Related" son solo campos calculados que convenientemente implementan las funciones de búsqueda e inversa. Esto significa que podemos realizar búsquedas y escribir en ellos sin tener que agregar código adicional.

Restricciones del Modelo

Para reforzar la integridad de los datos, los modelos también soportan dos tipos de restricciones: SQL y Python.

Las restricciones SQL son agregadas a la definición de la tabla en la base de datos e implementadas por PostgreSQL. Son definidas usando el atributo de clase _sql_constraints. Este es una lista de tuplas con el nombre del identificador de la restricción, el SQL para la restricción, y el mensaje de error que se usara.

Un caso común es agregar restricciones únicas a los modelos. Suponga que no queremos permitir que el mismo usuario tenga dos tareas activas con el mismo título:

```
# class TodoTask(models.Model):
    _sql_constraints = [
        ('todo_task_name_uniq',
         'UNIQUE (name, user_id, active)',
         'Task title must be unique!')]
```

Debido a que estamos usando el campo user_id agregado por el módulo todo_user, esta dependencia debe ser agregada a la clave "depends" del archivo manifiesto __openerp__.py.

Las restricciones Python pueden usar un pedazo arbitrario de código para verificar las condiciones. La función de verificación necesita ser decorada con @api.constrains indicando la lista de campos involucrados en la verificación. La validación es activada cuando cualquiera de ellos es modificado, y arrojara una excepción si la condición falla:

```
from openerp.exceptions import ValidationE
rror

# class TodoTask(models.Model):
    @api.one
    @api.constrains('name')
    def _check_name_size(self):
        if len(self.name) < 5:
            raise ValidationError('Must have 5 c
hars!')
```

El ejemplo anterior previene que el título de las tareas sean almacenados con menos de 5 caracteres.

Vistas – Diseñar la Interfaz

Este capítulo le ayudara a construir la interfaz gráfica para sus aplicaciones. Hay varios tipos disponibles de vistas y widgets. Los conceptos de contexto y dominio también juegan un papel fundamental en la mejora de la experiencia del usuario y la usuaria, y aprenderá más sobre esto.

- El módulo todo_ui tiene lista la capa de modelo, y ahora necesita la capa de vista con la interfaz. Agregaremos elementos nuevos a la IU y modificaremos las vistas existentes que fueron agregadas en capítulos anteriores.
- La mejor manera de modificar vistas existentes es usar la herencia, como se explico en el Capítulo 3. Sin embargo, para mejorar la claridad en la explicación, sobre escribiremos las vistas existentes, y las reemplazaremos por unas vistas completamente nuevas. Esto hará que los temas sean más fáciles de entender y seguir.

Es necesario agregar un archivo XML nuevo al módulo, así que comenzaremos por editar el archivo manifiesto __openerp__.py. Necesitamos usar algunos campos del módulo todo_user, para que sea configurado como una dependencia:

```
{ 'name': 'User interface improvements to the
To-Do app',
  'description': 'User friendly features.',
  'author': 'Daniel Reis',
  'depends': ['todo_user'],
  'data': ['todo_view.xml']
}
```

Comencemos con las opciones de menú y las acciones de ventana.

Acciones de ventana

Las acciones de ventana dan instrucciones a la interfaz del lado del cliente. Cuando un usuario o una usuaria hace clic en una opción de menú o en un botón para abrir un formulario, es la acción subyacente la que da instrucciones a la interfaz sobre lo que debe hacer.

Comenzaremos por crear la acción de ventana que será usada en las opciones de manú, para abrir las vistas de las tareas por hacer y de los estados. Cree el archivo de datos todo_view.xml con el siguiente código:

```
<?xml version="1.0"?>
  <openerp>
    <data>
      <act_window id="action_todo_stage" name="To-Do Task Stages" res_model="todo.task.stage" view_mode="tree,form"/>
      <act_window id="todo_app.action_todo_task" name="To-Do Tasks" res_model="todo.task" view_mode="tree,form,calendar,gantt,graph" target="current "context="{'default_user_id': uid}" domain="[]" limit="80"/>
      <act_window id="action_todo_task_stage" name="To-Do Task Stages" res_model="todo.task.stage" src_model="todo.task" multi="False"/>
    </data>
  </openerp>
```

Las acciones de ventana se almacenan en el modelo ir.actions.act_window, y pueden ser definidas en archivos XML usando el acceso directo <act_window> que recién usamos.

La primera acción abre el modelo de estados de la tarea, y solo usa los atributos básicos para una acción de ventana.

La segunda acción usa un ID en el espacio de nombre de todo_app para sobre escribir la acción original de tareas por hacer del módulo todo_app. Esta usa los atributos de aciones de ventana más relevantes.

- name: Este es el título mostrado en las vistas abiertas a través de esta acción.
- res_model: Es el identificador del modelo de destino.
- view_mode: Son los tipos de vista que estarán disponibles. El orden es relevenate y el primero de la lista será la vista que se abrirá de forma predeterminada.
- target: Si es fijado como "new", la vista se abrirá en una ventana de dialogo.De forma predeterminada esta fijado a "current", por lo que abre la vista en el área principal de contenido.
- context: Este fija información de contexto en las vistas de destino, la cual puede ser usada para establecer valores predeterminados en campos o filtros activos, entre otras cosas. Veremos más detalles sobre esto en este mismo capítulo.
- domain: Es una expresión de dominio que establece un filtro para los registros que estarán disponibles en las vistas abiertas.
- limit: Es el número de registros por cada página con vista de lista, 80 es el número predefinido.

La acción de ventana ya incluye los otros tipos de vista las cuales estaremos examinando en este capítulo: calendar, Gantt y gráfico. Una vez que estos cambios son instalados, los botones correspondientes serán mostrados en la esquina superior derecha, junto a los botones de lista y formulario. Note que esto no funcionará hasta crear las vistas correspondientes.

La tercera acción de ventana demuestra como agregar una opción bajo el botón "Mas", en la parte superior de la vista. Estos son los atributos usados para realizar esto:

- multi: Si esta fijado a "True", estará disponible en la vista de lista. De lo contrario, estará disponible en la vista de formulario.

Opciones de menú

Las opciones de menú se almacenan en el modelo ir.ui.menu, y pueden ser encontradas en el menú Configuraciones navegando a través de Técnico | Interfaz de Usuario | Opciones de Menú. Si buscamos Mensajería, veremos que tiene como submenú Organizador. Con la ayuda de las herramientas de desarrollo podemos encontrar el ID del XML para esa opción de menú: la cual es mail.mail_my_stuff.

Reemplazaremos la opción de menú existente en Tareas por Hacer con un submenú que puede encontrarse navegando a través de Mensajería | Organizador. En el todo_view.xml, despues de

las acciones de ventana, agregue el siguiente código:

```
<menuitem id="menu_todo_task_main" name="To-Do" parent="mail.mail_my_stuff"/>
<menuitem id="todo_app.menu_todo_task" name="To-Do Tasks" parent="menu_todo_task_main" sequence="10" action="todo_app.action_todo_task"/>
<menuitem id="menu_todo_task_stage" name="To-Do Stages" parent="menu_todo_task_main" sequence="20" action="action_todo_stage"/>
```

La opción de menú "data" para el modelo ir.ui.menu también puede cargarse usando el elemento de acceso directo <menuitem>, como se uso en el código anterior.

El primer elemento del menú, "To-Do", es hijo de la opción de menú Organizador mail.mail_my_stuff. No tiene ninguna acción asignada, debido a que será usada como padre para las próximas dos opciones.

El segundo elemento del menú re escribe la opción definida en el módulo todo_app para ser re ubicada bajo el elemento "To-Do" del menú principal.

El tercer elemento del menú agrega una nueva opción para acceder a los estados. Necesitaremos un orden para agregar algunos datos que permitan usar los estados en las tareas por hacer.

Contexto y dominio

- Nos hemos referido varias veces al contexto y al dominio. También hemos visto que las acciones de ventana pueden fijar valores en estos, y que los campos relacionales pueden usarlos en sus atributos. Ambos conceptos son útiles para proveer interfaces mas sofisticadas. Veamos como.

Contexto de sesión

El contexto es un diccionario que contiene datos de sesión usados por las vistas en el lado del cliente y por los procesos del servidor. Puede transportar información desde una vista hasta otra, o hasta la lógica del lado del servidor. Es usado frecuentemente por las acciones de ventana y por los campos relacionales para enviar información a las vistas abiertas a través de ellos.

Odoo estable en el contexto alguna información básica sobre la sesión actual. La información inicial de sesión puede verse así:

```
{'lang': 'en_US',  'tz': 'Europe/Brussels', 'uid': 1}
```

Tenemos información del ID de usuario actual, y las preferencias de idioma y zona horaria para la sesión de usuario.

Cuando se usa una acción en el cliente, como hacer clic en un botón, se agrega información al contexto sobre los registros seleccionados actualmente:

active_id es el ID del registro seleccionado en el formulario,

active_model es el modelo de los registros actuales,

active_ids es la lista de los ID seleccionados en la vista de árbol/lista.

El contexto también puede usarse para proveed valores predeterminados en los campos o habilitar filtros en la vista de destino.

Para fijar el valor predeterminado en el campo user_id, que corresponda a la sesión actual de usuario, debemos usar:

```
{'default_user_id': uid}
```

Y si la vista de destino tiene un filtro llamado **filter_my_task**, podemos habilitarlo usando:

```
{'search_default_filter_my_tasks':  True}
```

Expresiones de dominio

Los dominios se usan para filtrar los datos de registro. Odoo los analiza detenidamente para formar la expresión WHERE SQL usada para consultar a la base de datos.

Cuando se usa en una acción de ventana para abrir una vista, el dominio fija un filtro en los registros que estarán disponibles en esa vista. Por ejemplo, para limitar solo a las Tareas del usuario actual:

```
domain=[('user_id', '=', uid)]
```

El valor "uid" usado aquí es provisto por el contexto de sesión. Cuando se usa en un campo relacional, limitara las opciones disponibles de selección para ese campo. El filtro de dominio puede también usar valores de otros campos en la vista. Con esto podemos tener diferentes opciones disponibles dependiendo de lo seleccionado en otros campos. Por ejemplo, un campo de persona de contacto puede ser establecido para mostrar solo las personas de la compañía seleccionada previamente en otro campo.

Un dominio es una lista de condiciones, donde cada condición es una tupla ('field', 'operator', 'value').

El campo a la izquierda es al cual se aplicara el filtro, y puede ser usada la notación de punto en los campos relaciones.

Los operadores que pueden ser usados son:

- =, "like" para coincidencias con el valor del patrón donde el símbolo de guión bajo (_) coincida con cualquier carácter único, y % coincida con cualquier secuencia de caracteres. "like" para hacer coincidir con el patrón SQL %value% sensible a mayúsculas, e "ilike" para coincidencias sin sensibilidad de mayúsculas. Los operadores "not like" y "not ilike" hacen la operación inversa.
- child_of encuentra los hijos directos e indirectos, si las relaciones padre/hijo están configuradas en el modelo de destino.
- "in" y "not" verifican la inclusión en una lista. En este caso, el valor de la derecha debe ser una lista Python. Estos son los únicos operadores que pueden ser usados con valores de una lista. Un caso especial es cuando el lado izquierdo es un campo "a-muchos": aquí el operador "in" ejecuta una operación "contains".

Están disponibles los operadores de comparación usuales: <, >, <=, >=, =, y !=.

El valor dela derecha puede puede ser una constante o una expresión Python a ser evaluada. Lo que puede ser usado en estas expresiones depende del contexto disponible (no debe ser confundido con el contexto de sesión, discutido en la sección anterior). Existen dos posibles contextos de evaluación para los dominios: del lado del cliente y del lado del servidor.

Para los dominios de campo y las acciones de ventana, la evaluación es realizada desde el lado del cliente. El contexto de evaluación incluye aquí los campos disponibles para la vista actual, y la notación de puntos no esta disponible. Puede ser usados los valores del contexto de sesión, como "uid" y "active_id". Estan disponibles los módulo de Python "datetime" y "time" para ser usado en las operaciones de fecha y hora, y también esta disponible la función context_today() que devuelve la fecha actual del cliente.

Los dominios usados en las reglas de registro de seguridad y en el código Pyhton del servidor son evaluados del lado el servidor. El contexto de evaluación tiene los campos los registros actuales disponibles, y se permite la notación de puntos. También están disponibles los registros de la sesión de usuario actual. Al usar user.id es equivalente a usar "uid" en el contexto de evaluación del lado del cliente.

Las condiciones de dominio pueden ser combinadas usando los operadores lógicos: & para "AND" (el predeterminado), | para "OR" y ! para la negación.

La negación es usada antes de la condición que será negada. Por ejemplo, para encontrar todas las tareas que no pertenezca al usuario actual: ['!', ('user_id','=', uid)].

El "AND" y "OR" operan en las dos condiciones siguientes. Por ejemplo: para filtrar las tareas del usuario actual o sin un responsable asignado:

```
['|', ('user_id', '=', uid), ('user_id', '=', False)]
```

Un ejemplo más complejo, usado en las reglas de registro del lado del servidor:

```
['|', ('message_follower_ids', 'in', [user.partner_id.id]), '|', ('user_id', '=', user.id), ('user_id', '=', False)]
```

El dominio filtra todos los registro donde los seguidores (un campo de muchos a muchos) contienen al usuario actual además del resultado de la siguiente condición. La siguiente condición es, nuevamente, la unión de otras dos

condiciones: los registros donde el "user_id" es el usuario de la sesión actual o no esta fijado.

Vistas de Formulario

Como hemos visto en capítulos anteriores, las vistas de formulario cumplir con una diseño simple o un diseño de documento de negocio, similar a un documento en papel.

Ahora veremos como diseñar vistas de negocio y usar los elementos y widgets disponibles. Esto es hecho usualmente heredando la vista base. Pero para hacer el código más simple, crearemos una vista completamente nueva para las tareas por hacer que sobre escribirá la definida anteriormente.

De hecho, el mismo modelo puede tener diferentes vistas del mismo tipo. Cuando se abre un tipo de vista para un modelo a través de una acción, se selecciona aquella con la prioridad más baja. O como alternativa, la acción puede especificar exactamente el identificador de la vista que se usará. La acción que definimos al principio de este capítulo solo hace eso; el view_id le dice a la acción que use específicamente el formulario con el ID view_form_todo_task_ui. Esta es la vista que crearemos a continuación.

Vistas de negocio

En una aplicación de negocios podemos diferenciar los datos auxiliares de los datos principales del negocio. Por ejemplo, en nuestra aplicación los datos principales son las tareas por hacer, y las etiquetas y los estados son tablas auxiliares.

Estos modelos de negocio pueden usar diseños de vista de negocio mejorados para mejorar la experiencia del usuario y la usuaria. Si vuelve a ejecutar la vista del formulario de tarea agregada en el Capítulo 2, notará que ya sigue la estructura de vista de negocio.

La vista de formulario correspondiente debe ser agregada después de las acciones y los elementos del menú, que agregamos anteriormente, y su estructura genérica es esta:

```
<record id="view_form_todo_task_ui" mode
l="ir.ui.view">
  <field name="name">view_form_todo_tas
k_ui</field>
  <field name="model">todo.task</field>
  <field name="arch" type="xml">
    <form>
      <header><!-- Buttons and status widg
et --> </header>
      <sheet><!-- Form  content --> </sheet
>
      <!-- History and communication: -->
      <div class="oe_chatter">
        <field name="message_follower_ids
" widget="mail_followers" />
        <field name="message_ids" widget
="mail_thread" />
    </div>
    </form>
  </field>
</record>
```

Las vistas de negocio se componen de tres área visuales:

Un encabezado, "header"

Un "sheet" para el contenido

Una sección al final de historia y comunicación, "history and communication".

La sección historia y comunicación, con los widgets de red social en la parte inferior, es agregada por la herencia de nuestro modelo de mail.thread (del módulo mail), y agrega los elementos del ejemplo XML mencionado anteriormente al final de la vista de formulario. También vimos esto en el Capítulo 3.

La barra de estado del encabezado

La barra de estado en la parte superior usualmente presenta el flujo de negocio y los botones de acción.

Los botones de acción son botones regulares de formulario, y lo más común es que el siguiente paso sea resaltarlos, usandoclass="oe_highlight".
En todo_ui/todo_view.xml podemos ampliar el encabezado vacío para agregar le una barra de estado:

```
<header>
  <field name="stage_state" invisible="True
" />
  <button name="do_toggle_done" type="o
bject" attrs="{'invisible' [('stage_state','in',['
```

```
done','cancel'])]}" string="Toggle Done" clas
s="oe_highlight" />
    <!-- Add stage statusbar:  ... -->
</header>
```

Los botones de acción disponible puede diferir dependiendo en que parte del proceso se encuentre el documento actual. Por ejemplo, un botón Marcar como Hecho no tiene sentido si ya estamos en el estado "Hecho".

Esto se realiza usando el atributo "states", que lista los estados donde el botón debería estas visible, como esto: states="draft,open".

Para mayor flexibilidad podemos usar el atributo "attrs", el cual forma condiciones donde el botón debería ser invisible: attrs="{'invisible' [('stage_state','in', ['done','cancel'])].

Estas características de visibilidad también están disponibles para otros elementos de la vista, y no solo para los botones. Veremos esto en detalle más adelante en este capítulo.

El flujo de negocio

El flujo de negocio es un widget de barra de estado que se encuentra en un campo el cual representa el punto en el flujo donde se encuentra el registro. Usualmente es un campo de selección "State", o un campo "Stage" muchos a uno. En ambos casos puede encontrarse en muchos módulos de Odoo.

El "Stage" es un campo muchos a uno que se usa en un modelo donde los pasos del proceso están definidos. Debido a esto pueden ser fácilmente configurados por el usuario u la usuaria final para adecuarlo a sus procesos específicos de negocio, y son perfectos para el uso de pizarras kanban.

El "State" es una lista de selección que muestra los pasos estables y principales de un proceso, como Nuevo, En Progreso, o Hecho. No pueden ser configurados por el usuario o usuaria final, pero son fáciles de usar en la lógica de negocio. Los "States" también tienen soporte especial para las vistas: el atributo "state" permite que un elemento este habilitado para ser seleccionado por el usuario o usuaria dependiendo en el estado en que se encuentre el registro.

Para agregar un flujo de "stage" en nuestro encabezado de formulario:

```
<!--  Add stage  statusbar:  ... -->
<field name="stage_id" widget="statusbar"
clickable="True" options="{'fold_field': 'fol
d'}" />
```

El atributo "clickable" permite hacer clic en el widget, para cambiar la etapa o el estado del documento. Es posible que no queramos esto si el progreso del proceso debe realizarse a través de botones de acción.

En el atributo "options" podemos usar algunas configuraciones específicas:

fold_fields, cuando de usa "stages", es el nombre del campo que usa el "stage" del modelo usa para indicar en cuales etapas debe ser mostrado "fold".

statusbar_visible, cuando se usa "states", lista los estados que deben estar siempre visibles, para mantener ocultos los estados de excepción que se usan para casos menos comunes. Por ejemplo: statusbar_visible="draft,open.done".

La hoja canvas es el área del formulario que contiene los elementos principales del formulario. Esta diseñada para parecer un documento de papel, y sus registros de datos, a veces, puede ser referidos como documentos.

La estructura general del documento tiene estos componentes:

- Información de título y subtítulo
- Un área de botón inteligente, es la parte superior derecha de los campos del encabezado del documento.

Un cuaderno con páginas en etiquetas, con líneas de documento y otros detalles.

Título y subtítulo

Cuando se usa el diseño de hoja, los campos que están fuera del bloque <group> no se mostrarán las etiquetas automáticamente. Es responsabilidad de la persona que desarrolla controlar si se muestran las etiquetas y cuando.

También se puede usar las etiquetas HTML para hacer que el título resplandezca. Para mejores resultados, el título del documento debe estar dentro de un "div" con la clase oe_title:

```
<div class="oe_title">
```

```
    <label for="name" class="oe_edit_only"/>
    <h1><field name="name"/></h1>
    <h3>
        <span class="oe_read_only">By</span>
        <label for="user_id" class="oe_edit_only"/>
        <field name="user_id" class="oe_inline" />
    </h3>
</div>
```

Aquí podemos ver el uso de elementos comúnes de HTML como div, span, h1 y h3.

Etiquetas y campos

Las etiquetas de los campos no son mostradas fuera de las secciones **<group>**, pero podemos mostrarlas usando el elemento **<label>**:

El atributo "for" identifica el campo desde el cual tomaremos el texto de la etiqueta.

El atributo "string" sobre escribe el texto original de la etiqueta del campo.

Con el atributo "class" también podemos usar las clases CSS para controlar la presentación. Algunas clases útiles son:

oe_edit_only para mostrar lo solo cuando el formulario este modo de edición.

oe_read_only para mostrar lo solo cuando el formulario este en modo de lectura.

Un ejemplo interesante es reemplazar el texto con un ícono:

```
<label for="name" string=" " class="fafa-wrench"/>
```

Odoo empaqueta los íconos "Font Awesome", que se usan aquí. Los íconos disponibles puede encontrar se en http://fontawesome.org.

Botones inteligentes

El área superior izquierda puede tener una caja invisibles para colocar botones inteligentes. Estos funcionan como los botones regulares pero pueden incluir información estadística. Como ejemplo agregaremos un botón para mostrar el número total de tareas realizadas por el dueño de la tarea por hacer actual.

Primero necesitamos agregar el campo calculado correspondiente a todo_ui/todo_model.py. Agregue lo siguiente a la clase TodoTask:

```
@api.one def compute_user_todo_count(self):
    self.user_todo_count = self.search_count([('user_id', '=', self.user_id.id)])
    user_todo_count = fields.Integer('User To-Do Count', compute='compute_user_todo_count')
```

Ahora agregaremos la caja del botón con un botón dentro de ella. Agregue lo siguiente justo después del bloque div oe_title:

```
<div name="buttons" class="oe_right oe_button_box">
    <button class="oe_stat_button" type="action" icon="fa-tasks" name="%(todo_app.action_todo_task)d" string="" context="{'search_default_user_id': user_id, 'default_user_id': user_id}" help="Other to-dos for this user">
        <field string="To-dos" name="user_todo_count" widget="statinfo"/>
    </button>
</div>
```

El contenedor para los botones es un div con las clases oe_button_box y oe_right, para que este alineado con la parte derecha del formulario.

En el ejemplo el botón muestra el número total de las tareas por hacer que posee el documento responsable. Al hacer clic en el, este las inspeccionara, y si se esta creando tareas nuevas el documento responsable original será usado como predeterminado.

Los atributos usados para el botón son:

- class="oe_stat_button", es para usar un estilo rectángulo en vez de un botón.
- icon, es el ícono que será usaso, escogido desde el conjunto de íconos de Font Awesome.
- type, será usualmente una acción para la acción de ventana, y name será el ID de la acción que será ejecutada. Puede usarse la formula %(id-acción-externa)d, para transformar el ID externo en un número de ID real. Se espera que esta acción abra una vista con los registros relacionados.

- string, puede ser usado para agregar texto al botón. No se usa aquí porque el campo que lo contiene ya proporciona un texto.
- context, fija las condiciones estándar en la vista destino, cuando se haga clic a través del botón, para los filtros de datos y los valores predeterminados para los registros creados.
- help, es la herramienta de ayuda que será mostrada.

Por si solo el botón es un contenedor y puede tener sus campos dentro para mostrar estadísticas. Estos son campos regulares que usan el widget "statinfo".

El campo debe ser un campo calculado, definido en el módulo subyacente. También podemos usar texto estático en vez de o junto a los campos de "statinfo", como : User's To-dos

Organizar el contenido en formulario

El contenido principal del formulario debe ser organizado usando etiquetas <group>. Un grupo es una cuadrícula con dos columnas. Un campo y su etiqueta ocupan dos columnas, por lo tanto al agregar campos dentro de un grupo, estos serán apilados verticalmente.

Si anidamos dos elementos <group> dentro de un grupo superior, tendremos dos columnas de campos con etiquetas, una al lado de la otra.

```
<group name="group_top">
   <group name="group_left">
      <field name="date_deadline" />
      <separator string="Reference"/>
      <field name="refers_to"/>
   </group>
   <group name="group_right">
      <field name="tag_ids" widget="many2
many_tags"/>
   </group>
</group>
```

Los grupos pueden tener un atributo "string", usado para el título de la sección. Dentro de una sección de grupo, los títulos también pueden agregarse usando un elemento "separator".

Cuaderno con pestañas

Otra forma de organizar el contenido es el cuaderno, el cual contiene múltiples secciones a través de pestañas llamadas páginas. Esto puede usarse para mantener algunos datos fuera de la vista hasta que sean necesarios u organizar un largo número de campos por tema.

No necesitaremos esto en nuestro formulario de tareas por hacer, pero el siguiente es un ejemplo que podríamos agregar en el formularios de etapas de la tarea:

```
<notebook>
   <page string="Whiteboard" name="white
board">
      <field name="docs"/>
   </page>
   <page name="second_page">
      <!-- Second page content -->
   </page>
</notebook>
```

Se considera una buena practica tener nombres en las páginas, esto hace que la ampliación de estas por parte de otros módulo sea más fiable

Elementos de la vista

Hemos visto como organizar el contenido dentro de un formulario, usando elementos como encabezado, grupo y cuaderno. Ahora, podemos ahondar en los elementos de campo y botón y que podemos hacer con ellos.

Botones

- Los botones soportar los siguientes atributos:
- icon. A diferencia de los botones inteligentes, los íconos disponibles para los botones regulares son aquellos que se encuentran en addons/web/static/src/img/icons.
- string, es el texto de descripción del botón.
- type, puede ser "workflow", "object" o "action", para activar una señal de flujo de trabajo, llamar a un método Python o ejecutar una acción de ventana.
- name, es el desencadenante de un flujo de trabajo, un método del modelo, o la

ejecución de una acción de ventana, dependiendo del "type" del botón.

- args, se usa para pasar parámetros adicionales al método, si el "type" es "object".
- context, fija los valores en el contexto de la sesión, el cual puede tenet efecto luego de la ejecución de la acción de ventana, o al llamar a un método de Python. En el último caso, a veces puede ser usado como un alternativa a "args".
- confirm, agrega un mensaje con el mensaje de texto preguntando por una confirmación.
- special="cancel", se usa en los asistentes, para cancelar o cerrar el formulario. No debe ser usado con "type".

Campos

El campo tiene los siguientes atributos disponibles. La mayoría es tomado de los que fue definido en el modelo, pero pueden ser sobre escritos en la vista. Los atributos generales son:

- name: identifica el nombre técnico del campo.
- string: proporciona la descripción de texto de la etiqueta para sobre escribir aquella provista por el modelo.
- help: texto de ayuda a ser usado y que reemplaza el proporcionado por el modelo.
- placeholder: proporciona un texto de sugerencia que será mostrado dentro del campo.
- widget: sobre escribe el widget predeterminado usado por el tipo de campo. Exploraremos los widgets disponibles mas adelante en este mismo capítulo.
- options: contiene opciones adicionales para ser usadas por el widget.
- class: proporciona las clases CSS usadas por el HTML del campo.
- **invisible="1"**: invisibiliza el campo.
- **nolabel="1"**: no muestra la etiqueta del campo, solo es significativo para los campos que se encuentran dentro de un elemento **<group>**.
- **readonly="1"**: no permite que el campo sea editado.
- required="1": hace que el campo sea obligatorio.

Atributos específicos para los tipos de campos:

- sum, avg: para los campos numéricos, y en las vistas de lista/árbol, estos agregan un resumen al final con el total o el promedio de los valores.
- password="True": para los campos de texto, muestran el campo como una campo de contraseña.
- filename: para campos binarios, es el campo para el nombre del archivo.
- mode="tree": para campos One2many, es el tipo de vista usado para mostrar los registros. De forma predeterminada es de árbol, pero también puede ser de formulario, kanban o gráfico.

Para los atributos Boolean en general, podemos usar True o 1 para habilitarlo y False o 0 (cero) para deshabilitarlo. Por ejemplo, readonly="1" y realonly="True" son equivalentes.

Campos relacionales

En los campos relacionales, podemos tener controles adicionales referentes a los que el usuario o la usuaria puede hacer. De forma predeterminada el usuario y la usuaria pueden crear nuevos registros desde estos campos (también conocido como creación rápida) y abrir el formulario relacionado al registro. Esto puede ser deshabilitado usando el atributo del campo "options":

```
options={'no_open': True, 'no_create': True}
```

El contexto y el dominio también son particulares en los campos relacionales. El contexto puede definir valores predeterminados para los registros relacionados, y el dominio puede limitar los registros que pueden ser seleccionados, por ejemplo, basado en otro campo del registro actual. Tanto el contexto como el dominio pueden ser definidos en el modelo, pero solo son usados en la vista.

Widgets de campo

Cada tipo de campo es mostrado en el formulario con el widget predeterminado apropiado. Pero otros widget adicionales están disponible y pueden ser usados:

Widgets para los campos de texto:

- email: convierte al texto del correo electrónico en un elemento "mail-to" ejecutable.
- url: convierte al texto en un URL al que se puede hacer clic.
- html: espera un contenido en HTML y lo representa; en modo de edición usa un editor WYSIWYG para dar formato al contenido sin saber HTML.

Widgets para campos numéricos:

- handle: específicamente diseñado para campos de secuencia, este muestra una guía para dibujar líneas en una vista de lista y re ordenarlos manualmente.
- float_time: da formato a un valor decimal como tiempo en horas y minutos.
- monetary: muestra un campo decimal como un monto en monedas. La moneda a usar puede ser tomada desde un campo como options="{'currency_field': 'currency_id'}".
- progressbar: presenta un decimal como una barra de progreso en porcentaje, usualmente se usa en un campo calculado que computa una tasa de culminación.

Algunos widget para los campos relacionales y de selección:

- many2many_tags: muestran un campo muchos a muchos como una lista de etiquetas.
- selection: usa el widget del campo Selección para un campo mucho a uno.
- radio: permite seleccionar un valor para una opción del campo de selección usando botones de selección (radio buttons).
- kanban_state_selection: muestra una luz de semáforo para la lista de selección de esta kanban.
- priority: representa una selección como una lista de estrellas a las que se puede hacer clic.

Eventos on-change

A veces necesitamos que el valor de un campo sea calculado automáticamente cuando cambia otro campo. El mecanismo para esto se llama on-change.

Desde la versión o, los eventos on-change están definidos en la capa del modelo, sin necesidad de ningún marcado especial en las vistas. Es se hace creando los métodos para realizar el calculo y enlazándolos al campo(s) que desencadenara la acción, usando el decorador @api.onchenge('field1','field2').

En las versiones anteriores, ente enlace era hecho en la capa de vista, usando el atributo "onchange" para fijar el método de la clase que sería llamado cuando el campo cambiara. Esto todavía es soportado, pero es obsoleto. Tenga en cuenta que los métodos on-change con el estilo viejo no pueden ser ampliados usando la API nueva. Si necesita hacer esto, deberá usar la API vieja.

Vistas dinámicas

- Los elementos visibles como un formulario también pueden ser cambiados dinámicamente, dependiendo, por ejemplo de los permisos de usuario o la etapa del proceso en la cual esta el documento.
- Estos dos atributos nos permiten controlar la visibilidad de los elemento en la interfaz:
- groups: hacen al elemento visible solo para los miembros de los grupos de seguridad específicos. Se espera una lista separada por coma de los ID XML del grupo.
- states: hace al elemento visible solo cuando el documento esta en el estado especificado. Espera una lista separada por coma de los códigos de "State", y el modelo del documento debe tener un campo "state".
- Para mayor flexibilidad, podemos fijar la visibilidad de un elemento usando expresiones evaluadas del lado del cliente. Esto puede hacerse usando el atributoo "attrs" con un diccionario que

mapea el atributo "invisible" al resultado de una expresión de dominio.

- Por ejemplo, para hacer que el campo refers_to sea visible en todos los estados menos "draft":

```
<field name="refers_to" attrs="{'invisible': [('state','=','draft')]}" />
```

El atributo "invisible" esta disponible para cualquier elemento, no solo para los campos. Podemos usarlo en las páginas de un cuaderno o en grupos, por ejemplo.

El "attrs" también puede fijar valores para otros dos atributos: readonly y required, pero esto solo tiene sentido para los campos de datos, convirtiéndolos en campos que no pueden ser editados u obligatorios. Con esto podemos agregar alguna lógica de negocio haciendo a un campo obligatorio, dependiendo del valor de otro campo, o desde un cierto estado mas adelante.

Vistas de lista

Comparadas con las vistas de formulario, las vistas de listas son mucho más simples. Una vista de lista puede contener campos y botones, y muchos de los atributos de los formularios también están disponibles.

Aquí se muestra un ejemplo de una vista de lista para nuestra Tareas por Hacer:

```
<record id="todo_app.view_tree_todo_task" model="ir.ui.view">
   <field name="name">To-do Task Tree</field>
   <field name="model">todo.task</field>
   <field name="arch" type="xml">
      <tree editable="bottom" colors="gray:is_done==True" fonts="italic: state!='open'" delete="false">
         <field name="name"/>
         <field name="user_id"/>
      </tree>
   </field>
</record>
```

Los atributos para el elemento "tree" de nivel superior son:

- editable: permite que los registros sean editados directamente en la vista de lista. Los valores posibles son "top" y "bottom", los lugares en donde serán agregados los registros nuevos.
- colors: fija dinámicamente el color del texto para los registros, basándose en su contenido. Es una lista separada por punto y coma de valores **color:condition**. "color" es un color válido CSS (vea **http://www.w3.org/TR/css3-color/#html4**), y "condition" es una expresión Python que evalúa el contexto del registro actual.
- fonts: modifica dinámicamente el tipo de letra para los registros basándose en su contexto. Es similar al atributo "colors", pero este fija el estilo de la letra a "bold", "italic" o "underline".
- create, delete, edit: si se fija a "false" (en minúscula), deshabilita la acción correspondiente en la vista de lista.

Vistas de búsqueda

Las opciones de búsqueda disponibles en las vistas son definidas a través de una vista de lista. Esta define los campos que serán buscados cuando se escriba en la caja de búsqueda. También provee filtros predefinidos que pueden ser activados con un clic, y opciones de agrupación de datos para los registros en las vistas de lista o kanban.

Aquí se muestra una vista de búsqueda para las tareas por hacer:

```
<record id="todo_app.view_filter_todo_task" model="ir.ui.view">
   <field name="name">To-do Task Filter</field>
   <field name="model">todo.task</field>
   <field name="arch" type="xml">
      <search>
         <field name="name" domain_filter="['|', ('name','ilike',self),('user_id','ilike',self)]" />
         <field name="user_id"/>
         <filter name="filter_not_done" string="Not Done" domain="[('is_done','=',False)]"/>
         <filter name="filter_done" string="Done" domain="[('is_done','!=',False)]"/>
         <separator/>
         <filter name="group_user" string="By User" context="{'group_by':'user_id'}"/>
      </search>
```

```
    </field>
</record>
```

Podemos ver dos campos que serán buscados: "name" y "user_id". En "name" tenemos una regla de filtro que hace la "búsqueda si" tanto en la descripción como en el usuario responsable. Luego tenemos dos filtros predefinidos, filtrando las "tareas no culminadas" y "tareas culminadas". Estos filtros pueden ser activados de forma independiente, y serán unidos por un operador "OR" si ambos son habilitados. Los bloques de "filters" separados por un elemento <separator/> serán unidos por un operador "AND".

El tercer filtro solo fija un contexto o "group-by". Esto le dice a la vista que agrupe los registros por ese campo, user_id en este caso.

- Los elementos "filed" pueden usar los siguientes atributos:
- name: identifica el campo.
- string: proporciona el texto de la etiqueta que será usado, en vez del predeterminado.
- operator: nos permite usar un operador diferente en vez del predeterminado - = para campos numéricos y "ilike" para otros tipos de campos.
- filter_domain: puede usarse para definir una expresión de dominio específica para usar en la búsqueda, proporcionando mayor flexibilidad que el atributo "operator". El texto que será buscado se referencia en la expresión usando "self".

- groups: permite hacer que la búsqueda en el campo solo este disponible para una lista de grupos de seguridad (identificado por los Ids XML)

Estos son los atributos disponibles para los elementos "filter":

- name: en un identificador, usado para la herencia o para habilitar la a través de la clave **search_default_** en el contexto de acciones de ventana.
- string: proporciona el texto de la etiqueta que se mostrara para el filtro (obligatorio)
- domain: proporciona la expresión de dominio del filtro para ser añadida al dominio activo.
- context: es un diccionario de contexto para agregarlo al contexto actual. Usualmente este fija una clave **group_by** con el nombre del filtro que agrupara los registros.
- groups: permite hacer que el filtro de búsqueda solo este disponible para una lista de grupos.

Otros tipos de vista

Los tipos de vista que se usan con mayor frecuencia son los formularios y las listas, discutidos hasta ahora. A parte de estas, existen otros tipos de vista, y daremos un vistazo a cada una de ellas. Las vistas kanban no serán discutidas aquí, ya que las veremos en el Capítulo 8.

Recuerde que los tipos de vista disponibles están definidos en el atributo **view_mode** de la acción de ventana correspondiente.

Vistas de Calendario

Como su nombre lo indica, esta presenta los registros en un calendario. Una vista de calendario para las tareas por hacer puede ser de la siguiente manera:

```
<record id="view_calendar_todo_task" mod
el="ir.ui.view">
   <field name="name">view_calendar_todo
_task</field>
   <field name="model">todo.task</field>
   <field name="arch" type="xml">
      <calendar date_start="date_deadline" c
olor="user_id" display="[name], Stage[stage
_id]">
         <!-- Fields used for the text of display
attribute -->
         <field name="name" />
         <field name="stage_id"  />
      </calendar>
   </field>
</record>
```

Los atributos de "calendar" son los siguientes:

- **date_start**: El campo para la fecha de inicio (obligatorio).
- **date_end**: El campo para la fecha de culminación (opcional).

- **date_delay**: El campo para la duración en días. Este puede ser usado en vez de **date_end**.
- color: El campo para colorear las entradas del calendario. Se le asignará un color a cada valor en el calendario, y todas sus entradas tendrán el mismo color.
- display: Este es el texto que se mostrará en las entradas del calendario. Los campos pueden ser insertados usando **[<field>]**. Estos campos deben ser declarados dentro del elemento "calendar".

Vistas de Gantt

Esta vista presenta los datos en un gráfico de Gantt, que es útil para la planificación. La tarea por hacer solo tiene un campo de fecha para la fecha de límite, pero podemos usarla para tener una vista funcional de un gráfico Gantt básico:

```
<record id="view_gantt_todo_task" model="
ir.ui.view">
   <field name="name">view_gantt_todo_tas
k</field>
   <field name="model">todo.task</field>
   <field name="arch" type="xml">
      <gantt date_start="date_deadline" defa
ult_group_by="user_id" />
   </field>
</record>
```

Los atributos que puede ser usados para las vistas Gantt son los siguientes.

- **date_start**: El campo para la fecha de incio (obligatorio).
- **date_stop**: El campo para la fecha de culminación. Puede ser reemplazado por **date_delay**.
- **date_delay**: El campo con la duración en días. Puede usarse en vez de **date_stop**.
- progress: Este campo proporciona el progreso en porcentaje (entre 0 y 100).
- **default_group_by**: Este campo se usa para agrupar las tareas Gantt.

Vistas de Gráfico

Los tipos de vista de gráfico proporcionan un análisis de los datos, en forma de gráfico o una tabla pivote interactiva.

Agregaremos una tabla pivote a las tareas por hacer. Primero, necesitamos agregar un campo. En la clase TodoTask, del archivo todo_ui/todo_model.py, agregue este línea:

```
effort_estimate = fields.Integer('Effort Estim
ate')
```

También debe ser agregado al formulario de tareas por hacer para que podamos fijar datos allí. Ahora, agreguemos la vista de gráfico con una tabla pivote:

```
<record id="view_graph_todo_task" model=
"ir.ui.view">
   <field name="name">view_graph_todo_ta
sk</field>
   <field name="model">todo.task</field>
   <field name="arch" type="xml">
      <graph type="pivot">
         <field name="stage" type="col" />
         <field name="user_id"  />
         <field name="date_deadline" interval
="week" />
         <field name="effort_estimate" type="
measure" />
      </graph>
   </field>
</record>
```

El elemento "graph" tiene el atributo "type" fijado a "pivot". También puede ser "bar" (predeterminado), "pie" o "line". En el caso que sea "bar", gráfico de barras, adicionalmente se puede usar stacked="True" para hacer un gráfico de barras apilado.

"graph" debería contener campos que pueden tener estos posibles atributos:

name: Identifica el campo que será usado en el gráfico, así como en otras vistas.

type: Describe como será usado el campo, como un grupo de filas (predeterminado), "row", como un grupo de columnas, "col", o como una medida, "mesure".

interval: Solo es significativo para los campos de fecha, es un intervalo de tiempo para agrupar datos de fecha por "day", "week", "month", "quarter" o "year".

QweB - Creando vistas Kanban y Reportes

QWeb es un motor (engine) de plantillas por Odoo. Está basado en XML y es utilizado para generar fragmentos y páginas html. QWeb fue introducido por primera vez en la versión 7.0 para habilitar vistas Kanban más ricas, y con las versión 8.0, también se usa para la generación de reportes y páginas web CMS (CMS: Sistemas Manejadores de Contenido).

Aquí aprenderás acerca de la sintaxis QWeb y como usarla para crear tus propias vistas Kanban reportes personalizados.

Para entender los tableros Kanban, Kanban es una palabra de origen japonés que es usada para representar un método de gestión de colas (queue) de trabajo. Fue inspirado del Sistema de Producción y Fabricación Ligera (lean) de Toyota, y se ha vuelto popular en la la industria del software con su adopción en las metodologías Ágiles.

Las vistas Kanban una característica distintiva de Odoo, haciendo fácil implementar estos tableros. Aprendamos cómo usarlos.

Vistas Kanban

En las vistas de formulario, usamos mayormente elementos XML específicos, tales como <field> y <group>, y algunos elementos HTML, tales como <h1> o . Con las vistas Kanban, es un poco lo opuesto; ellas son plantillas basadas en HTML y soportan solo dos elementos específicos de Odoo, <field> y <button>.

El HTML puede ser generado dinámicamente usando el motor de plantilla Qweb. Éste procesa los atributos de etiqueta especiales en los elementos HTML para producir el HTML final para ser presentado por el cliente web. Esto proporciona mucho control sobre cómo renderizar el contenido, pero también permite hacer diseños de vistas más complejas.

El tablero Kanban es una herramienta para visualizar la cola de trabajo. Los artículos (items) de trabajo están representados por tarjetas que son organizadas en columnas represantando las etapas (stages) del proceso de trabajo. Nuevos artículos de trabajo inician en la columna más a la izquierda y viaja a través del tablero hasta que alcanzan la columna más a la derecha, representando el trabajo completado.

Iniciándose con el tablero Kanban

La simplicidad y el impacto visual del tablero Kanban los hace excelente para soportar procesos de negocio simples. Un ejemplo básico de un tablero Kanban puede tener tres columnas, como se muestra en la siguiente imagen: "ToDo", "Doing" y "Done" (Por hacer, haciendo y hecho), pero, por supuesto puede ser extendido a cualquier paso de un proceso específico que necesitemos:

Gráfico 8.1 - tablero Kanban

Las vistas Kanban son tan flexibles que pueden haber muchas formas diferentes de diseñarlas, y puede ser difícil proveer una receta para seguir. Una buena regla general es encontrar un vista Kanban existente similar a lo que queremos alcanzar, y crear nuestro nuevo trabajo de vista Kanban basada en ella.

Observando las vistas Kanban usadas en los módulos estándar, es posible identificar dos estilos de vistas Kanban principales: viñeta y tarjeta

Ejemplos de las vistas Kanban de estilo viñeta pueden ser encontrados en Clientes, Productos, y también, Aplicaciones y Módulos. Ellos usualmente no tienen borde y son decorados con imágenes en el lado de la izquierda, tal como se muestra en la siguiente imagen:

Gráfico 8.2 - Ejemplo de vistas Kanban tipo Viñeta

El estilo **tarjeta** Kanban es usualmente usada para mostrar tarjetas organizadas en columnas para las etapas de procesos. Ejemplo de esto son las **Oportunidades CRM y las Tareas de Proyectos**. El contenido principal es mostrado en el área superior de la tarjeta y la información adicional puede ser mostrada en las áreas inferior derecha e inferior izquierda, tal como se muestra en la siguiente imagen:

Gráfico 8.3 - Ejemplo de estilo de tarjeta Kanban

Veremos el esqueleto y elementos típicos usados en ambos estilos de vistas tal que puedas sentirte cómodo adaptándolos a tus casos de usos particular.

Diseña vistas Kanban

La primera cosa es crear un nuevo módulo agregando nuestras vistas Kanban a la lista de tareas por hacer. En un trabajo del mundo real, una situación de uso de un módulo para esto podría ser, probablemente, excesiva y ellas podrían ser perfectamente agregadas directamente en el módulo todo_ui. Pero para una explicación más clara, usaremos un nuevo módulo y evitaremos demasiados, y posiblemente confusos, cambios en archivos ya creados. Lo nombraremos todo_kanban y crearemos los archivos iniciales tal como sigue:

```
$ cd ~/odoo-dev/custom-addons
$ mkdir todo_kanban
$ touch todo_kanban/__init__.py
```

Ahora, edita el archivo descriptor todo_kanban/**opernerp**.py tal como sigue:

```
{
 'name': 'To-Do Kanban',
 'description': 'Kanban board for to-do tasks
.',
 'author': 'Daniel Reis',
 'depends': ['todo_ui'],
 'data': ['todo_view.xml']
}
```

Next, create the XML file where our shiny new Kanban views will go and set Kanban as the default view on the to-do task

Lo que sigue es crear el archivo XML donde nuestras nuevas y brillantes vistas Kanban irán y establecer Kanban como la vista por defecto en la acción (action) de ventana de las tareas por hacer (to-do tasks), tal como se muestre a continuación:

```
<?xml version="1.0"?>
<openerp>
  <data>
    <!-- Agrega el modo de vista kanban al
menu Action: -->
  <act_window id="todo_app.action_todo_ta
sk" name="To-Do Tasks" res_model="todo.
task" view_mode="kanban,tree,form,calend
ar,gantt,graph" context="{'search_default_fi
lter_my_tasks':True}" />
    <!-- Agregar vista kanban -->
      <record id="To-do Task Kanban" mod
el="ir.ui.view">
        <field name="name">To-do Task Ka
nban</field>
        <field name="model">todo.task</fiel
d>
        <field name="arch" type="xml">
          <!-- vacío por ahora, pero el Kanban
irá aquí! -->
        </field>
      </record></data>
</openerp>
```

Ahora tenemos ubicado el esqueleto básico para nuestro módulo. Las plantillas usada en las vistas kanban y los reportes son extendidos usando las técnicas regulares usadas para otras vistas, por ejemplos usando expresiones XPATH., Herencia – Extendiendo Aplicaciones Existentes.

Antes de iniciar con las vistas kanban, necesitamos agregar un para de campos en el modelo tareas por hacer. (to-do tasks model)

Prioridad y estado Kanban

Los dos campos que son frecuentemente usados en las vistas kanban son: priority y kanban state. **Priority** permite a los usuarios organizar sus elementos de trabajo, señalando lo que debería estar ubicado primero. **Kanban state** señala cuando una tarea está lista para pasar a la siguiente etapa o si es bloqueada por alguna razón. Ambos son soportados por campos selection y tienen widgets específicos para ser usados en las vistas de formulario y kanban.

Para agrega estos campos a nuestro modelo, agregaremos al archivo todo_kanban/todo_task.py, tal como se muestra a continuación:

```
from openerp import models, fields
   class TodoTask(models.Model):
      _inherit = 'todo.task'
      priority = fields.Selection([
                           ('0','Low'),
                           ('1','Normal'),
                           ('2','High')],
                           'Priority',default='1')
      kanban_state = fields.Selection([
                              ('normal', 'In Progres
s'),
                              ('blocked', 'Blocked'),
                              ('done', 'Ready for ne
xt stage')],
                              'Kanban State', defau
lt='normal')
```

No olvidemos el archivo todo_kanban/**init**.py que cargará el código precedente:

```
from . import todo model
```

Elementos de la vista kanban

La arquitectura de la vista kanban tiene un elemento superior y la siguiente estructura básica:

```
<kanban>
  <!-- Fields to use in expressions... -->
  <field name="a_field" />
  <templates>
    <t t-name="kanban-box">
      <!-- HTML Qweb template ... -->
    </t>
  </templates>
</kanban>
```

El elemento contiene las plantillas para los fragmentos HTML a usar —uno o más. La plantilla principal a ser usada debe ser nombrada kanban-box. Otras plantillas son permitidas para fragmentos HTML para se incluido en la plantilla principal.

Las plantillas usan html estándar, pero pueden incluir etiquetas <field> para insertar campos del modelo. También pueden ser usadas algunas directivas especiales de Qweb para la generación dinámica de contenido, tal como el t-name usado en el ejemplo previo.

Todos los campos del modelo usados deben ser declarados con una etiqueta <field>. Si ellos son usados solo en expresiones, tenemos que declararlos antes de la sección <templates>. Uno de esos campos se le permite tener un valor agregado, mostrado en en el área superior de las columnas kanban. Esto se logra mediante la adición de un atributo con la agregación a usar, por ejemplo:

```
<field name="effort_estimated" sum="Total
Effort" />
```

Aquí, la suma para el campo de estimación de esfuerzo es presentada en el área superior de las columnas kanban con la etiqueta Total Effort. Las agregaciones soportadas son sum, avg, min, max y count.

El elemento superior también soporta algunos atributos interesantes:

- default_group_by: Establece el campo a usar para la agrupación por defecto de columnas
- default_order: Establece un orden por defecto para usarse en los elementos kanban

- quick_create="false": Deshabilita la opción de creación rápida en la vista kanban
- class: Añade una clase CSS al elemento raíz en la vista kanban renderizada.

Ahora démosle una mirada más de cerca a las plantillas Qweb usadas en las vistas kanban.

La vista kanban viñeta

Para las plantilas QWeb de las viñetas kanban, el esqueleto se ve así:

```
<t t-name="kanban-box"/>
  <div class="oe_kanban_vignette">
    <!-- Left side image: -->
    <img  class="oe_kanban_image"  name="..."
>
      <div class="oe_kanban_details">
        <!-- Title and data -->
        <h4>Title</h4>
        <br>Other data <br/>
        <ul>
          <li>More data</li>
        </ul>
      </div>
  </div>
</t>
```

Puedes ver las dos clases CSS principales provistas para los kanban de estilo viñeta: oe_kanban_vignette para el contenedor superior y oe_kanban_details para el contenido de datos.

La vista completa de viñeta kanban para las tareas por hacer es como sigue:

```
<kanban>
  <templates>
    <t t-name="kanban-box">
      <div class="oe_kanban_vignette">
        <img t-att-src="kanban_image('res.p
artner',
                        'image_medium',
                        record.id.value)"
          class="oe_kanban_image"/>
        <div class="oe_kanban_details">
          <!-- Title and Data content -->
          <h4><a type="open">
            <field name="name"/> </a></h
4>
            <field name="tags" />
              <ul>
                <li><field  name="user_id" /
></li>
                <li><field  name="date_dead
line"/></li>
              </ul>
            <field name="kanban_state" w
idget="kanban_state_selection"/>
            <field  name="priority"  widget
="priority"/>
          </div>
        </div>
      </t>
    </templates>
</kanban>
```

Podemos ver los elementos discutidos hasta ahora, y también algunos nuevos. En la etiqueta , tenemos el atributo QWeb especial t-att-src. Esto puede calcular el contenido src de la imagen desde un campo almacenado en la base de datos. Explicaremos esto en otras directivas QWeb en un momento. También podemos ver el uso del atributo especial type en la etiqueta <a>. Echémosle un vistazo más de cerca.

Acciones en las vistas Kanban

En las plantillas Qweb, la etiqueta para enlaces puede tener un atributo type. Este establece el tipo de acción que el enlace ejecutará para que los enlaces puedan actuar como los botones en los formularios regulares. En adición a los elementos <button>, las etiquetas <a> también pueden ser usadas para ejecutar acciones Odoo.

Así como en las vistas de formulario, el tipo de acción puede ser acción u objeto, y debería ser acompañado por atributo nombre, que identifique la acción específica a ejecutar. Adicionalmente, los siguientes tipos de acción también están disponibles:

- open: Abre la vista formulario correspondiente
- edit: Abre la vista formulario correspondiente directamente en el modo de edición
- delete: Elimina el registro y remueve el elemento de la vista kanban.

La vista kanban de tarjeta El kanban de **tarjeta** puede ser un poco más complejo. Este tiene un área de contenido principal y dos sub-contenedores al pie, alineados a cada lado de la tarjeta. También podría contener un boton de apertura de una acción de menú en la esquina superior derecha de la tarjeta.

El esqueleto para esta plantilla se vería así:

```
<t t-name="kanban-box">
    <div class="oe_kanban_card">
        <div class="oe_dropdown_kanban oe_d
ropdown_toggle">
        <!-- Top-right drop down menu -->
        </div>
        <div class="oe_kanban_content">
            <!-- Content fields go here... -->
            <div class="oe_kanban_bottom_right
"></div>
            <div class="oe_kanban_footer_left">
</div>
        </div>
    </div>
</t>
```

Un kanban **tarjeta** es más apropiada para las tareas to-do, así que en lugar de la vista descrita en la sección anterior, mejor deberíamos usar la siguiente:

```
<t t-name="kanban-box">
    <div class="oe_kanban_card">
        <div class="oe_kanban_content">
            <!-- Option menu will go here! -->
            <h4><a type="open">
                <field name="name" />
                </a></h4>
                <field name="tags" />
                <ul>
                    <li><field name="user_id" /></li
>
                    <li><field name="date_deadline"
/></li>
                </ul>
                <div class="oe_kanban_bottom_rig
ht">
                    <field name="kanban_state" wid
get="kanban_state_selection"/>
                </div>
                <div class="oe_kanban_footer_left"
>
                    <field name="priority" widget="
priority"/>
                </div>
        </div>
    </div>
</t>
```

Hasta ahora hemos visto vistas kanban estáticas, usando una combinación de HTML y etiquetas especiales (field, button, a). Pero podemos tener resultados mucho más interesantes usando contenido HTML generado dinámicamente. Veamos como podemos hacer eso usando Qweb.

Agrengano contenido dinámico Qweb

El analizador Qweb busca atributos especiales (directivas) en las plantillas y las reemplaza con HTML generado dinámicamente.

Para las vistas kanban, el análisis se realiza mediante Javascript del lado del cliente. Esto significa que las evaluaciones de expresiones hechos por Qweb deberían ser escritas usando la sintaxis Javascript, no Python.

Al momento de mostrar una vista kanban, los pasos internos son aproximadamente los siguientes:

- Obtiene el XML de la plantilla a renderizar
- Llama al método de servidor read() para obtener la data de los campos en las plantillas.
- Ubica la plantilla kanban-box y la analiza usando Qweb para la salida de los fragmentos HTML finales.
- Inyecta el HTML en la visualización del navegador (el DOM).

Esto no significa que sea exacto técnicamente. Es solo un mapa mental que puede ser útil para entender como funcionan las cosas en las vistas kanban.

A continuación exploraremos las distintas directivas Qweb disponibles, usando ejemplos que mejorarán nuestra tarjeta kanban de la tarea to-do.

Renderizado Condicional con t-if

- La directiva t-if, usada en el ejemplo anterior, acepta expresiones JavaScript para ser evaluadas. La etiqueta y su contenido serán renderizadas si la condición se evalúa verdadera.
- Por ejemplo, en la tarjeta kanban, para mostrar el esfuerzo estimado de la Tarea, solo si este contiene un valor, después del campo date_deadline, agrega lo siguiente:

```
<t t-if="record.effort_estimate.raw_value > 0">
    <li>Estimate <field name="effort_estimate"/></li>
```

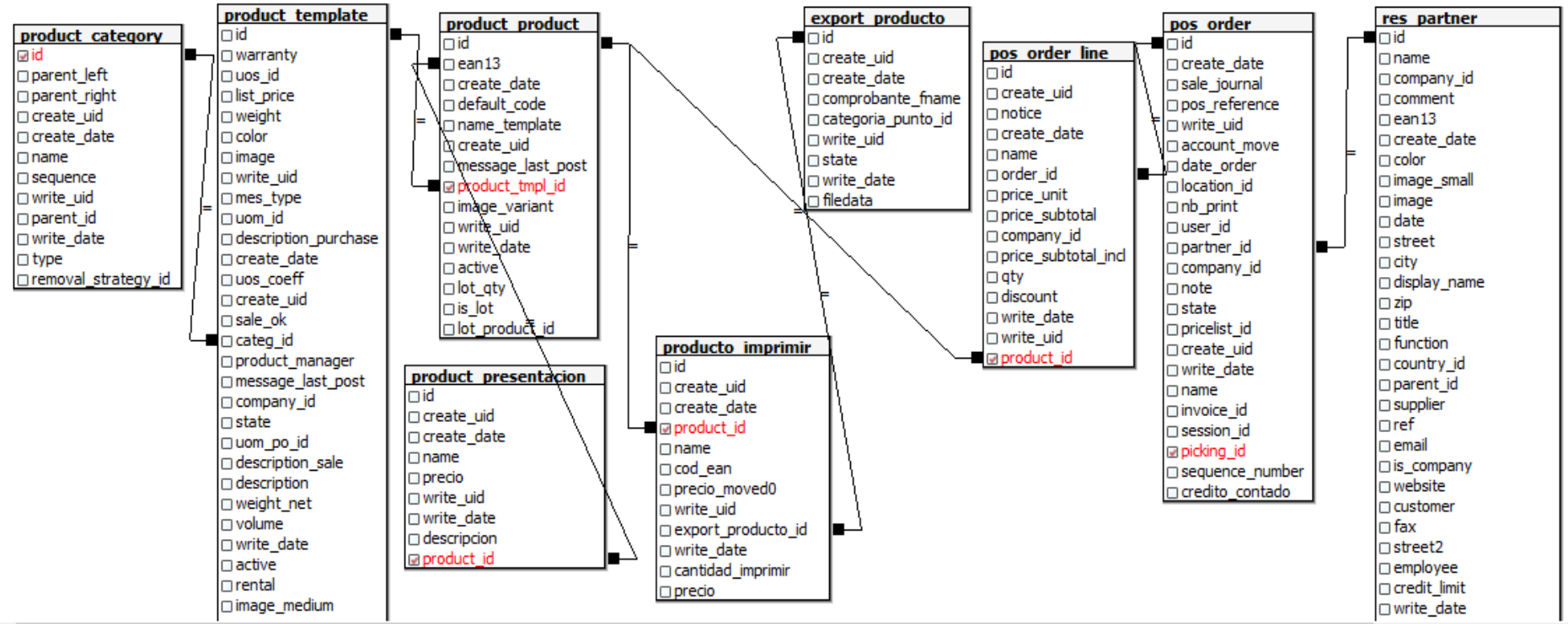
```
</t>
```

El contexto de evaluación JavaScript tiene un objeto de registro que representa el registro que está siendo renderizado, con los campos solicitados del servidor. Los valores de campo pueden ser accedidos usando el atributo raw_value o el value:

- raw_value: Este es el valor retornado por el método de servidor read(), así que se ajusta más para usarse en expresiones condicionales.
- value: Este es formateado de acuerdo a las configuraciones de usuario, y está destinado a ser mostrado en la interfaz del usuario.

El contexto de evaluación de Qweb también tiene referencias disponibles para la instancia JavaScript del cliente web. Para hacer uso de ellos, se necesita una buena comprensión de la arquitectura de cliente web, pero no podremos llegar a ese nivel de detalle. Para propósitos referenciales, los identificadores siguientes están disponibles en la evaluación de expresiones Qweb:

DISEÑO DE LA BASE DE DATOS DE LOS MODULOS MODIFICADOS



DESCRIPCIÓN DE TABLAS DE MODULOS MODIFICADOS:

Tabla	Descripción
Product_product	Los datos principales de los productos
Product_template	Los datos detallados de los productos los cuales pueden ser más extensos y relacionados con otras tablas
Product_presentacion	Tabla que contiene los datos de los pesos y precios de los productos que son tomados en cuenta en el módulo Peso_precio.
Producto_imprimir	Contiene los datos de cuando exportamos los productos para la impresión de etiquetas.
Export_producto	Guarda los datos de peso precio al momento de imprimir
Product_category	Los datos de esta tabla son las variantes que los productos pueden tener.
Pos_order_line	Detalle de las ordenes que se dan al momento de la venta en el punto de venta
Pos_order	Datos principales del punto de venta
Res_partner	Datos del cliente o proveedor

CODIGO FUENTE DE MODULO DESARROLLADOS.

Módulo peso.

peso_precio.py

```
# -*- coding: utf-8 -*-
##############################################################################
#
#
#    OpenERP, Open Source Business Applications
#    Copyright (C) 2004-2012 OpenERP S.A. (<http://openerp.com>).
#
#    This program is free software: you can redistribute it and/or modify
#    it under the terms of the GNU Affero General Public License as
#    published by the Free Software Foundation, either version 3 of the
#    License, or (at your option) any later version.
#
#    This program is distributed in the hope that it will be useful,
#    but WITHOUT ANY WARRANTY; without even the implied warranty of
#    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
#    GNU Affero General Public License for more details.
#
#    You should have received a copy of the GNU Affero General Public License
#    along with this program.  If not, see <http://www.gnu.org/licenses/>.
#
##############################################################################
#

from openerp.osv import osv , fields as field
from openerp import pooler, fields, api, models
from openerp.tools.translate import _
from openerp.exceptions import (
   except_orm,
   Warning as UserError,
   RedirectWarning
   )
import math
import re


class posOrder(models.Model):
          _inherit = 'pos.order'
          def _order_fields(self, cr, uid, ui_order, context=None):
             # for x in ui_order['lines']:
             #    pu = x[2]['price_unit']
             #    q  = x[2]['qty']
             #    sub = round(pu*q,2)
             #    pu_redondeado = sub/q
             #    x[2]['price_unit'] = pu_redondeado
```

```
            # raise osv.except_osv(_('No Customer Defined!'), ui_order)
            return {
               'name':        ui_order['name'],
               'user_id':     ui_order['user_id'] or False,
               'session_id':   ui_order['pos_session_id'],
               'lines':       ui_order['lines'],
               'pos_reference':ui_order['name'],
               'partner_id':   ui_order['partner_id'] or False,
            }

         def _amount_all(self, cr, uid, ids, name, args, context=None):
                  def redondear(n):
                           return round(n*100)/100
                  cur_obj = self.pool.get('res.currency')
                  res = {}
                  for order in self.browse(cr, uid, ids, context=context):
                     res[order.id] = {
                        'amount_paid': 0.0,
                        'amount_return':0.0,
                        'amount_tax':0.0,
                     }
                     val1 = val2 = 0.0
                     cur = order.pricelist_id.currency_id
                     for payment in order.statement_ids:
                        res[order.id]['amount_paid'] +=  payment.amount
                        res[order.id]['amount_return'] += (payment.amount < 0 and
payment.amount or 0)
                     for line in order.lines:

                           t = self._amount_line_tax(cr, uid, line, context=context)
                           val2 += redondear(line.price_subtotal_incl-t)
                        val1 += line.price_subtotal_incl-redondear(line.price_subtotal_incl-t)

                     res[order.id]['amount_tax'] = cur_obj.round(cr, uid, cur, val1)
                     amount_untaxed = cur_obj.round(cr, uid, cur, val2)
                     # raise osv.except_osv(_('No Customer Defined!'),
str(res[order.id]['amount_tax']) + ' - ' +str(val1)+' - '+str(val2)+' - ' + str(amount_untaxed))
                     res[order.id]['amount_total'] = res[order.id]['amount_tax'] +
amount_untaxed
                  return res

         _columns = {
                  'amount_tax': field.function(_amount_all, string='Taxes',
digits_compute=(16,4), multi='all'),
                  'amount_total': field.function(_amount_all, string='Total',
digits_compute=(16,4),  multi='all'),
         }
```

```
class posOrderLine(osv.osv):
        _inherit = 'pos.order.line'
        def _amount_line_all(self, cr, uid, ids, field_names, arg, context=None):
                def redondear(n):
                        return round(n*100)/100
                res = dict([(i, {}) for i in ids])
                account_tax_obj = self.pool.get('account.tax')
                cur_obj = self.pool.get('res.currency')
                for line in self.browse(cr, uid, ids, context=context):
                  taxes_ids = [ tax for tax in line.product_id.taxes_id if tax.company_id.id ==
line.order_id.company_id.id ]
                  price = line.price_unit * (1 - (line.discount or 0.0) / 100.0)
                  taxes = account_tax_obj.compute_all(cr, uid, taxes_ids, price, line.qty,
product=line.product_id, partner=line.order_id.partner_id or False)
                  # raise UserError(taxes)
                  cur = line.order_id.pricelist_id.currency_id
                  res[line.id]['price_subtotal'] = (taxes['total'])
                  res[line.id]['price_subtotal_incl'] = (taxes['total_included'])



                return res
        _columns = {
                'price_subtotal': field.function(_amount_line_all,
multi='pos_order_line_amount',  digits=(16,4), string='Subtotal w/o Tax', store=True),
                'price_subtotal_incl': field.function(_amount_line_all,
multi='pos_order_line_amount',  digits=(16,4), string='Subtotal', store=True),
                }
class producto(models.Model):
        _inherit = 'product.template'
        _description = "producto"
        presentacion_ids =
fields.One2many('product.presentacion','product_id',string='Presentaciones', copy=True)

        def rellenar_ean13(self,ean13):
                """Creates and returns a valid ean13 from an invalid one"""
                if not ean13:
                  return "0000000000000"
                ean13 = re.sub("[A-Za-z]","0",ean13);
                ean13 = re.sub("[^0-9]","",ean13);
                ean13 = ean13[:13]
                if len(ean13) < 13:
                  ean13 = ean13 + '0' * (13-len(ean13))
                return ean13[:-1] + str(self.ean_check(ean13))

        def ean_check(self,eancode):
                """returns the checksum of an ean string of length 13, returns -1 if the string
has the wrong length"""
                if len(eancode) != 13:
```

```
            return -1
        oddsum=0
        evensum=0
        total=0
        eanvalue=eancode
        reversevalue = eanvalue[::-1]
        finalean=reversevalue[1:]

        for i in range(len(finalean)):
            if i % 2 == 0:
                oddsum += int(finalean[i])
            else:
                evensum += int(finalean[i])
        total=(oddsum * 3) + evensum

        check = int(10 - math.ceil(total % 10.0)) %10
        return check

    @api.one
    def setCodE(self):
        if self.ean13 == False:    # problemas
            seq = self.env["ir.sequence"].get("seq_barcode")

            #raise UserError(self.rellenar_ean13(seq))

            self.write({'ean13': self.rellenar_ean13(seq)})


class presentacion(models.Model):
    _name = 'product.presentacion'
    _description = "presentacion"
    _rec_name = "descripcion"
    descripcion = fields.Char('Descripcion',required=True)
    name = fields.Float('Peso',required=True)
    precio = fields.Float('Precio',required=True)
    product_id=fields.Many2one('product.template','producto')

    @api.onchange('name')
    def onchange_peso(self):
        self.precio =self.name*self.product_id.list_price

    @api.onchange('precio')
    def onchange_precio(self):
        if self.name == 0:
            try:
                self.name = self.precio/self.product_id.list_price
            except:
                pass
    @api.one
```

```
        def calcular(self):
                if self.name == 0 and self.precio == 0:
                        return True
                if self.name != 0 and self.precio != 0:
                        return True
                self.onchange_precio()
                self.onchange_peso()

class SaleOrder(models.Model):
        _inherit = 'sale.order.line'
        product_tmpl_id = fields.Many2one(
    related='product_id.product_tmpl_id',
    string='template',
    store=True
    )
        presentacion_id =
fields.Many2one("product.presentacion","Presentacion",domain="[('product_id','=',product_t
mpl_id)]")

        @api.onchange('presentacion_id')
        def onchange_presentacion(self):
                try:
                        precioNuevo = self.presentacion_id.precio/self.presentacion_id.name
                        self.product_uom_qty = self.presentacion_id.name
                        self.price_unit = precioNuevo
                except:
                        pass


        '''@api.onchange('product_id','product_uom_qty')
        def product_id_change_v8(self):
                pricelist = self.order_id.pricelist_id.id
                product = self.product_id.id
                qty = self.product_uom_qty
                uom = self.product_uom.id or False
                qty_uos = self.product_uos_qty  or  0
                uos = self.product_uos.id  or  False
                name = self.name
                partner_id = self.order_id.partner_id.id or False
                lang = False
                update_tax = True
                date_order = self.order_id.date_order
                packaging = False
                fiscal_position = self.order_id.fiscal_position.id
                flag = True
                warehouse_id=self.order_id.warehouse_id.id,
                context = None
                #raise osv.except_osv(_('No Customer Defined!'), partner_id)
```

```
                res =  self.product_id_change_with_wh(pricelist, product, qty,
                        uom, qty_uos, uos, name, partner_id,
                        lang, update_tax, date_order, packaging, fiscal_position,
flag,warehouse_id, context=None)


                #result = self.browse()
                #raise osv.except_osv(_('asdsad'), str(res['value'])+" -- "+str(self))
                #raise self.write(res['value'])
                return {'value': res['value']}
                #raise osv.except_osv(_('asdsad'), str(res['value']['price_unit']))
        '''

        #@api.onchange('product_id','product_uom_qty')
        # @api.multi
        # def product_id_change_with_wh(self, pricelist, product, qty=0,
        #                 uom=False, qty_uos=0, uos=False, name='', partner_id=False,
        #                 lang=False, update_tax=True, date_order=False, packaging=False,
fiscal_position=False, flag=False, warehouse_id=False, context=None):
        #         #if self.presentacion_id:
        #         print str(self.read())
        #         res = super(SaleOrder, self).product_id_change_with_wh(pricelist, product,
qty,
        #                 uom, qty_uos, uos, name, partner_id,
        #                 lang, update_tax, date_order, packaging, fiscal_position,
flag,warehouse_id, context = None)

                # return {'value': res['value']}
        def create(self, cr, uid, values, context=None):
            if values.get('order_id') and values.get('product_id') and  any(f not in values for f in
['name', 'price_unit', 'product_uom_qty', 'product_uom']):
                order = self.pool['sale.order'].read(cr, uid, values['order_id'], ['pricelist_id',
'partner_id', 'date_order', 'fiscal_position'], context=context)
                defaults = self.product_id_change(cr, uid, [], order['pricelist_id'][0],
values['product_id'],
                    qty=float(values.get('product_uom_qty', False)),
                    uom=values.get('product_uom', False),
                    qty_uos=float(values.get('product_uos_qty', False)),
                    uos=values.get('product_uos', False),
                    name=values.get('name', False),
                    partner_id=order['partner_id'][0],
                    date_order=order['date_order'],
                    fiscal_position=order['fiscal_position'][0] if order['fiscal_position'] else False,
                    flag=False,  # Force name update
                    context=dict(context or {}, company_id=values.get('company_id'))
                )['value']
                if defaults.get('tax_id'):
                    defaults['tax_id'] = [[6, 0, defaults['tax_id']]]
```


```
            values = dict(defaults, **values)
        try:
                if(values['presentacion_id']):
                    pres = self.pool['product.presentacion'].read(cr,
uid,values['presentacion_id'])
                    values['price_unit'] = pres['precio']/pres['name']
        except:
            pass
            #raise osv.except_osv(_('asdsad'), str(pres['precio']/pres['name']))
        #raise osv.except_osv(_('asdsad'), str(values))
        return super(SaleOrder, self).create(cr, uid, values, context=context)
    def write(self, cr, uid, ids, vals, context=None):
            try:
                if(vals['presentacion_id']):
                    pres = self.pool['product.presentacion'].read(cr,
uid,vals['presentacion_id'])
                    vals['price_unit'] = pres['precio']/pres['name']
                raise osv.except_osv(_('asdsad'),str(vals))
            except:
                pass
            res = super(SaleOrder, self).write(cr, uid, ids, vals, context=context)
            return res
```

partnet.py

```
# -*- coding: utf-8 -*-
##############################################################################
#
#
#   OpenERP, Open Source Business Applications
#   Copyright (C) 2004-2012 OpenERP S.A. (<http://openerp.com>).
#
#   This program is free software: you can redistribute it and/or modify
#   it under the terms of the GNU Affero General Public License as
#   published by the Free Software Foundation, either version 3 of the
#   License, or (at your option) any later version.
#
#   This program is distributed in the hope that it will be useful,
#   but WITHOUT ANY WARRANTY; without even the implied warranty of
#   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
#   GNU Affero General Public License for more details.
#
#   You should have received a copy of the GNU Affero General Public License
#   along with this program.  If not, see <http://www.gnu.org/licenses/>.
#
##############################################################################
#

from openerp.osv import osv
```

```
from openerp import pooler, fields, api, models
from openerp.tools.translate import _

class partner(models.Model):
   _inherit = 'res.partner'
   _name = 'res.partner'
   @api.model
   def create(self, vals):
      try:
         if vals['property_account_position']:
            vals['property_account_position'] = int(vals['property_account_position'])
      except:
         pass
      rec = super(partner, self).create(vals)
      return rec

   @api.multi
   def write(self, vals):
      try:
         if vals['property_account_position']:
            vals['property_account_position'] = int(vals['property_account_position'])
      except:
         pass

      rec = super(partner, self).write(vals)
      return rec
```

pos_config.py

```
# # -*- coding: utf-8 -*-
#
##############################################################################
#
# #
# #    OpenERP, Open Source Business Applications
# #    Copyright (C) 2004-2012 OpenERP S.A. (<http://openerp.com>).
# #
# #    This program is free software: you can redistribute it and/or modify
# #    it under the terms of the GNU Affero General Public License as
# #    published by the Free Software Foundation, either version 3 of the
# #    License, or (at your option) any later version.
# #
# #    This program is distributed in the hope that it will be useful,
# #    but WITHOUT ANY WARRANTY; without even the implied warranty of
# #    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
# #    GNU Affero General Public License for more details.
# #
# #    You should have received a copy of the GNU Affero General Public License
# #    along with this program.  If not, see <http://www.gnu.org/licenses/>.
# #
```

```
#
##############################################################################
#

from openerp.osv import osv
from openerp import pooler, fields, api, models
from openerp.tools.translate import _
class pos_config(models.Model):

   _inherit = 'pos.config'
   _name = 'pos.config'

   fiscal_position_default = fields.Many2one("account.fiscal.position", string="Posicion fiscal
por defecto en esta caja")
   country_default_id = fields.Many2one("res.country", string="Pais por defecto")

exportpro.py
# -*- encoding: utf-8 -*-
##############################################################################

from openerp import models, fields, api, _
from openerp.osv import osv
import openerp.addons.decimal_precision as dp
from openerp.tools.translate import _
from openerp.tools import config
import time
from xml.etree.ElementTree import Element, SubElement, ElementTree, tostring
import sys
import base64
import xlwt
#import StringIO
from io import BytesIO , StringIO
import gzip
from openerp.exceptions import (
   except_orm,
   Warning as UserError,
   RedirectWarning
   )

#from xlsxwriter import workbook as Workbook
#import StringIO


class export_prod(models.Model):



   @api.one
   def generate_excel(self):
```

```
        output = BytesIO()
        ext = '.xls'
        style0 = xlwt.easyxf('font: name Times New Roman, colour red, bold on')
        style1 = xlwt.easyxf('',num_format_str='$#,##0.00')
        wb = xlwt.Workbook(encoding="utf-8")
        ws = wb.add_sheet('Hoja de Calculo',cell_overwrite_ok=True)

        ws.write(0, 0,'productos')
        ws.write(0, 1,'CodigoEan')
        ws.write(0, 2,'Precio')
        ws.write(0, 3,'Cantidad')
        i = 1
        for line in self.productos_imprimir_ids:
            ws.write(i, 0, line.name)
            ws.write(i, 1, line.cod_ean)
            ws.write(i, 2, line.precio , style1)
            ws.write(i, 3, line.cantidad_imprimir)
            i=i+1

        wb.save(output)

        return base64.b64encode(output.getvalue())# base64.encodestring(f.read())

    @api.multi
    def act_export(self):

        #root = self.generate_excel(cr,uid,ids)
        #self._write_attachment(cr,uid,ids,root,context)
        out = self.generate_excel()

        self.write({'filedata':out,'state':'load','comprobante_fname':'productos.xls'})
        #raise UserError('Horror')
        return {
            'type': 'ir.actions.act_window',
            'res_model': 'export.producto',
            'view_mode': 'form',
            'view_type': 'form',
            'res_id': self.id,
            'views': [(False, 'form')],
            'target': 'new',
             }


    _name = 'export.producto'

    categoria_punto_id = fields.Many2one('pos.category', string = 'Categoria de TPV' )
    state = fields.Selection([('choose','Choose'),('get','Get'),('load','Load'),],  'state',
required=True, readonly=True)
```

```
    filedata = fields.Binary('Texto',filters='*.xls')
    comprobante_fname = fields.Char( default="productos.xls")

    productos_imprimir_ids = fields.One2many('producto.imprimir', 'export_producto_id', string
= 'Productos a Imprimir' )
    _defaults = {
            'state': lambda *a: 'choose'
            }



    @api.multi
    def cargartarifas(self):
        lista = []
        for line in self.productos_imprimir_ids:
            lista.append(line)
        #raise UserError(lista)
        for line in lista:
            #raise UserError('skdksad')
            if line.product_id.presentacion_ids!=False:
                for pro in line.product_id.presentacion_ids:

                    name = line.product_id.name[:14].replace("1 LB","").replace("1LB","")+'..
'+pro.descripcion
                    cod_ean = line.product_id.ean13+("%.2f" %
round(pro.name,2)).replace(".","").zfill(4)#+("%.2f" %
round(pro.precio,2)).replace(".","").zfill(4)
                    precio = pro.precio

                    temp = self.env['producto.imprimir'].create({
                        'name':name,
                        'cod_ean':cod_ean,
                        'precio': "{0:.2f}".format(precio).replace(".",","),
                        'cantidad_imprimir':1,

                    })

                    self.productos_imprimir_ids |= temp
        self.state = 'get'
        return {
            'type': 'ir.actions.act_window',
            'res_model': 'export.producto',
            'view_mode': 'form',
            'view_type': 'form',
            'res_id': self.id,
            'views': [(False, 'form')],
            'target': 'new'
        }
```

```
    @api.multi
    def cargarproductos(self):
        #self.productos_imprimir_ids = None

        #raise UserError(self.productos_imprimir_ids)
        if len(self.categoria_punto_id)==0:
            self.cargartarifas()
            self.state = 'get'
            return {
                'type': 'ir.actions.act_window',
                'res_model': 'export.producto',
                'view_mode': 'form',
                'view_type': 'form',
                'res_id': self.id,
                'views': [(False, 'form')],
                'target': 'new'
             }

        productos = self.env['product.template']

        res = productos.search(['&', ('pos_categ_id', '=', self.categoria_punto_id.id), ('active', '=',
'True')])
        for pro in res:
            name = pro.name
            cod_ean = pro.ean13
            precio = pro.list_price
            self.productos_imprimir_ids |= self.env['producto.imprimir'].create({
                'name':name,
                'cod_ean':cod_ean,
                'precio':precio,
                'cantidad_imprimir':1,
                })
        self.state = 'get'
        return {
            'type': 'ir.actions.act_window',
            'res_model': 'export.producto',
            'view_mode': 'form',
            'view_type': 'form',
            'res_id': self.id,
            'views': [(False, 'form')],
            'target': 'new'
             }

class producto_imprimir(models.Model):


    _name = 'producto.imprimir'
    product_id =fields.Many2one('product.template', string = 'Producto')
```

```
    name = fields.Char(string = 'Descripcion' )
    cod_ean = fields.Char(string = 'Codigo Ean' )
    precio = fields.Char(string = 'Precio' )
    cantidad_imprimir = fields.Integer(string = 'Cantidad' )

    export_producto_id = fields.Many2one('export.producto', string = 'Clave de Exportacion')

    @api.one
    @api.onchange('product_id')
    def product_id_changes(self):
        try:
            self.name = self.product_id.name[:24]
            self.cod_ean = self.product_id.ean13
            self.precio = "{0:.2f}".format(self.product_id.list_price).replace(".",",")
            self.cantidad_imprimir = 1

        except:
            pass
        return True


export.xml
<?xml version="1.0" encoding="utf-8"?>
<openerp>
        <data>

                <record id="wizard_export_producto" model="ir.ui.view">
            <field name="name">export.producto.wizard.view</field>
            <field name="model">export.producto</field>
            <field name="type">form</field>
            <field name="arch" type="xml">
                <form string="Export Productos">
                    <group col="8">
                        <group colspan="4" >
                            <label colspan="4" width="250" string="Asistente para la impresion de codigo
EAN:"/>
                        </group>
                        <separator orientation="vertical" rowspan="15"/>
                        <group colspan="4">
                            <group colspan="4" states="choose">
                                <separator colspan="4" string="XLS Exportacion"/>
                                <field name="categoria_punto_id" />
                            </group>

                            <group colspan="4" states='choose,get'>
                                <field name="productos_imprimir_ids" nolabel="1" />
                                <field invisible="1" name="state"/>
                            </group>
                            <group colspan="4" states='choose'>
```

```
                    <button icon="gtk-ok" name ="cargarproductos" string="Cargar"
type="object"/>
                </group>

                <group colspan="4" states="load">
                  <separator string="Guarde el archivo en su computador" colspan="4"/>
                  <field name="comprobante_fname" invisible="0" />
                  <field name="filedata" nolabel="1"  filename="comprobante_fname"
readonly="1" colspan="4"/>

                </group>
              </group>
              <group colspan="8" col="8" states="choose">


              </group>
               <group colspan="8" col="8" states="get">
                <separator string="" colspan="8"/>
                <label colspan="7" string="Esta!!!!" width="220"/>


              </group>
            </group>

             <footer>
             <button icon="gtk-cancel" name="act_cancelar" special="cancel" string="_Close"
type="object" states="choose"/>
             <button icon="gtk-ok" name ="act_export" string="_Export" type="object"
states="get,choose"/>
             <button icon="gtk-close" name="act_destroy" special="cancel" string="_Close"
type="object" states="get" />
             </footer>
          </form>
        </field>
    </record>

    <record id="action_wizard_imprimir_producto" model="ir.actions.act_window">
      <field name="name">Exportar productos xls</field>
      <field name="type">ir.actions.act_window</field>
      <field name="res_model">export.producto</field>
      <field name="view_type">form</field>
      <field name="view_mode">form</field>
      <field name="target">new</field>
    </record>

                <menuitem id="menu_impresion_codigos" name="Impresion de Codigos"
parent="base.menu_reporting" />
                <menuitem id="menu_productos_imp" name="Listado de Productos"
parent="menu_impresion_codigos" action ='action_wizard_imprimir_producto'/>
```

```
    <record id="tree_producto_imprimir" model="ir.ui.view">
      <field name="name">producto.imprimir.wizard.view</field>
      <field name="model">producto.imprimir</field>
      <field name="type">tree</field>
      <field name="arch" type="xml">
        <tree editable="bottom">
          <field name="product_id"/>
          <field name="name"/>
          <field name="cod_ean"/>
          <field name="precio"/>
          <field name="cantidad_imprimir"/>
        </tree>
      </field>
    </record>
      </data>
</openerp>
```

peso_precio_view.xml

```
<?xml version="1.0" encoding="utf-8"?>
<openerp>
  <data>
    <record model="ir.ui.view" id="product_pesoprecion_form_view_inherit">
      <field name="name">product.template.pesoprecio</field>
      <field name="model">product.template</field>
      <field name="inherit_id" ref="product.product_template_only_form_view"/>
      <field name="arch" type="xml">
        <div name="options" position="inside">
          <div>
            <field name="presentacion_ids">
              <tree editable="bottom">
                <field name="descripcion" />
                <field name="name" />
                <field name="precio" />
                <button name="calcular" type="object" string="calcular"/>
              </tree>
            </field>

          </div>
        </div>
      </field>
    </record>
    <record id="product_template_form_view_inherit_ean_generate" model="ir.ui.view">
      <field name="name">product.template.only.form.inherit.ean.generate</field>
      <field name="model">product.template</field>
      <field name="inherit_id" ref="product.product_template_only_form_view"/>
      <field name="arch" type="xml">
```

```
            <field name="ean13" position="before">
              <button colspan="2" name="setCodE" type="object" string="Generar Codigo Ean"
                  attrs="{'invisible': [('product_variant_count', '>', 1)]}" class="oe_link
oe_edit_only"/>
            </field>

          </field>

        </record>

        <record id="sale_order_presentacion_form" model="ir.ui.view">
          <field name="name">sale.order.generate</field>
          <field name="model">sale.order</field>
          <field name="inherit_id" ref="sale.view_order_form"/>
          <field name="arch" type="xml">


            <xpath expr="//field[@name='order_line']/tree/field[@name='name']"
position="after">
              <field name="product_tmpl_id" invisible="1"/>
              <field name="presentacion_id" widget="selection"/>
            </xpath>

          </field>

        </record>
    </data>
</openerp>
```

Módulo product_add_iva

product_add_iva.py

```
from openerp.osv import osv
from openerp import pooler, fields, api, models
from openerp.tools.translate import _
from openerp.exceptions import (
   except_orm,
   Warning as UserError,
   RedirectWarning
   )
```

```
class Company(models.Model):

        _inherit = "res.company"
        _descripcion = "company"

        producto_iva_ids = fields.Many2many(string="Impuestos de venta diferente de
0%",comodel_name='account.tax',
                relation='tax_company_iva_supplier_rel')
        producto_iva_compras_ids = fields.Many2many(string="Impuestos de compra",
comodel_name='account.tax',
                relation='tax_company_iva_rel')

        producto_iva0_id = fields.Many2one('account.tax',string="Impuestos de venta 0%")
        producto_iva0_compras_id = fields.Many2one('account.tax',string="Impuestos de
compra 0%")


class Product(models.Model):
        _inherit = 'product.template'
        _description = "producto"

        @api.multi
        def generarIva(self):
                for record in self:
                        record.taxes_id = None
                        record.supplier_taxes_id = None
                        for line_tax in record.env.user.company_id.producto_iva_ids:
                                record.taxes_id |= line_tax
                        for line_tax_supplier in
record.env.user.company_id.producto_iva_compras_ids:
                                record.supplier_taxes_id |= line_tax_supplier



        @api.multi
        def quitarIva(self):
                for record in self:
                        record.taxes_id = None
                        record.supplier_taxes_id = None
                        record.taxes_id |= record.env.user.company_id.producto_iva0_id
                        record.supplier_taxes_id |=
record.env.user.company_id.producto_iva0_compras_id
```

product_add_iva.xml

```
<?xml version="1.0" encoding="utf-8"?>
<openerp>
```

```
  <data>
    <record model="ir.ui.view" id="product_add_iva_form_view_inherit">
      <field name="name">res.company.addiva</field>
      <field name="model">res.company</field>
      <field name="inherit_id" ref="account_check_writing.check_format_company"/>
      <field name="arch" type="xml">
        <field name="check_layout" position="after">
          <field name="producto_iva0_id"/>
          <field name="producto_iva0_compras_id" />
          <field name="producto_iva_ids" widget="many2many_tags"/>
          <field name="producto_iva_compras_ids" widget="many2many_tags"/>

        </field>
      </field>
    </record>

    <record id="product_template_form_view_tax_auto" model="ir.ui.view">
      <field name="name">product.template.form.tax.auto</field>
      <field name="model">product.template</field>
      <field name="inherit_id" ref="account.product_template_form_view"/>
      <field name="arch" type="xml">
        <field name="taxes_id" position="after">
          <button colspan="2" name="generarIva" type="object" string="Iva Distinto a 0%"
              attrs="{'invisible': [('product_variant_count', '>', 1)]}" class="oe_link"/>

          <button colspan="2" name="quitarIva" type="object" string="Iva 0%"
              attrs="{'invisible': [('product_variant_count', '>', 1)]}" class="oe_link"/>
        </field>

      </field>

    </record>

  </data>
</openerp>
```

Modulo verificador

contenedor.py

```
from openerp import models, fields, api, _
```

```
class rbs_contenedor(models.Model):
        _name="rbs.contenedor"
        _description=""

        name = fields.Char(string='Contenedor')
        imagenes_ids = fields.One2many("rbs.imagenes","contenedor_id","imagenes")



class rbs_imagenes(models.Model):
        _name="rbs.imagenes"
        _description=""

        imagen = fields.Binary(string="Imagen")
        contenedor_id = fields.Many2one("rbs.contenedor", string="Contenedor de
imagenes")

        def actualizarImagen (self, cr, uid, id, binary):
                self.browse(cr, uid, id).write({"imagen":binary})
```

verificador.py

```
from openerp import models, fields, api, _
from openerp.osv import osv



class verificador(models.Model):
        _name = 'rbs.verificador'


        def open_verificador(self, cr, uid, ids, context=None):
                data = self.browse(cr, uid, ids[0], context=context)
                context = dict(context or {})
                #context['active_id'] = data.ids[0]
                return {
                        'type' : 'ir.actions.act_url',
                        'url':   '/verificador/web',
                        # 'target': 'current',
                }
```

Verificador.xml

```
<?xml version="1.0" encoding="utf-8"?>
<openerp>
   <data>
```

```
        <record id="veridicador_view" model="ir.ui.view">
      <field name="name">Abrir el verificador</field>
      <field name="model">rbs.verificador</field>
      <field name="arch" type="xml">
      <form string="Report Options">
        <footer>
          <button name="open_verificador" string="Abrir" type="object" default_focus="1"
class="oe_highlight"/>
          or
          <button string="Cancel" class="oe_link" special="cancel" />
        </footer>
      </form>
      </field>
    </record>

    <record id="accion_verificador" model="ir.actions.act_window">
      <field name="name">Abrir el verificador</field>
      <field name="res_model">rbs.verificador</field>
      <field name="view_type">form</field>
      <field name="view_mode">form</field>
      <field name="view_id" ref="veridicador_view"/>
      <field name="target">new</field>
    </record>


    <menuitem id="submenu_verificador" name="Verificador"  action="accion_verificador"
groups="group_user_verificador"/>



            <template id="Verficador" name="POS Index">&lt;!DOCTYPE html&gt;
                <html>
                    <head>
                        <title>Verificador Precio</title>
                        <meta charset="utf-8" />
                        <meta name="viewport" content="width=device-
width, initial-scale=1" />

                        <link rel="stylesheet"
href="/verificador_precios/static/src/css/main.css" />
<!--
                        <link rel="stylesheet"
href="/verificador_precios/static/src/css/mainProduct.css" />
                        <noscript><link rel="stylesheet"
href="/verificador_precios/static/src/css/noscript.css" /></noscript> -->
```

```
                                        <script type="text/javascript"
src="/verificador_precios/static/src/js/main.js"></script>
                                        <script type="text/javascript"
src="/verificador_precios/static/src/libs/js/jquery.js"></script>
                                        <t t-call-assets="web.assets_common" t-css="false" />
                        <t t-call-assets="web.assets_backend" t-css="false" />
                                        <script type="text/javascript" id="loading-script">
                                    $(function() {
                                      <t t-raw='init' />
                                    });
                                </script>
                                    </head>
                                    <body>
                                        <!--<div id="wrapper">
                                            < div id="bg"></div>
                                            <div id="overlay"></div>
                                        </div>  -->
                                        <!-- <script>
                                            window.onload = function() {
document.body.className = ''; }
                                            window.ontouchmove = function() { return
false; }
                                            window.onorientationchange = function() {
document.body.scrollTop = 0; }
                                        </script> -->

                                            <!-- Intro -->
                                        <div id="intro">


                                                    <!-- <ul class="actions">
                                                        <li><a href="#header"
class="button icon solo fa-arrow-down scrolly">Continue</a></li>
                                                    </ul> -->


                                        </div>
                                        <div id="infoproducto">
                                            <div id = 'cabecera'>
                                                <div id = 'cab-nombre'>
                                                    <span id="nombre"></span>
                                                </div>
                                                <div id = 'cab-imagen'>
                                                    <img id="imagen"/>
                                                </div>

                                            </div>
```

```
                                        <div id='cuerpo'>
                                                <div id='precios'>
                                                        <span id="precio"></span>
                                                        <div id="variantes">
                                                        </div>
                                                </div>
                                        </div>
                                        <div id='pie'>
                                                <div id="pie-pagina">
                                                </div>

                                        </div>
                                </div>
                        </body>
                </html>
        </template>

  </data>
</openerp>
```

Todo Criollo

MANUAL DE USUARIO

APLICACIÓN MOVIL CATALOGO

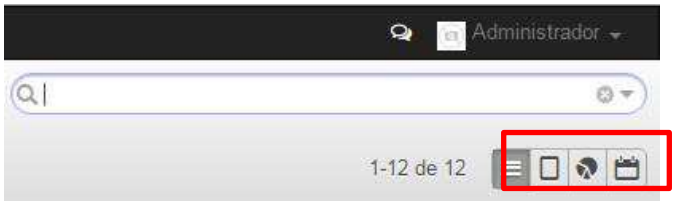
MANUAL DE USUARIO

En el presente documento se observará de manera detallada el manejo de los diferentes botones y funcionalidades del software implementado. Entre todos ellos, destacan algunos elementos básicos que se utilizan frecuentemente en el manejo de la interfaz del sistema:

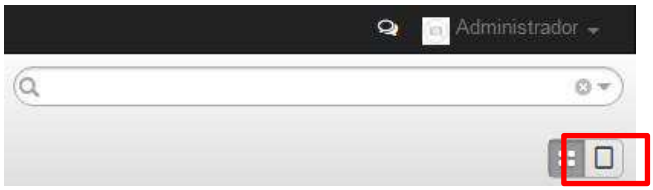
Botones de acciones: son los botones de **Crear**, **Importar**, **Editar**, **Guardar**, **Descartar** y **Aplicar** entre otros, utilizados en la mayor parte de los módulos y funciones del sistema Odoo. Cada vez que queramos generar un nuevo elemento (producto, proveedor, diario contable, pedido de compra, ruta, etc.) debemos pinchar en Crear situado en la parte superior izquierda de la pantalla. Una vez introducidos todos los datos de los elementos, podemos confirmar o eliminar dichos elementos utilizando los botones Guardar y Descartar respectivamente. Para modificar los datos de algún elemento creado presionamos en Editar. Y, por último, se utiliza Aplicar cuando se quieren aplicar determinadas opciones en los módulos de la aplicación.

Barra de búsqueda avanzada: situada en la esquina superior derecha de la pantalla, nos ayuda a localizar elementos del sistema empleando filtros de búsqueda, que se emplean por los diferentes tipos de campos existentes en cada uno de los módulos.

Vistas de la interfaz: la visualización de los elementos dentro de los módulos se puede modificar según nuestras preferencias. El sistema permite seleccionar varios tipos de vista: lista, formulario, Kanban, gráfico y calendario. Estas opciones se pueden cambiar en la esquina superior derecha de la pantalla justo debajo de la barra de búsqueda avanzada.



Vistas de la interfaz y la barra de búsqueda avanzada



Otras vistas (Kanban y formulario)

Creación y configuración de datos maestros

Una vez instalados los módulos y realizada su configuración general, vamos a explicar cómo crear usuarios, productos, proveedores y clientes. Es importante destacar que esta tarea se realiza después de la instalación de todos los módulos necesarios, si no fuese así, no se podrían crear ni los productos, ni los proveedores, ni los clientes. Además, las cuentas de cada usuario permiten el acceso a ciertas funciones del sistema y a configurar determinados módulos si se les dan los permisos correspondientes, por tanto, resulta más sencillo de esta forma.

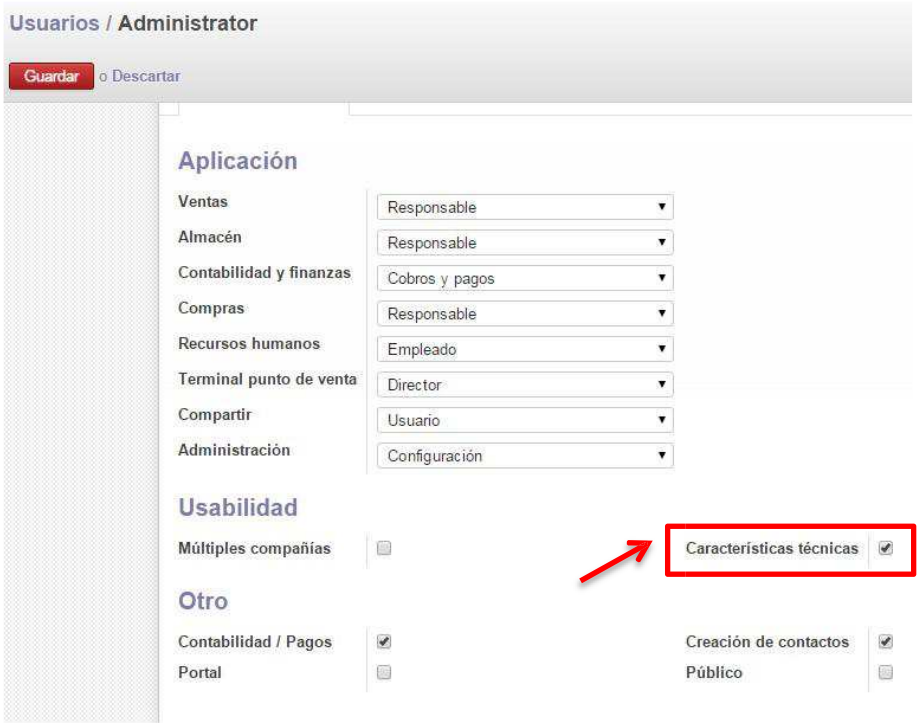
Usuarios

Para poder comenzar a trabajar con el sistema Odoo, existe una cuenta de usuario creada por defecto que toma el nombre de Administrador.

Gracias a ella podemos acceder al programa con todos los permisos y accesos posibles para configurar las aplicaciones, la vista de la interfaz o incluso crear nuevas cuentas de usuario del sistema. Dado que la empresa con la que trabajamos va a requerir varios usuarios para la gestión de sus establecimientos vamos a explicar cómo se crea un usuario nuevo y se le dan los permisos correspondientes a su cargo dentro del negocio.

Primero, entramos en el sistema como Administrador, vamos a Configuración (en la barra superior de la pantalla), en el menú de la izquierda nos situamos en la pestaña Usuarios. Así aparece una lista de las cuentas de usuario que posee el sistema, pero si damos a Crear se pueden añadir nuevas cuentas a la lista.

Antes de crear el primer usuario, es conveniente entrar en la configuración del Administrador y marcar la opción Características técnicas. Sirve para añadir un conjunto de funciones que permiten personalizar ciertos aspectos del sistema como flujos, formatos y secuencias de informes, etc. Incluso es recomendable seleccionar esta opción antes de instalar cualquier módulo del sistema.



Vista de la opción Características técnicas en la configuración del usuario Administrador

A la hora de crear un nuevo usuario aparece una serie de campos a rellenar como nombre de usuario y el nombre para acceder al sistema o correo electrónico (equivalente a admin). Para el nuevo usuario de la empresa hay tres apartados: Permisos de acceso, Preferencias y Terminal punto de venta Según los módulos que tengamos instalados surgen sus apartados y opciones correspondientes (como el de TPV).

En este caso, dentro de la pestaña **Permisos de acceso** podemos configurar los siguientes aspectos:

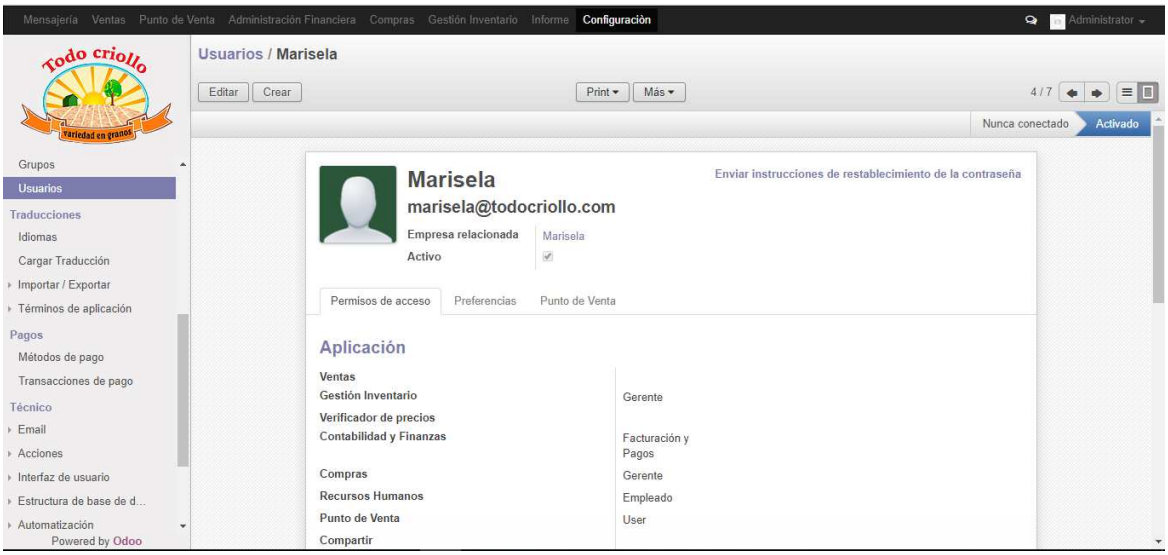
Acceso a las aplicaciones: establece permisos para cada módulo. Según el usuario escoja una de las opciones (responsable, director, empleado, etc.) se restringen las

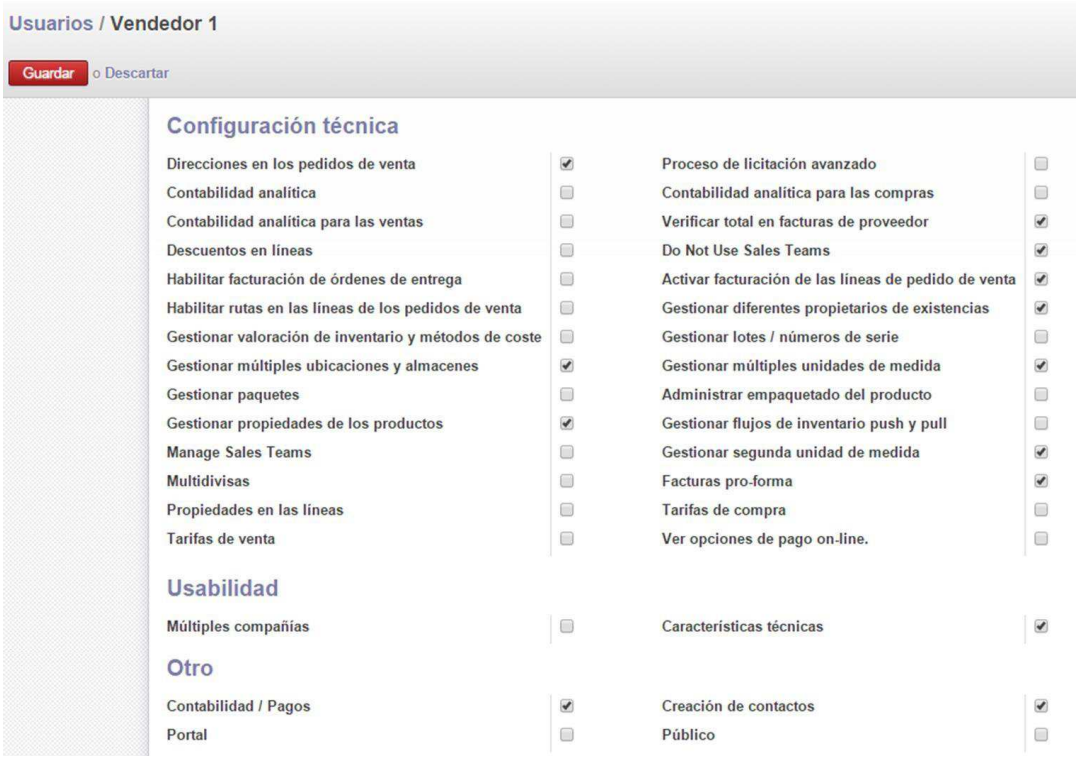
actividades que pueden realizar dentro de cada aplicación. Básicamente hay dos papeles dentro del sistema:

- **Responsable o director**: puede acceder a cada uno de los módulos asignados a él y modificar sus características y los datos que contienen.
- **Empleado o usuario**: puede acceder a los módulos, pero no realizar modificación alguna sobre ellos.

Podemos ver los distintos papeles en cada módulo del usuario. En algunos actúa como responsable (Ventas, Compras, Almacén, Terminal punto de venta) para poder configurarlos y manejarlos sin problemas.

- Características técnicas: permiten asignar al usuario una serie de acciones relacionadas con las distintas aplicaciones. Se fijarán para el usuario aquellas que conciernen a su puesto y la forma en la que pretende gestionar las actividades de la empresa. En este caso seleccionamos las que consideramos más adecuadas y que corresponden a la configuración general de los módulos, como las referidas a los productos que tratan de gestionar sus propiedades. Cuando hagamos la parametrización detallada de los módulos es muy probable que haya que volver a este punto para adaptar al usuario correctamente.
- Opciones de usabilidad: permiten al usuario acceder a más de una compañía y trabajar con ellas (Múltiples compañías o Multi Company), y añaden nuevas funciones en el menú (la opción de Configuraciones técnicas).
- Otras opciones: en este punto aparecen opciones que permiten al usuario configurar todo lo relacionado con contabilidad y pagos, crear contactos en el sistema y convertir el perfil del usuario en público.



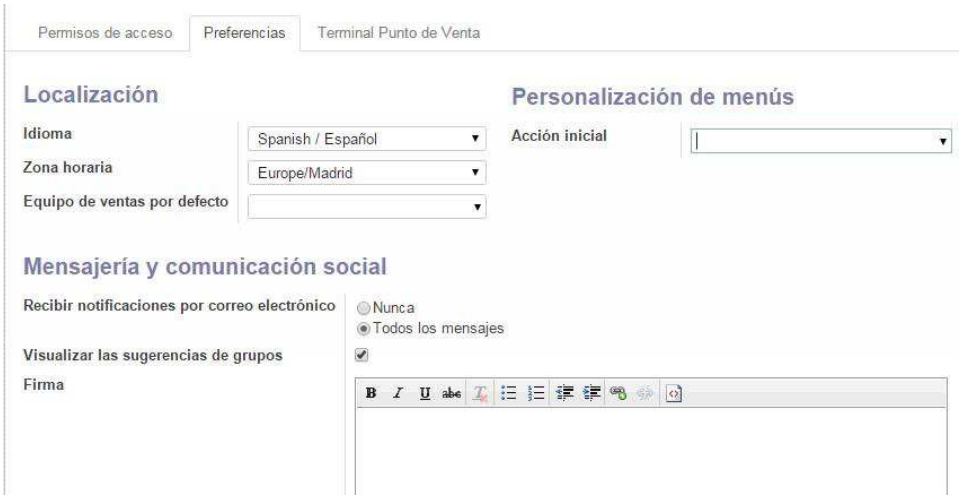


El siguiente apartado es el de **Preferencias**. Aquí podemos modificar aspectos como el idioma que utiliza el usuario y el que aparece en los documentos asociados a éste, la zona horaria en la que se encuentra y el equipo de ventas al que corresponde por defecto.

También se puede elegir cuando y como recibir los correos (nunca o todos en la bandeja de entrada), si el usuario desea ver las notificaciones de los grupos a los que pertenezca y editar su firma personal. En cuanto a las características del menú, se puede seleccionar una acción predeterminada de forma que cada vez que el usuario entra en el sistema, la ejecuta directamente.

Por último, se incorpora un apartado correspondiente a uno de los módulos instalados como es **Terminal punto de venta**. Volveremos a él cuando hayamos configurado dicho módulo y podamos ajustar sus características al usuario. De momento solo describimos los campos que contiene:

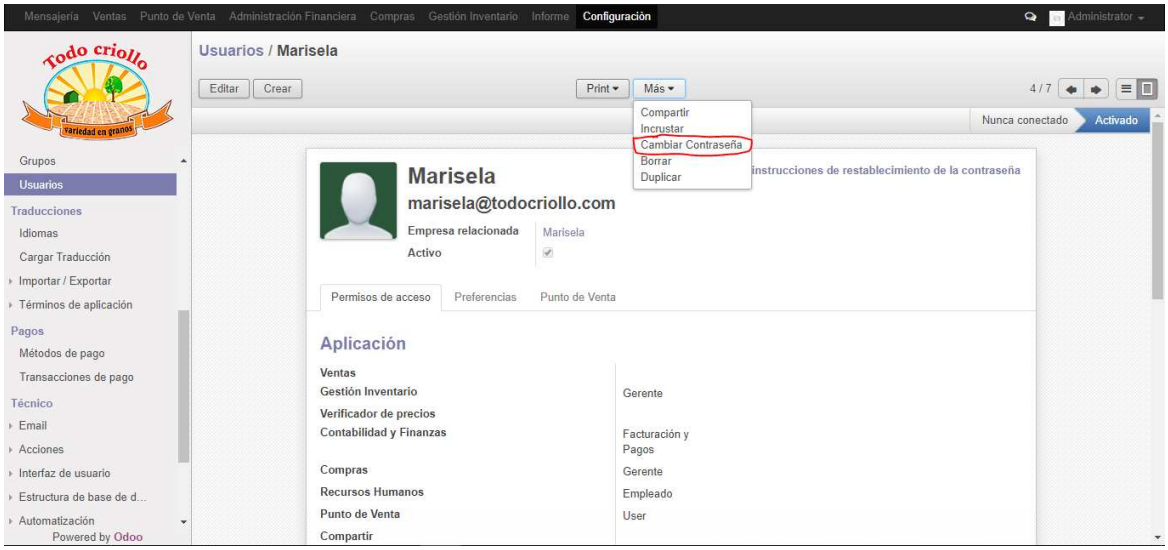
- **Terminal punto de venta (TPV) por defecto**: se establece en que punto de venta (tienda) trabaja el usuario, es decir, indica la tienda donde realiza las ventas de productos a los clientes.
- **Código de barras asociado (EAN13)**: Odoo es capaz de generar un código de barras válido según el sistema EAN introduciendo la correspondiente referencia del usuario.



De momento ya hemos especificado todos los parámetros del usuario necesarios, aunque es muy probable que posteriormente haya que volver a esta parte para modificar algún

aspecto o añadir alguna opción. Para terminar de crear el usuario vamos al botón **Guardar** y asignamos una contraseña de acceso al sistema pinchando en Más y seleccionando Cambiar contraseña; así aparece otra ventana donde poder escribirla.

Después, comprobamos que el usuario creado (Marisela) puede entrar en el sistema Odoo introduciendo su correo electrónico y la contraseña.



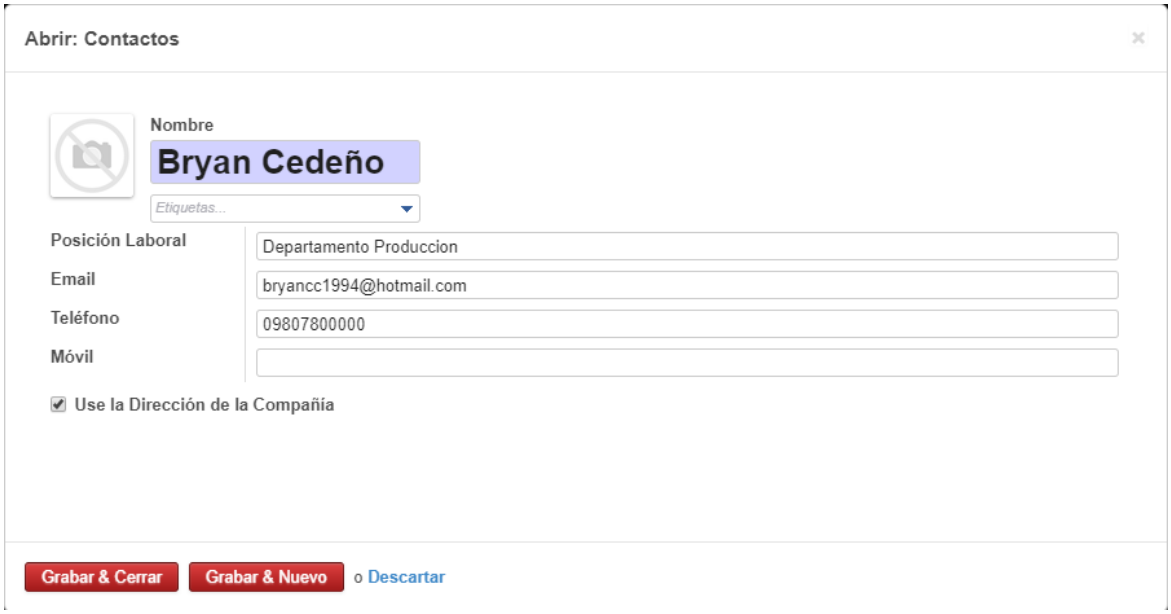
Proveedores y clientes

El sistema ERP Odoo permite crear proveedores y clientes de la empresa. A ambos se les denomina Partner y la forma de generarlos es muy similar salvo por algún parámetro específico. En teoría, los proveedores se crean desde el módulo de Compras (Purchases) y los clientes desde el módulo de Ventas, pero realmente se pueden introducir en el sistema a través otros módulos como el de Contabilidad.

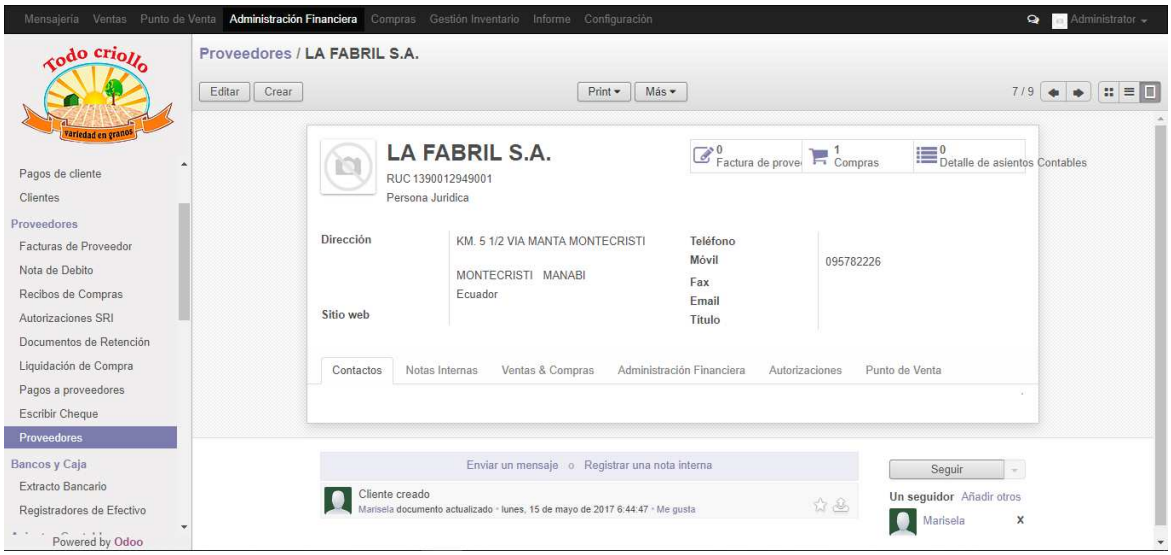
Comenzamos con la creación de los proveedores. Una vez hemos accedido al sistema como Administrador o usuario con permisos, vamos a **Compras** (Purchases) en la

barra superior de la pantalla, y en el menú izquierdo nos dirigimos a Suppliers (Proveedores). Si tenemos creado algún proveedor aparecerá en este apartado, sino podemos crearlo pinchando en **Crear**. De esta forma, surge una nueva ventana donde introducir todos los datos necesarios. Lo primero que debemos rellenar son los campos sobre información general:

- Empresa o contacto: definimos si nuestro proveedor va a ser un contacto (una persona física o un particular) o una empresa (entidad). Si se trata de una empresa, nos da la opción de añadir una lista de contactos o personas pertenecientes a ella.
- Nombre, imagen o logo para identificarla, etiqueta o categoría a la que pertenece la empresa. Las categorías se pueden crear en este apartado o yendo a Configuración, Contactos y Etiquetas de empresa. Por ejemplo, vamos a definir la categoría padre Proveedores y la categoría hijo y damos a Guardar y así queda clasificada la empresa proveedora como aquella que vende productos frescos a las tiendas.
- Datos básicos: dirección (calle, ciudad, código postal, etc.), página web de la empresa, teléfono, móvil, email, fax, tipo de empresa según el criterio jurídico (S.A., S.L., etc.).



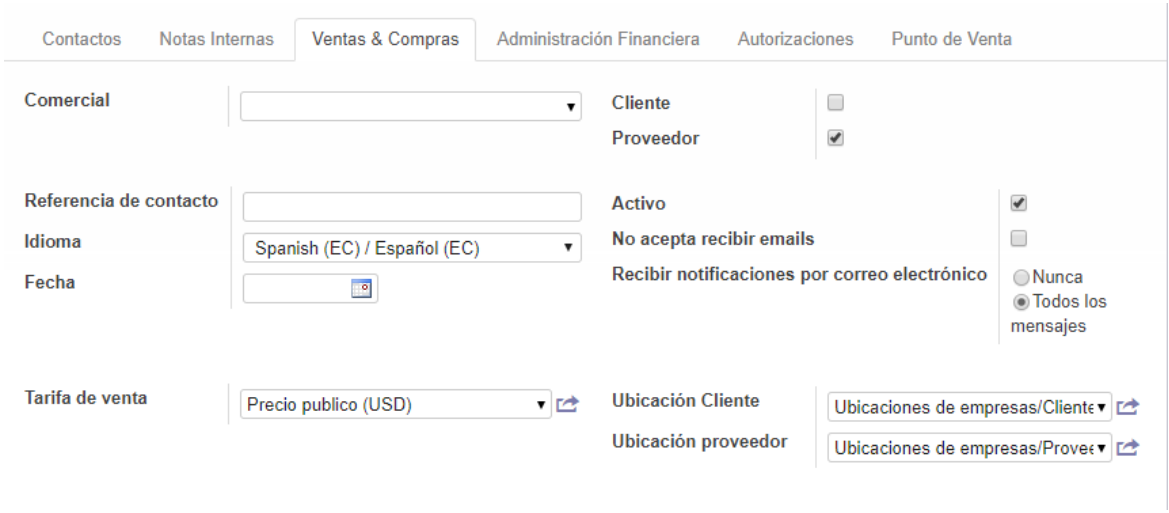
Dentro de esta ventana hay varias pestañas donde configurar el proveedor y añadir la información oportuna. Al definir el proveedor como una empresa tenemos la pestaña **Contactos** para añadir a las personas físicas vinculadas a dicha empresa. Como ejemplo, incluimos a uno de los comerciales de la empresa por ser la persona con la habitualmente nos comunicamos a la hora de adquirir sus productos.



La siguiente pestaña es la de **Notas internas**, para que los usuarios del sistema puedan hacer anotaciones sobre el proveedor. Llegamos a **Ventas y compras** donde podemos rellenar los siguientes campos:

- Comercial: es el usuario encargado de comunicarse con este proveedor. En este caso se lo podemos asignar al Vendedor 1.
- Referencia del contacto: se trata del código interno utilizado por la empresa para identificar al proveedor. Le asignamos la referencia P000001.
- Idioma y fecha: se trata del idioma utilizado en todos los documentos asociados a este proveedor y de la fecha en la que es registrado o en la que la empresa empieza a trabajar con él.
- No acepta mensajes: al seleccionar esta opción, dejamos de enviarle correos electrónicos masivos cuando hay campañas de marketing o publicidad.
- Notificaciones por email: se pueden recibir o no correos electrónicos por parte del proveedor o relacionados con él en la bandeja de entrada del usuario.

En esta pestaña podemos marcar las opciones de Cliente o Proveedor. Si en lugar de crear un proveedor como estamos haciendo, queremos generar un **cliente**, simplemente hay que seleccionar la opción correspondiente.



Productos

Después de introducir un proveedor o cliente, ya podemos crear productos para su compra-venta. Anteriormente comentamos que esta empresa se dedica a la venta al por menor de pescados y mariscos. Estos productos del mar no son procesados, no se realiza ninguna manipulación sobre ellos que modifique su estado natural, por tanto, actúan a la vez de materias primas y de productos vendidos directamente al consumidor. Dicho esto, pasamos a ver cómo se crean las categorías de un producto y a continuación, cómo generar el propio producto en el sistema ERP Odoo.

Categorías de productos

Esta función del sistema ERP nos permite clasificar los productos según sus características o los criterios que la empresa considere adecuados. Entramos dentro del sistema como Administrador y nos situamos en el módulo **Almacén**, después en el menú izquierdo donde está el apartado **Configuración**, seleccionamos **Productos y Categorías de Productos**. En la pantalla aparecerá una lista de las categorías creadas por defecto en el sistema Odoo.

Nuestro objetivo es ver cómo se crea una nueva categoría, y que mejor manera de hacerlo que generando una que vayamos a utilizar posteriormente como la de **Existencias**. Emplearemos este tipo para agrupar en él a los productos de nuestra empresa que se

encuentran en stock. Damos a **Crear** y se abre una ventana donde introducir los datos que se piden.

- Datos generales:
    - ✓ Nombre: damos un nombre a la categoría (Existencias)
    - ✓ Categoría padre: el nombre de la categoría a la que pertenece (Todos).
    - ✓ Tipo de categoría: dejamos la opción Normal por defecto.

- Rutas: permite definir rutas complejas dentro del almacén donde se encuentran los productos pertenecientes a esta categoría. Se basa en las reglas push y pull para gestionar los movimientos de productos. Este apartado lo dejamos como está.

- Propiedades de la cuenta: presenta dos cuentas contables distintas:
    - ✓ **Cuenta de ingresos**: sirve para valorar el stock saliente según el precio de venta impuesto.
    - ✓ **Cuentas de gastos**: sirve para valorar el stock entrante según el precio de coste del producto.
- Propiedades de la cuenta de existencias: se trata de las cuentas que registran el movimiento y valor de las existencias de los almacenes de la empresa. Se utilizan cuando se selecciona un método de valoración de existencias. Aunque en nuestro caso no será necesario, vemos que datos se pueden configurar:

    - ✓ **Entrada de existencias**: cuenta que registra el valor del stock entrante según el precio de coste del producto.
    - ✓ **Salida de existencias**: se trata de una cuenta para registrar, al cierre de ejercicio, las variaciones entre las existencias finales y las iniciales. Se identifica con los dígitos 610 Variación de existencias de mercaderías.
    - ✓ **Valoración de existencias**: es la cuenta que recoge las existencias adquiridas por la empresa destinadas a la venta.
    - ✓ **Diario de existencias**: es el diario contable donde se crean los asientos automáticamente cuando se procesan los movimientos de existencias. Viene ya definido por el sistema como Stock Journal. También es posible crear un diario nuevo si fuese necesario.

Para terminar, pinchamos en **Guardar** y así tenemos creada esta nueva categoría de productos.

❖ Creación de productos

Para crear un producto accedemos al sistema como el usuario Administrador o como otro que tenga los permisos y accesos correspondientes para llevar a cabo esta acción. Una vez dentro, nos situamos en el menú superior de la pantalla y podemos elegir entre cuatro vías distintas para generar nuestros productos; a través de Ventas, Terminal punto de venta, Compras (o Purchases) y Almacén. Realmente es indiferente desde donde creamos los productos. En este caso, lo haremos desde el módulo de gestión de almacenes llamado Almacén. Lo siguiente es buscar el apartado **Productos** y la opción Productos, damos a **Crear** y añadimos un producto perteneciente a la empresa. Como ejemplo, creamos el producto Salmón especificando ciertos atributos y dando toda la información necesaria sobre él.

Aparecen varias pestañas para configurar el producto, en la primera llamada **Información**, introducimos datos generales del producto. Para que aparezca en los pedidos de compra y venta es necesario marcar las opciones de Puede ser comprado o vendido respectivamente. El resto de campos son los siguientes:

- Tipo de producto: este campo es obligatorio y existen tres clases de productos según su naturaleza:

- **Almacenables**: son aquellos que se guardan en los almacenes de la empresa, se lleva una reposición de su stock automatizada, se pueden definir reglas de abastecimiento para ello y existe un control del stock. Nuestro producto pertenece a esta clase.

- **Consumibles**: son los productos que se recepcionan, se consumen o se fabrican. El sistema no permite realizar una reposición del stock automática, ni configurar la valoración de inventarios, ni tampoco se generan asientos contables asociados a los movimientos de inventario de dicho producto.
- **Servicios**: no requieren ningún control del nivel de inventarios. Se pueden comprar mediante la subcontratación de servicios o fabricar al incorporar tareas a un proyecto.

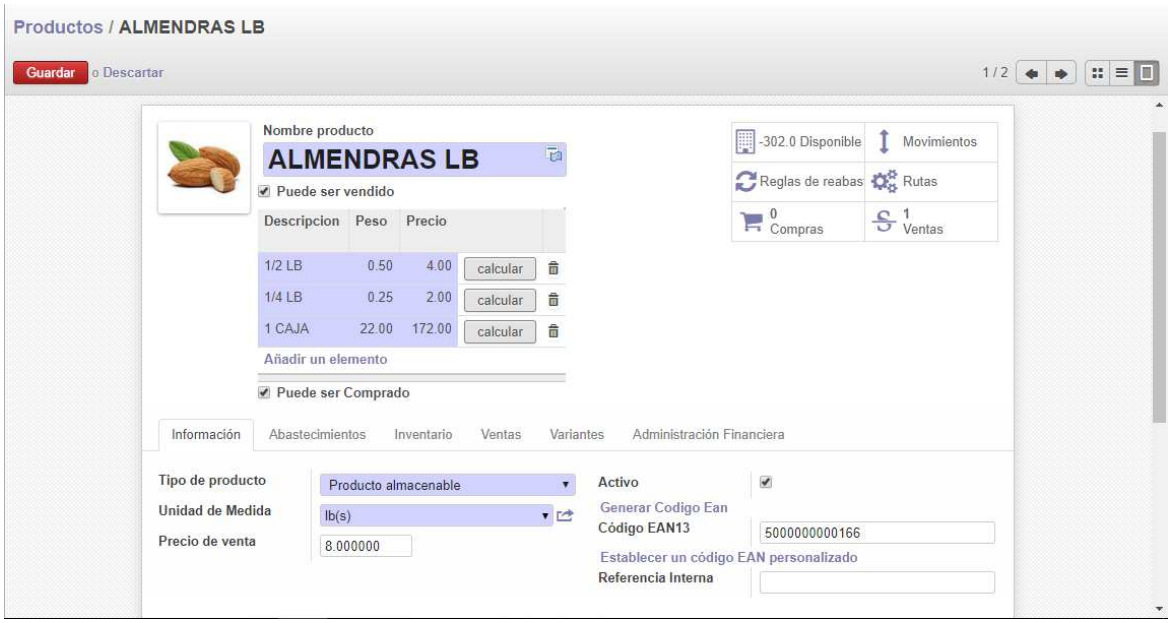
- Unidad de medida: se trata de otro campo obligatorio, es la medida utilizada por defecto en las operaciones de stock. Se puede elegir entre multitud de ellas o incluso crearla. Este producto se mide en kilogramos (peso).

- Precio de venta: precio que asumen los clientes. Rellenar este campo es opcional, y además hay que tener en cuenta que los precios de venta varían con los de compra casi diariamente en el caso de nuestra empresa. Se puede poner un precio base de referencia como por ejemplo, 8 dólares la libra.

- Código EAN13: se puede asignar o generar un código de barras para tener un registro del producto. Se utiliza sobre todo cuando la empresa cuenta con sistema de escaneado, concretamente para el TPV.

- Referencia interna: se trata de un código único, propio de la empresa, que se asigna a un producto para facilitar su búsqueda o identificación. Su introducción es opcional.

- Información interna del producto: son comentarios o avisos sobre el producto, útiles para los usuarios del sistema ERP.



En la siguiente pestaña de **Abastecimientos,** podemos seleccionar o modificar aspectos como:

- Precio de coste: precio base en los pedidos de compra y también utilizado en la valoración de existencias. Al igual que el de venta, es opcional. En este negocio del comercio minorista, los proveedores modifican los precios habitualmente. Podemos introducir un valor como ejemplo de 8 dólares por libra.

- Unidad de medida de compras: unidad por defecto utilizada en los pedidos de compra. Es el kilogramo (peso).

- Rutas del producto: se refiere al método de suministro de los productos.

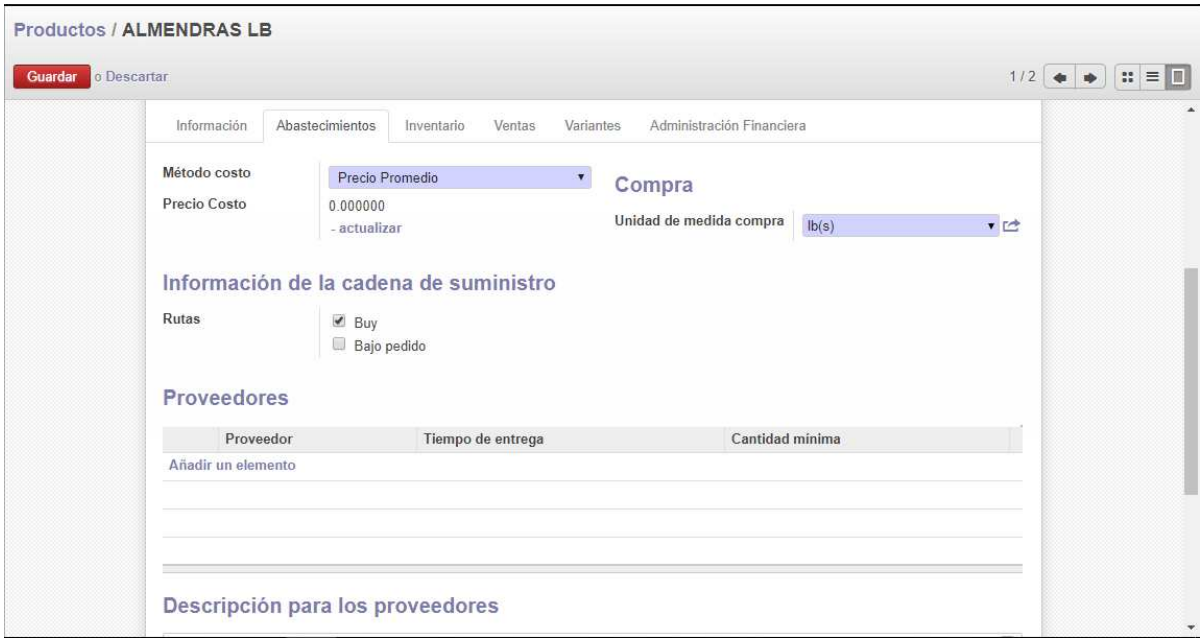
  Tenemos dos opciones **Buy** (comprar) y **Make to order** (orden de pedido), ambas se basan en el abastecimiento de productos a través de pedidos de compra a proveedores. La diferencia reside en que la segunda opción supone la creación de un borrador de orden de compra cuando se produce una orden de venta de un producto sin tener en cuenta la cantidad de existencias de dicho producto. En nuestro caso, seleccionamos la primera opción.

- Proveedores: se pueden asignar uno o varios proveedores. En este caso, la empresa encargada de proporcionarnos el producto.

- Nombre y Código producto proveedor: se trata del nombre y del código con los que el proveedor identifica el producto. Pueden ser distintos a los que emplea la empresa, si coincidieran se dejan estos campos en blanco. Estas identificaciones aparecen en las solicitudes de presupuesto y en las órdenes de pedidos de compra.

- Cantidad mínima: es el número mínimo de productos que se pueden pedir al proveedor.
- Tiempo de entrega: es el tiempo que transcurre desde que se realiza el pedido al proveedor hasta que la empresa lo recibe.

- Descripción para los proveedores: permite crear una nota o aviso acerca del producto para los proveedores. Aparece en los pedidos, recepciones y facturas.



Continuamos con Inventario y configuramos los parámetros que hagan falta. Tenemos varios grupos:

- Nivel de existencias y variaciones: nos permite establecer la cantidad de producto que hay en un almacén determinado (**Cantidad a mano**), se suele introducir el nivel de existencias iniciales, aunque también sirve para actualizarlo en un momento concreto. Otra opción es realizar un pedido de compra en **Solicitar abastecimiento** e introducir la cantidad necesaria. La **Cantidad disponible** nos informa del nivel de inventario real, excluyendo aquellos productos que permanecen en el almacén hasta su entrega final o aquellos que se han comprado, pero aún no se han recibido.

- Estado y ubicación de las existencias: el estado del producto nos permite decir si puede ser usado o no. También se le asigna un responsable y podemos definir la ubicación exacta dentro del almacén (estante, fila, caja). Estos campos no son estrictamente necesarios, dependen de los datos que maneje la empresa.

- Propiedades de las ubicaciones parte recíproca: se refiere a las ubicaciones de abastecimiento, inventario y producción. Sólo nos interesan los dos primeros que indican el lugar de origen para los movimientos de stock generados por los abastecimientos y por los inventarios respectivamente. Por defecto aparecen ya definidas las ubicaciones.

- Pesos: se refiere al volumen (metros cúbicos), peso bruto (kg) y peso neto (kg) del producto.

- Fechas: son una serie de fechas y periodos que pueden ser importantes para la gestión del producto:

- **Tiempo de vida**: periodo transcurrido hasta que el producto deja de ser consumible y resulta peligroso.

- **Duración del producto**: tiempo que transcurre hasta que el producto se deteriora sin ser peligroso para el consumo.
    - **Tiempo de eliminación**: tiempo que pasa hasta que los bienes se eliminan del stock.
    - **Tiempo de alerta**: periodo que transcurre hasta notificarse la alerta

Pasamos a ver la pestaña de **Ventas,** donde tratamos información relacionada con el proceso de ventas del producto:

Terminal punto de venta: son opciones especificas del módulo TPV instalado que podemos seleccionar como:

- **Categoría del TPV**: se pueden definir categorías en el módulo TPV para agrupar productos similares. Cuando creamos dichas categorías en el TPV se seleccionan en este campo.
- **Disponible en el TPV**: hace que el producto aparezca en el TPV. Si por alguna razón se agota un producto en un determinado momento, basta con no seleccionar esta opción y no aparecerá para su venta.
- **Otras opciones**: si tenemos instalado el hardware necesario podemos realizar operaciones como pesar el producto en una balanza electrónica conectada e introducir o sacar dinero en efectivo a través de un cajón de monedas integrado durante su venta.

Llegamos al apartado de **Contabilidad** de este producto. En él vemos los siguientes campos:

- Categoría interna: es el grupo contable al que pertenece el producto. Por defecto aparece la categoría Todos pero se puede cambiar e incluso crear una nueva. Vamos a asignar a este producto la categoría **Existencias** (creada en el punto anterior), de este modo, el producto toma las cuentas de ingresos, gastos y

existencias de la categoría a la que pertenece. Es una forma de agrupar los productos según las necesidades de la empresa.

- Cuenta de ingresos: es la cuenta contable que se emplea para valorar el stock saliente para la categoría de producto actual utilizando el precio de venta. Como ya hemos dicho, aunque se deje en blanco, se toman las cuentas de la categoría del producto.

- Impuestos del cliente: son los correspondientes a la venta del producto. Se trata del IVA repercutido para el cliente.

- Cuenta de gastos: es la cuenta contable que se usa para valorar el stock entrante para la categoría de producto actual utilizando el precio de coste. Este precio es el que aparece en la pestaña de Abastecimiento.

- Impuestos del proveedor: son los correspondientes a la compra del producto. Se refiere al IVA soportado por la empresa al adquirir los productos.

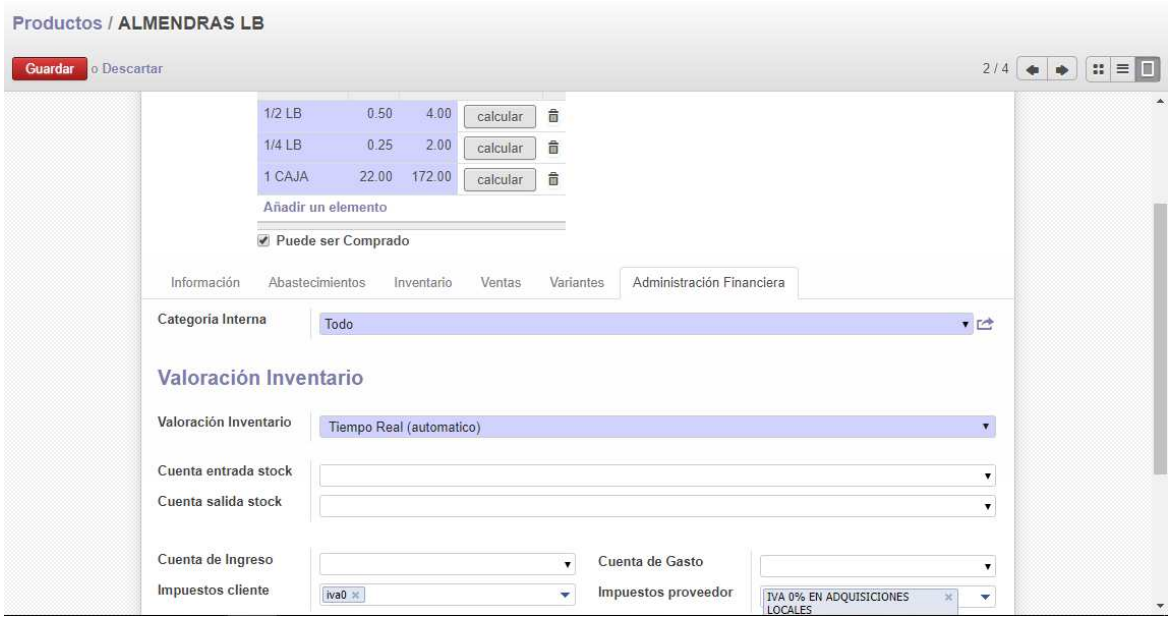
- Si en la configuración general del módulo de Gestión de almacenes decidimos marcar la opción de **Generar abastecimiento en tiempo real,** aparece el siguiente campo:

- Valoración de existencias: cuenta con dos métodos:

    - **Tiempo real (automatizado)**: la idea es registrar todas las variaciones en la valoración del inventario cada vez que hay algún movimiento en el almacén. El propio sistema genera automáticamente apuntes contables correspondientes a los movimientos de existencias del producto. Las cuentas que aparecen debajo se pueden dejar en blanco si son establecidas en la categoría del producto.

- **Periódico (manual)**: la valoración del inventario se hace forma periódica y manual al final del periodo. No hay un seguimiento diario de los movimientos de stock.

Volveremos a comprobar esta configuración sobre el funcionamiento del módulo de Contabilidad y finanzas.



La última pestaña es la de **Avisos** en la que se pueden configurar advertencias o notificaciones para los usuarios a la hora de vender o comprar el producto. De momento no vemos necesario incluir ningún comentario.

Para terminar, nos fijamos en un recuadro de la parte derecha de la ventana del producto que funciona como un menú con varias utilidades:

- Compras: nos permite ver la lista de pedidos de compra asociados al producto.
- Ventas: nos permite ver la lista de pedidos de venta realizados del producto.
- Disponible: nos da información sobre las existencias del producto en los almacenes de la empresa.
- Movimientos: muestra los movimientos de stock asociados al producto.
- Reglas de abastecimiento: permite establecer una serie de normas referidas al nivel de existencias del producto. Se puede introducir un rango de valores (cantidad máxima y mínima) para que el sistema genere un abastecimiento del producto si el nivel de stock excede el límite impuesto. Es muy útil para llevar un control del inventario.
- Rutas: nos enseña las rutas asociadas al producto. Se basan en reglas push y pull. Para este producto solo existen dos: Buy (comprar) y Make to order (orden de pedido).

CONTABILIDAD Y FINANZAS

El módulo de Contabilidad y finanzas, como su propio nombre indica, permite al usuario configurar y gestionar las actividades contables y financieras de la empresa. Al comienzo de su instalación, el sistema Odoo permite seleccionar el plan contable que la empresa desea aplicar. Por ello, se instalan los módulos de localización ecuatoriana que sirven para la gestión financiera y contable siguiendo la legislación ecuatoriana mediante el plan de contabilidad. La decisión de configurar primero la gestión de la contabilidad de la empresa

se toma porque este módulo interacciona en gran medida con todos los demás, y de esta manera será más fácil parametrizarlos.

Para acceder al módulo debemos entrar en el sistema con los permisos adecuados (es mejor hacerlo como Administrador) y situarnos en la opción **Contabilidad** del menú superior. De aquí en adelante vamos a explicar aquellas funciones y configuraciones del módulo que la empresa puede necesitar. Por tanto, los objetivos de este apartado son los siguientes:

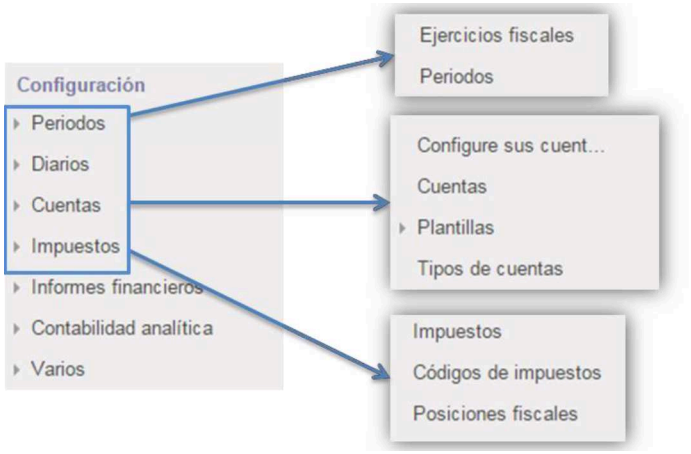
- Conocer ciertas funciones del módulo que pueden resultar útiles a la hora de manejar la contabilidad: crear un ejercicio fiscal, así como sus periodos, comprobar los diarios contables, las cuentas y los regímenes de impuestos establecidos por el sistema.

- Ver los planes y asientos contables generados automáticamente por el sistema, además de mostrar cómo crear asientos contables de forma manual.
- Establecer métodos de pago a través de la creación de diarios contables.

- Gestionar las facturas de proveedores, así como las facturas rectificativas correspondientes. Vemos cómo crear manualmente una factura, hacemos un seguimiento de su estado a lo largo del proceso y también realizamos un ejemplo de devolución o rectificación de una factura.

Configuración general: periodos, diarios, cuentas e impuestos

Para llevar la gestión de la contabilidad de cualquier empresa mediante el sistema Odoo, es preciso establecer ciertas bases. A través de la interfaz del sistema, comenzamos situándonos en el último apartado del menú izquierdo llamado **Configuración**. Dentro de él vamos a ver las pestañas: Periodos, Diarios, Cuentas e Impuestos, y sus desgloses:



En la pestaña **Periodos** encontramos información sobre los ejercicios fiscales de la empresa. Si accedemos a la opción **Ejercicios fiscales**, vemos que ya tenemos creado el ejercicio fiscal del año 2017 dividido en 12 periodos mensuales Para definir un nuevo ejercicio fiscal vamos a crear el del año 2016. Simplemente damos al botón **Crear** indicado y configuramos los datos:

- Ejercicio fiscal: el año al que corresponde, 2017.

- Código: se trata de una referencia para identificarlo que aparece en informes y documentos generados por el sistema. Se utiliza también el año del ejercicio.

- Fecha inicial: es la fecha de apertura del ejercicio. Normalmente suele ser el 1 de enero.
- Fecha final: es la fecha de cierre del ejercicio. Si son 12 meses desde el inicio, la fecha corresponde al 31 de diciembre del mismo año.

En función del nivel de control que se desea llevar sobre las actividades de la empresa, seleccionamos la creación de 12 periodos mensuales o 4 periodos trimestrales. En este caso elegimos periodos mensuales como los del año 2017.

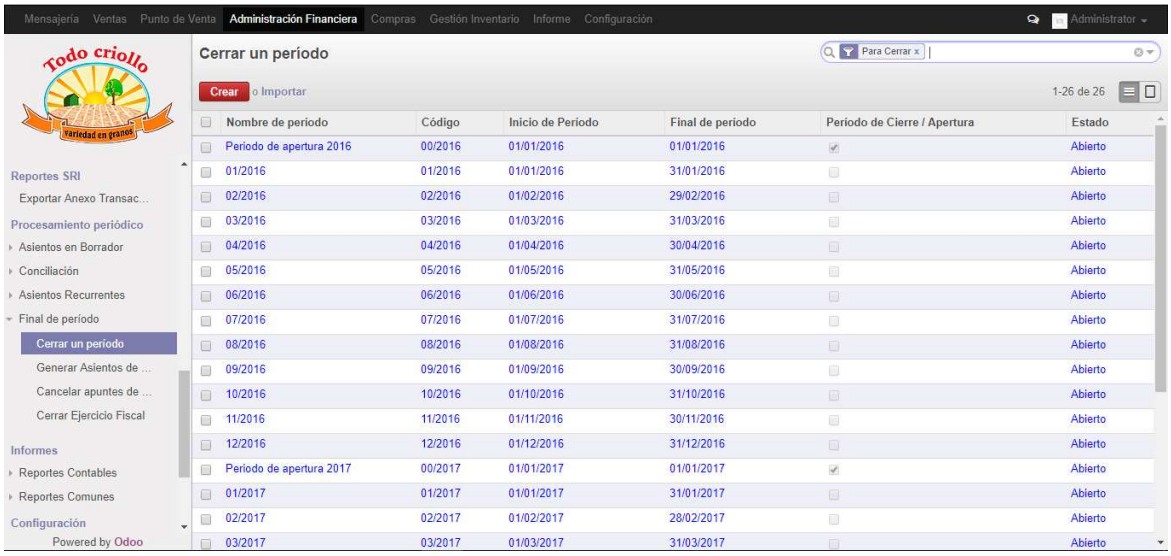
Dentro de la misma pestaña, podemos acceder a **Periodos** para comprobar todos los periodos fiscales correspondientes a los ejercicios abiertos. En este caso, habrá 24 periodos abiertos. Si

seleccionamos uno de ellos vemos que contiene la siguiente información: nombre, código, ejercicio fiscal al que pertenece, duración y una opción llamada Periodo de apertura/cierre que crea un periodo especial que se puede solapar con otro como ocurre en la apertura o cierre del ejercicio (el asiento de cierre del año 2016 es el asiento de apertura del año 2017).

El **cierre** de un **ejercicio fiscal o** de sus periodos se realiza yendo al apartado Procesamiento periódico del menú izquierdo y pinchando en la pestaña **Fin de periodo**. Se despliegan varias opciones, entre ellas:

- **Cerrar periodos**: una vez cerrados, no permiten la creación de asientos en ellos. Se recomienda cerrar un período una vez la empresa ha calculado y liquidado sus impuestos.

- **Cerrar un ejercicio fiscal**: al final del año se procede al cierre del ejercicio, tras el cual no se pueden crear ni modificar operaciones financieras de ese mismo año. Se abre una ventana para poder cerrar el año indicado.



Los diarios de contabilidad de Odoo son donde se registran todos los asientos contables. El sistema posee un conjunto de diarios por defecto en función de los asientos que van a contener. Podemos verlos entrando en la pestaña **Diarios** del menú izquierdo donde aparece una lista con todos ellos. Los que más vamos a utilizar son:

- **Diario de compras**: sirve para registrar los asientos asociados a las compras de la empresa, es decir, a las facturas de proveedores.

- **Diario de ventas**: se utiliza para registrar asientos asociados a las ventas, o lo que es lo mismo, a las facturas de clientes. También registra las ventas de los distintos puntos de venta (TPV).

- **Efectivo**: registra todos los pagos a proveedores y cobros a clientes que se hacen con dinero en efectivo.

- **Banco**: este diario está vinculado a una caja o banco con el que trabaja la empresa. Registra los pagos y cobros que se realizan a través del banco sin necesidad de tener un nº de cuenta asignado o una entidad bancaria concreta. Se puede crear otro diario asociado a una cuenta específica.

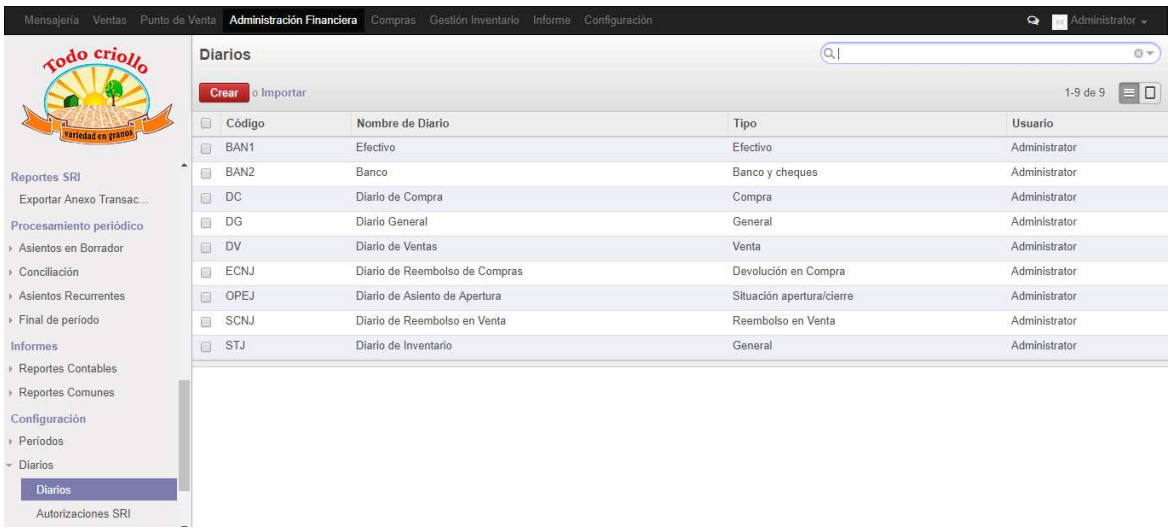
- **Diario de abono de compras**: sirve para registrar todos los asientos asociados a facturas rectificativas o devoluciones de proveedores.

- **Stock Journal**: sirve para registrar el valor de las existencias. Cuando hay un movimiento de inventario dentro del almacén (entrada o salida), el sistema puede generar un asiento que pertenece a este diario.

Se puede configurar y personalizar el código y la secuencia de los elementos generados en cada diario para facilitar la consulta de las actividades contables de forma separada. En el momento en el que instalemos el sistema en la propia empresa y configuremos los módulos, como crear nuevos identificadores según las preferencias de la empresa.

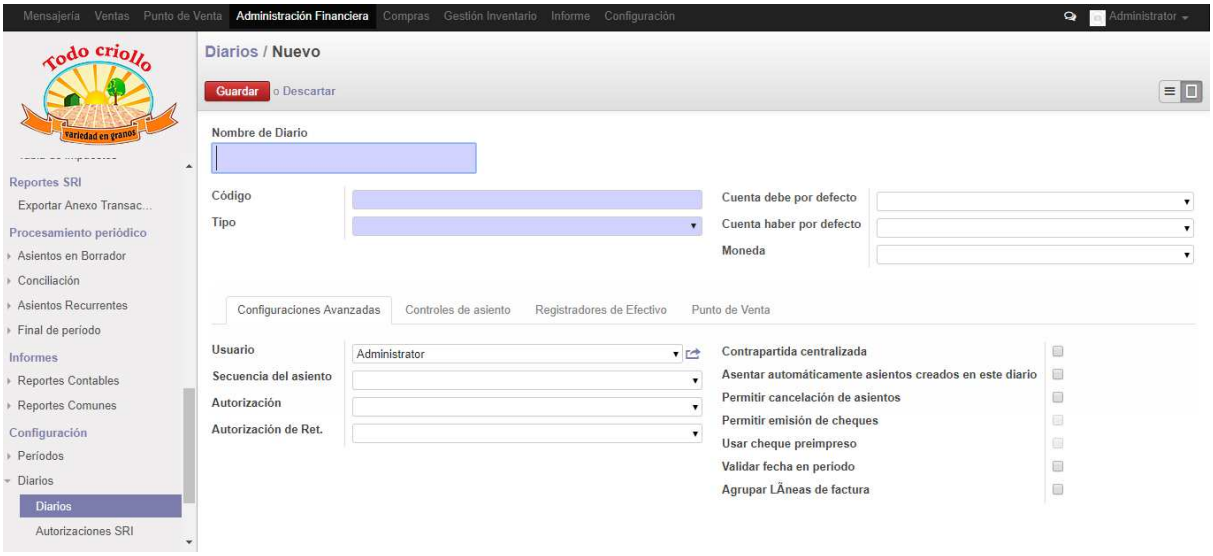
Puede ser útil para el negocio saber cómo **crear un nuevo diario** en Odoo. Supongamos que por determinados motivos la empresa quiere registrar las **facturas de un proveedor** en un diario concreto para él. La creación del „Diario de compras (Nombre del proveedor)" se inicia dando al botón **Crear** situado en la ventana donde aparece la lista de diarios del sistema, seguidamente se abre una ventana donde debemos introducir los datos básicos del nuevo diario

- Nombre del diario: le damos el nombre de „Diario de compras (Todo Criollo) por estar vinculado a dicho proveedor.
    - Código: asignamos una referencia.
    - Tipo: según la actividad que registra es de tipo Compra.
    - Cuentas deudora y acreedora por defecto: son las cuentas que recogen todas las actividades de compra asociadas a este diario.

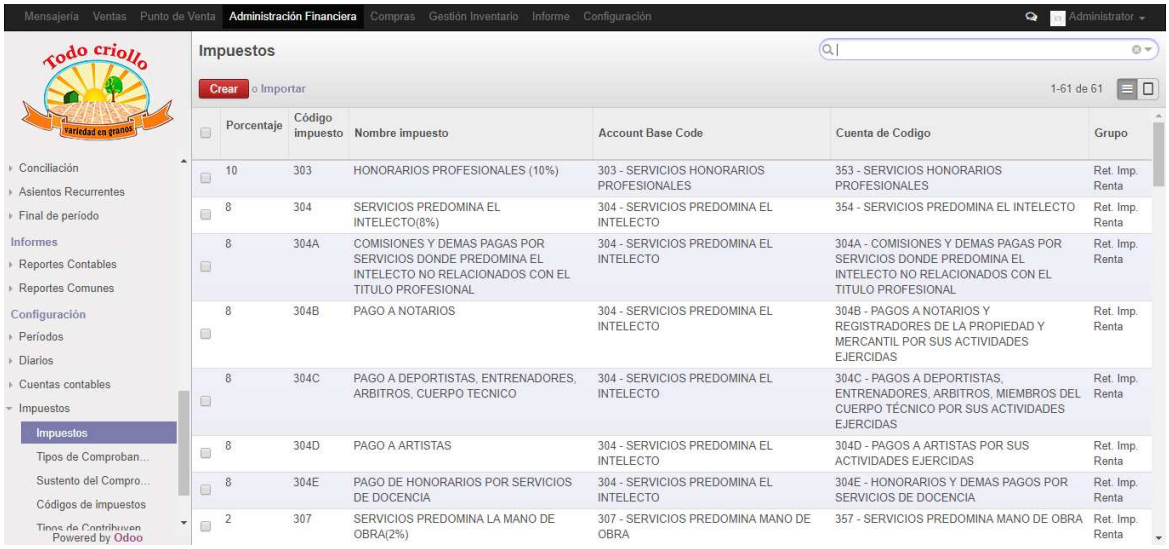


Además de toda esta información, existen una serie de funciones:

- **Permitir cancelación de asientos**: si se encuentra instalado su módulo correspondiente, marcando esta opción podemos eliminar asientos del diario.
- **Agrupar líneas de factura**: sirve para agrupar las líneas del asiento cuando son generadas desde las facturas.



En cuanto a los impuestos, el sistema configura por defecto un conjunto de ellos, sus códigos y una serie de posiciones fiscales debido a la instalación de los módulos de contabilidad y localización española. Dado que es una empresa nacional, se pueden aplicar a sus actividades todos los impuestos que aparecen en Odoo. Para acceder a ellos vamos al apartado **Configuración** y a la pestaña **Impuestos**. De la lista que aparece volvemos a seleccionar Impuestos para revisar aquellos que vayamos a utilizar habitualmente o crearlos, en caso de que aún no existan. Como ya vimos en la creación y configuración de los proveedores, las posiciones fiscales se aplican sólo en determinados casos. Yendo a la pestaña **Posiciones fiscales** observamos que el sistema ha creado algunas por defecto. comprobamos los impuestos que se utilizan cuando abrimos el Recargo de Equivalencia, posición utilizada cuando se emiten facturas de proveedores a la



Porcentaje	Código impuesto	Nombre impuesto	Account Base Code	Cuenta de Codigo	Grupo
10	303	HONORARIOS PROFESIONALES (10%)	303 - SERVICIOS HONORARIOS PROFESIONALES	353 - SERVICIOS HONORARIOS PROFESIONALES	Ret. Imp. Renta
8	304	SERVICIOS PREDOMINA EL INTELECTO(8%)	304 - SERVICIOS PREDOMINA EL INTELECTO	354 - SERVICIOS PREDOMINA EL INTELECTO	Ret. Imp. Renta
8	304A	COMISIONES Y DEMAS PAGAS POR SERVICIOS DONDE PREDOMINA EL INTELECTO NO RELACIONADOS CON EL TITULO PROFESIONAL	304 - SERVICIOS PREDOMINA EL INTELECTO	304A - COMISIONES Y DEMAS PAGAS POR SERVICIOS DONDE PREDOMINA EL INTELECTO NO RELACIONADOS CON EL TITULO PROFESIONAL	Ret. Imp. Renta
8	304B	PAGO A NOTARIOS	304 - SERVICIOS PREDOMINA EL INTELECTO	304B - PAGOS A NOTARIOS Y REGISTRADORES DE LA PROPIEDAD Y MERCANTIL POR SUS ACTIVIDADES EJERCIDAS	Ret. Imp. Renta
8	304C	PAGO A DEPORTISTAS, ENTRENADORES, ARBITROS, CUERPO TECNICO	304 - SERVICIOS PREDOMINA EL INTELECTO	304C - PAGOS A DEPORTISTAS, ENTRENADORES, ARBITROS, MIEMBROS DEL CUERPO TÉCNICO POR SUS ACTIVIDADES EJERCIDAS	Ret. Imp. Renta
8	304D	PAGO A ARTISTAS	304 - SERVICIOS PREDOMINA EL INTELECTO	304D - PAGOS A ARTISTAS POR SUS ACTIVIDADES EJERCIDAS	Ret. Imp. Renta
8	304E	PAGO DE HONORARIOS POR SERVICIOS DE DOCENCIA	304 - SERVICIOS PREDOMINA EL INTELECTO	304E - HONORARIOS Y DEMAS PAGOS POR SERVICIOS DE DOCENCIA	Ret. Imp. Renta
2	307	SERVICIOS PREDOMINA LA MANO DE OBRA(2%)	307 - SERVICIOS PREDOMINA MANO DE OBRA	357 - SERVICIOS PREDOMINA MANO DE OBRA	Ret. Imp. Renta

empresa modificando así el impuesto habitual. Se puede aplicar este régimen de forma automática y aplicar solamente en caso de que el proveedor tenga el NIF. Como podemos ver (indicado en color azul), aplicando dicha posición fiscal, el impuesto de origen corresponde al mismo impuesto del 12% de IVA.

GESTIÓN DE COMPRAS

Este segundo módulo que instalamos sirve para gestionar todo el proceso de compra de los productos necesarios para que la empresa lleve a cabo sus actividades. Dicho proceso consta de varias etapas.

Anteriormente configuramos los parámetros generales del módulo según las necesidades de la empresa y ahora pasamos a ver cómo se maneja y de qué manera somos capaces de gestionar las compras gracias a sus funcionalidades.

Para acceder a esta aplicación, vamos a **Compras** (Purchases) en menú de la parte superior de la pantalla. Según los requerimientos de la empresa, las acciones que vamos a llevar a cabo son las siguientes:

- Crear una orden de compra para un proveedor de la empresa. La solicitud de presupuesto a un proveedor generada como solicitud de compra puede hacerse manualmente o provenir de una orden de abastecimiento del almacén situado en una de las tiendas.

- Llevar un control del estado del pedido de compra pues sabemos que el proceso termina con la recepción y pago de los productos adquiridos. Veremos como este módulo permite hacer un seguimiento del proceso.

- Comprobar el método de generación de facturas que utiliza el sistema. Nos permite llevar un control sobre la facturación a los proveedores.

Creación de órdenes de compra

Sabemos que el sistema ERP Odoo gestiona automáticamente todos los procesos asociados a las órdenes de compra. La creación de una **orden de compra** para cierto

producto q u e necesita la empresa se consigue generando una solicitud de presupuesto a un proveedor. Existen varias maneras:

- Manualmente a través de las pestañas del menú izquierdo llamadas **Pedidos de compra** y **Solicitudes de presupuesto**. En la primera aparece una lista con todas las órdenes de compra realizadas; y la segunda pestaña, contiene la lista de los presupuestos solicitados a los proveedores que están por confirmar.

- Automáticamente mediante una orden de abastecimiento procedente de algún almacén situado en una de las tiendas que requiera aumentar su nivel de stocks o cuyos productos estén agotados.

Si la empresa desea pedir un **presupuesto** a un proveedor de forma manual (ya sea accediendo a una pestaña u otra) tenemos que configurar los siguientes campos:

- Nombre del proveedor: si tenemos creado el proveedor en el sistema podemos seleccionarlo directamente de la lista.

- Referencia del proveedor: se trata de un código asociado al pedido asignado por el proveedor. Es posible que las referencias de la empresa sean distintas a las del proveedor. Este código sirve para casar el pedido y la factura generada con el albarán del proveedor en la recepción de dicho pedido.

- Fecha de pedido: es la fecha en la que el presupuesto se confirma y se convierte en un orden de compra. Por defecto aparece la fecha actual.

- Lugar de entrega: podemos elegir la ubicación y el almacén donde se reciben los productos. La empresa cuenta con 4 tiendas y podemos elegir el almacén situado en una de ellas, apareciendo indicado en el albarán correspondiente a la entrega de

los productos. En el siguiente apartado vemos como se crean los distintos almacenes situados en las tiendas.

Dentro de la ventana podemos acceder a tres pestañas. La primera de ellas es la de **Productos** donde se añaden los productos que se pretenden comprar al proveedor. En el momento en el que introducimos un producto de la empresa se autocompletan los demás campos a excepción de la Cantidad que introducimos manualmente.

La siguiente pestaña es la de **SdP** (Solicitud de presupuesto) **y licitación**. En nuestro caso, la empresa no realiza ningún proceso de licitación con los proveedores por tanto no será necesario introducir los datos que pide.

Por último, entramos en la pestaña de **Entregas y facturas** donde se pide información sobre la entrega de los productos y el modo de facturación. Los campos que debemos rellenar son los siguientes:

- Fecha prevista: es la fecha mínima de entrega planificada para todos los productos de la orden de pedido. Tiene en cuenta el plazo de entrega de compra establecido en la configuración de los datos de la compañía.

- Destino: es la ubicación física a la que el proveedor debe entregar el pedido de compra. Para la empresa, hay 4 posibilidades, los almacenes de cada una de las tiendas.

- Método de facturación: existen tres formas de generar las facturas y en función de la que escojamos tendremos que consultar una de las pestañas del apartado Control de facturas situado en el menú izquierdo. Los métodos son:

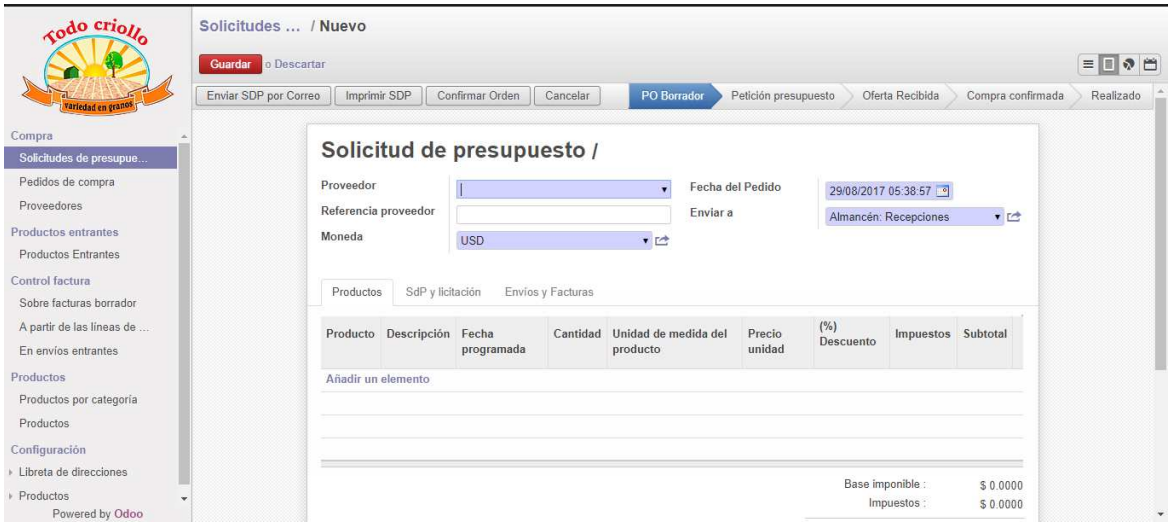
    - Manual, basado en las **líneas de pedidos de compra**. Es preciso ir a la pestaña A partir de las líneas de pedido para poder crear las facturas correspondientes.

- Automático, el sistema crea una **factura borrador** en el momento en el que se confirma el pedido de compra. En la pestaña Sobre facturas borrador se pueden consultar esta clase de facturas que se validan más tarde.

- Al llegar el pedido de compra al almacén de la empresa, es decir, basado en las **recepciones**. En la pestaña En envíos entrantes se pueden crear facturas de proveedor según se reciban las mercancías.

- Plazo de pago del proveedor: se puede establecer el margen de tiempo que tenemos para pagar al proveedor según las condiciones del contrato.

- Posición fiscal: se establece cuando el pedido de compra está sujeto a algún tipo de posición fiscal.



GESTIÓN DE ALMACENES

Se trata de un módulo estrechamente relacionado con el de compras, ventas y TPV de forma que nos permite administrar y controlar la cadena de suministro de nuestra empresa. Como ya describimos anteriormente, los inventarios son gestionados bajo un sistema de doble entrada basado en que cada movimiento de existencias tiene un origen y un destino.

Esto quiere decir que los productos se mueven desde el almacén del proveedor hasta nuestros propios almacenes, o desde nuestros almacenes hasta los del cliente, creando inicialmente un movimiento con origen en el almacén del proveedor para luego generar un movimiento de entrada a nuestros almacenes, y lo mismo con los clientes.

Para comenzar, accedemos al sistema y nos situamos en la opción **Almacén** situada en el menú superior de la pantalla, vamos a realizar una serie de tareas para conocer y configurar este módulo:

- Crear los almacenes así como las ubicaciones correspondientes asociadas a las tiendas. Mostramos como crear los almacenes ubicados en los establecimientos.

- Comprobar las funciones del módulo como los ajustes de inventarios y movimientos y la trazabilidad de los productos. Para ello, realizamos ejemplos como: introducir las cantidades iniciales de stock de los productos y verificar los movimientos de existencias generados al hacer un pedido de compra o al realizar una venta desde el TPV situado en una de la tiendas.

- Establecer reglas de abastecimiento, elegir los niveles mínimos y máximos de stock para los productos que lo requieran y ver cómo se ejecutan las acciones correspondientes.

TERMINAL PUNTO DE VENTA

Este módulo resulta imprescindible para la gestión del negocio en estudio debido a su forma de trabajar. Al fin y al cabo, la empresa está constituida por varias tiendas y necesitamos gestionar sus actividades de venta diaria al público de forma rápida y sencilla. El Terminal punto de venta cumple las funciones que necesita la empresa en este sentido:

- En cada tienda se puede acceder al terminal correspondiente para realizar las ventas de cada jornada de trabajo.

- Utiliza como interfaz cualquier ordenador de sobremesa, portátil, tabletas, móviles e incluso máquina TPV si se dispone de ellas. La empresa cuenta con ordenadores en cada tienda.

- Se puede utilizar a través de un navegador web, conectado en línea con el servidor de Odoo. Aunque se pierda la conexión a Internet el TPV sigue funcionando y restablece los datos cuando vuelve la conexión.

- Permite actualizar el nivel de inventario de los productos a tiempo real debido a su integración con el módulo de Almacenes.

- Las actividades que lleva a cabo generan facturas de ventas y llevan asociado un asiento contable.

- Funciona con distintos métodos de pago: efectivo, venta a crédito. Usando el efectivo, simplemente se introduce la cantidad abonada por el cliente a través de la interfaz (teclado), calcula el cambio y confirma la venta generando una factura.

- Su manejo resulta sencillo e intuitivo pues los productos disponibles que hay en el almacén aparecen en la pantalla con su imagen y precio correspondiente, ordenados por categorías. La búsqueda de cualquiera de ellos es fácil y rápida.

Vamos a ver cómo utilizar esta funcionalidad del sistema Odoo realizando una serie de pasos. Primero entramos en el sistema como Administrador o como usuario con permisos, de la forma que hemos venido haciendo. Situados en el menú superior de la pantalla, damos a **Terminal Punto de Venta**. Una vez dentro, podemos ver en el menú izquierdo todos los apartados que incluye:

- Creación y configuración de los terminales de venta necesarios para la empresa. De momento creamos uno a mayores del que ya existe por defecto y configuramos los parámetros que contiene.

- Creación de categorías de los productos para clasificarlos según sus características facilitando su búsqueda en el sistema.

- Explicar el funcionamiento básico del módulo en una de las tiendas, es decir, ver cómo se registran las ventas a clientes que entran en el establecimiento durante una jornada a través del interfaz TPV. ´

Categorías de productos para el TPV

Para examinar los productos vendidos en los establecimientos a través de la interfaz de la pantalla, vamos a ver cómo se crean sus distintas categorías. En primer lugar, nos situamos en el apartado **Productos** del menú izquierdo y pinchamos en **Categorías de productos para el TPV**. A continuación damos al botón **Crear** e introducimos el nombre de la categoría que deseamos crear (por ejemplo GRANOS), el nombre de la categoría padre si pertenece a alguna, el orden de secuencia en el que aparece si se muestra en una lista (dejamos la secuencia por defecto) y una imagen o icono del grupo. Pinchamos en **Guardar** y se crea esta nueva categoría de productos.

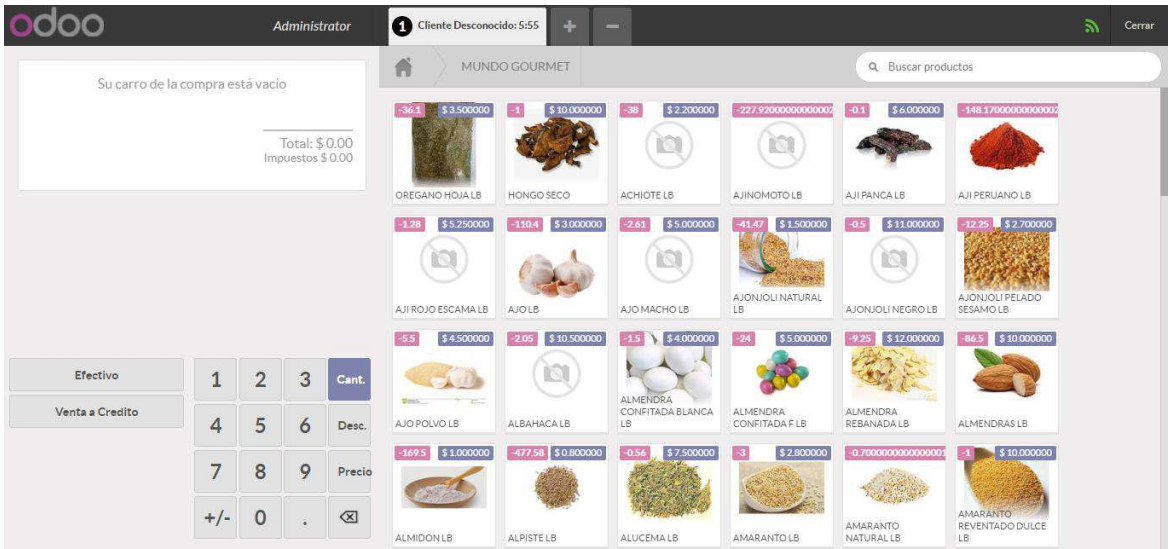
Realizamos el mismo proceso para crear cualquier otra categoría y vemos como nos aparece un listado de ellas al ir de nuevo a la opción Categorías de productos para el TPV. Una vez hecho esto, clasificamos cada producto en función de la categoría a la que pertenezca. Situándonos en la pestaña Productos, modificamos su configuración de Ventas seleccionando la Categoría del TPV.

Funcionamiento del TPV

En este punto vamos a ver cómo es el funcionamiento del TPV aplicado a la empresa. Para comenzar vamos al apartado **Operaciones diarias** del menú izquierdo e iniciamos la sesión de ventas del día en **Su sesión**. En la pantalla nos permite elegir desde que terminal de venta o tienda nos conectamos.

Pinchamos en **Nueva sesión** y pasamos a la ventana de control de efectivo o caja donde contabilizamos las monedas y billetes que contiene la caja registradora en la fecha actual. El dinero en efectivo que aparece al abrir la sesión es el mismo con el que se cierra la anterior sesión. vemos el saldo de apertura. Damos al botón **Guardar** y a **Validar y abrir sesión**.

Para comenzar el sistema nos lleva a una nueva interfaz desde donde el usuario puede gestionar las ventas de la tienda. A partir de aquí, explicamos parte por parte lo que nos muestra dicha interfaz.



En la zona superior hay una barra/menú donde aparece:

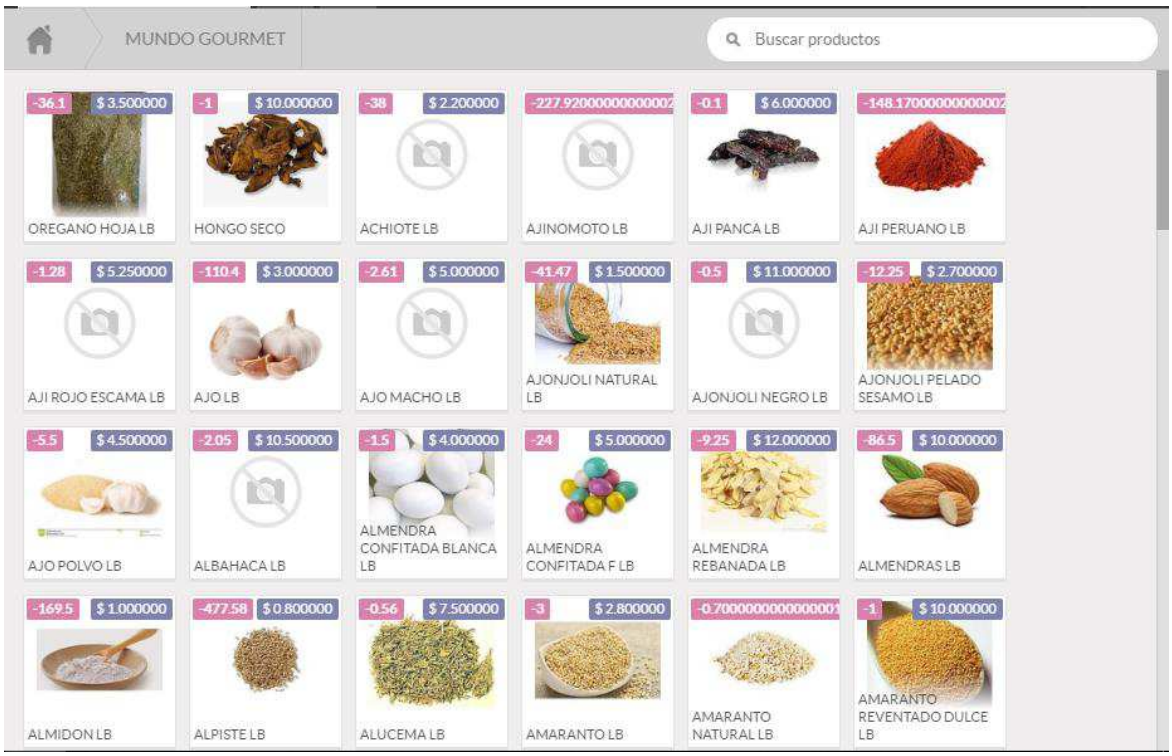
- El **usuario** que ha iniciado la sesión (Administrador).

- Los tickets de los **clientes** que están realizando una compra: vemos el número de ticket, el nombre del cliente (por defecto Cliente Desconocido) y la hora a la que se ha creado. Si hay varias personas en el establecimiento y están siendo atendidas podemos añadir sus tickets pinchando en el símbolo "+" del menú. Si por el contrario alguien abandona la tienda se puede eliminar su ticket dando a "-".

- **Botón de conexión**: nos indica el estado de la conexión (indicado con un círculo amarillo). Cuando se pone en rojo significa que se ha perdido la conexión a Internet.

- **Cerrar**: sirve para cerrar la sesión y volver a la pantalla anterior. Si tenemos pedidos pendientes nos avisa antes de salir.

Como creamos tres **categorías** de productos Pescados, Mariscos y Congelados, estas aparecen en la pantalla. Si accedemos a una de las categorías comprobamos que dentro de cada una tenemos los productos. Para volver a ver todas las categorías y los **artículos disponibles** en la tienda basta con dar al símbolo de la casa.



Podemos **añadir artículos** o productos al carro de la compra de un cliente, simplemente haciendo click sobre el icono del producto. Lo que hacemos es seleccionar los productos y las cantidades. Las cantidades vienen dadas en unidades y peso porque así lo configuramos en la unidad de medida de ventas. Si queremos poner un peso que no sea un número entero (0,900 lb) tenemos que usar el teclado virtual inferior activando la pestaña Cant. (cantidad).

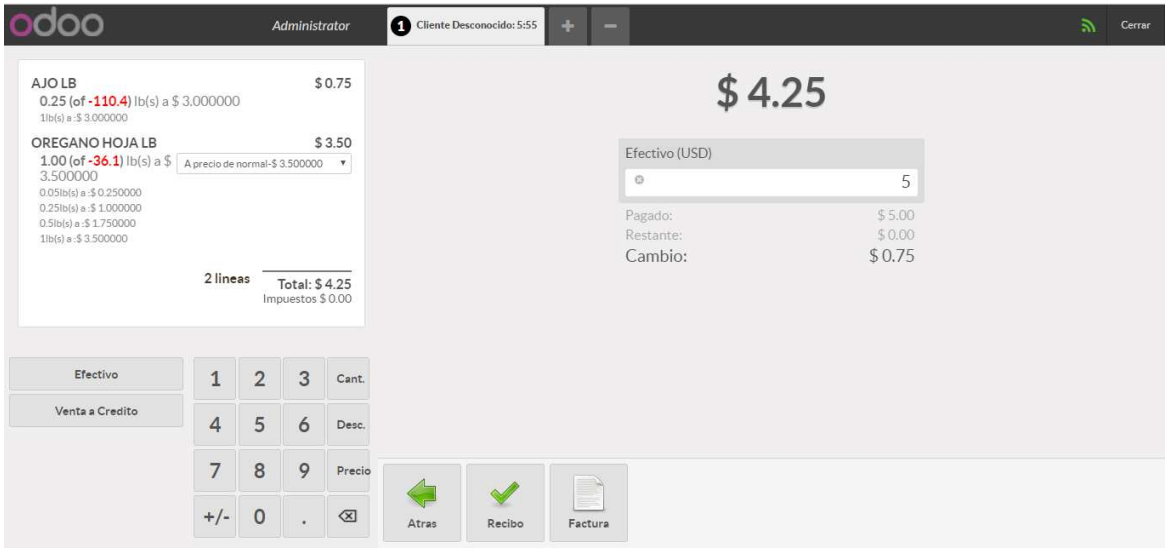
En varios productos debido a que el negocio ofrece diferentes presentaciones se pueden agregar, cada uno de los ítems que se deseen establecer, de igual manera la implementación de la etiqueta permitirá la lectura del mismo incluyendo su peso y su valor a la venta

En la parte inferior izquierda de la interfaz, encontramos el teclado virtual nombrado y los **métodos de pago** disponibles (Efectivo, Venta a Credito) Con este teclado introducimos los siguientes datos:

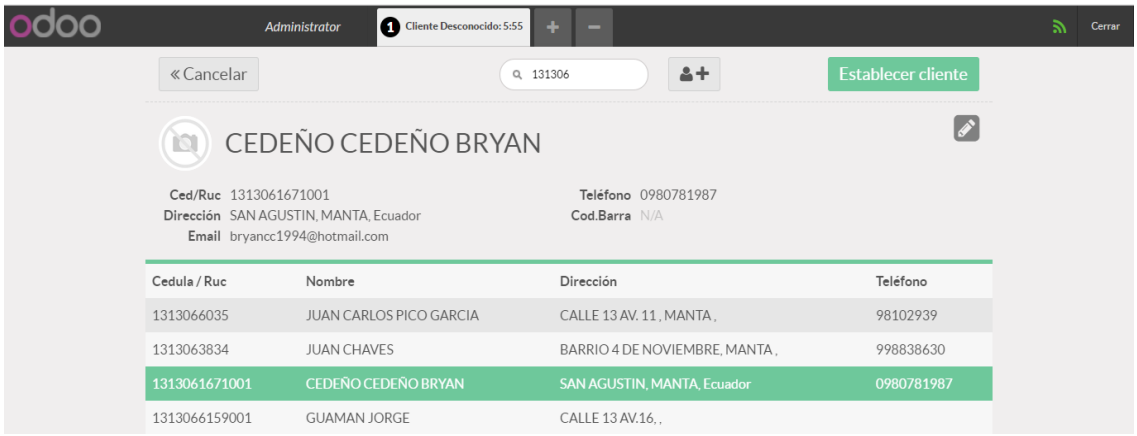
- Cantidades: Cant. como ya dijimos es útil si queremos definir una cantidad no entera, solo hay que introducir los dígitos precisos.
- Descuentos: si damos a Desc. Nos permite realizar descuentos sobre los productos que seleccionemos. Aparece en tanto por ciento sobre el precio final.
- Precios: con esta opción podemos dar el precio que queramos al producto. Si por ejemplo regalamos un artículo a un cliente podemos darle un valor de 0,00 dolares.

Una vez finalizado el pedido del cliente, escogemos el método de pago que utiliza y pinchamos sobre él. En este caso, se trata de pago en efectivo, por tanto, pulsamos en el botón **Efectivo**. Aparece un campo donde introducir la cantidad de dinero que nos entrega el cliente para que el sistema calcule el cambio. Si todo es correcto pinchamos en **Validar** y nos permite imprimir la factura o guardarla en formato PDF.

Una vez concluida la venta, volvemos a la pantalla inicial y seguimos con los pedidos de los demás clientes.


odoo
Administrator
1 Cliente Desconocido: 5:55
Cerrar
AJO LB $ 0.75
0.25 (of -110.4) lb(s) a $ 3.000000
1lb(s) a :$ 3.000000
OREGANO HOJA LB $ 3.50
1.00 (of -36.1) lb(s) a $ 3.500000
A precio de normal-$ 3.500000
0.05lb(s) a :$ 0.250000
0.25lb(s) a :$ 1.000000
0.5lb(s) a :$ 1.750000
1lb(s) a :$ 3.500000
2 lineas
Total: $ 4.25
Impuestos $ 0.00
$ 4.25
Efectivo (USD)
5
Pagado: $ 5.00
Restante: $ 0.00
Cambio: $ 0.75
Efectivo
Venta a Credito
1 2 3 Cant.
4 5 6 Desc.
7 8 9 Precio
+/- 0 .
Atras
Recibo
Factura


odoo
Administrator
1 Cliente Desconocido: 5:55
Cerrar
« Cancelar
131306
Cedula / Ruc | Nombre | Dirección | Teléfono
1313066035 | JUAN CARLOS PICO GARCIA | CALLE 13 AV. 11 , MANTA , | 98102939
1313063834 | JUAN CHAVES | BARRIO 4 DE NOVIEMBRE, MANTA , | 998838630
1313061671001 | CEDEÑO CEDEÑO BRYAN | SAN AGUSTIN, MANTA, Ecuador | 0980781987
1313066159001 | GUAMAN JORGE | CALLE 13 AV.16, , |


odoo
Administrator
1 Cliente Desconocido: 5:55
Cerrar
« Cancelar
131306
Establecer cliente
CEDEÑO CEDEÑO BRYAN
Ced/Ruc 1313061671001
Dirección SAN AGUSTIN, MANTA, Ecuador
Email bryancc1994@hotmail.com
Teléfono 0980781987
Cod.Barra N/A
Cedula / Ruc | Nombre | Dirección | Teléfono
1313066035 | JUAN CARLOS PICO GARCIA | CALLE 13 AV. 11 , MANTA , | 98102939
1313063834 | JUAN CHAVES | BARRIO 4 DE NOVIEMBRE, MANTA , | 998838630
1313061671001 | CEDEÑO CEDEÑO BRYAN | SAN AGUSTIN, MANTA, Ecuador | 0980781987
1313066159001 | GUAMAN JORGE | CALLE 13 AV.16, , |

FECHA:	29 AGOSTO 2017		
NOMBRE:	CEDEÑO CEDEÑO BRYAN		
CED/RUC:	1313061671001		
DIRECCION:	SAN AGUSTIN		
AJO LB	0.25	3.00	0.75
OREGANO HOJA LB	1.00	3.50	3.50

TAR 12% : $ 0.00 **TAR 0% :** $ 4.25

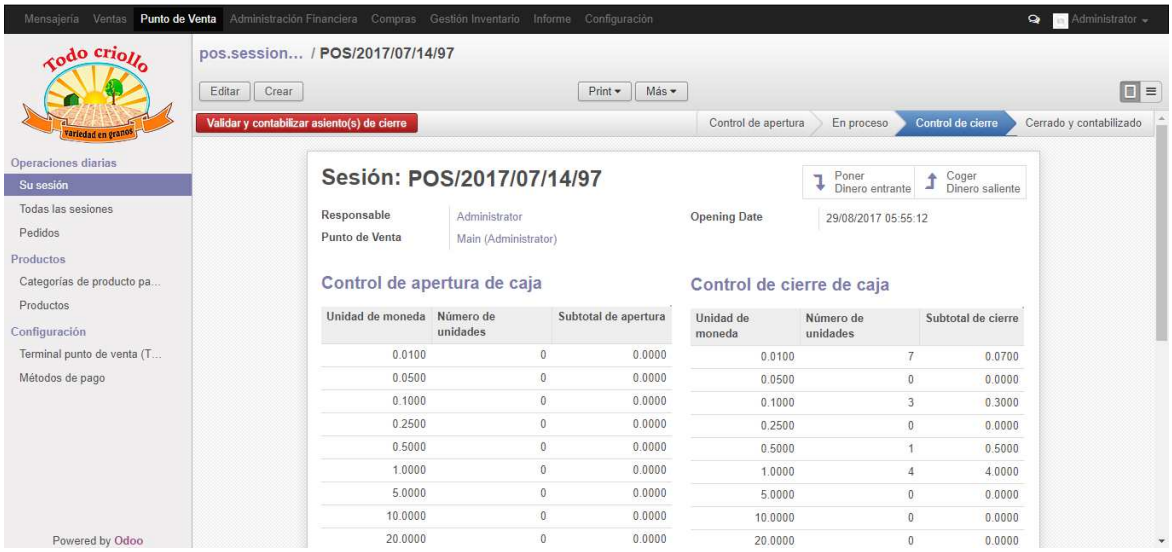
SUBTOTAL : $ 4.25 **DESCUENTO :** $ 0.00

IVA : $ 0.00 **TOTAL :** $ 4.25

--GRACIAS--

Finalmente, cuando la jornada de ventas termina, salimos de la sesión pulsando el botón **Cerrar** y de nuevo en **Confirmar** (situados en la esquina superior derecha). Con esta acción volvemos a la interfaz de Odoo habitual donde procedemos a **cerrar** definitivamente **la sesión.** El siguiente paso nos lleva al **Control de cierre** donde establecemos la cantidad final de efectivo en caja. De esta forma, validamos que el saldo final coincide con el saldo inicial más el dinero recaudado en la sesión de ventas. Si nos fijamos en la esquina superior derecha hay dos opciones: Poner dinero entrante y Coger dinero saliente, ambas se utilizan para lograr el cuadre del valor monetario indicado en el sistema y el real.

Por último, damos al botón **Validar y contabilizar asiento(s) de cierre** para finalizar la operación. La pestaña Todas las sesiones (del menú izquierdo) contiene una lista que incluye dicha sesión cerrada y contabilizada.



Nota 1: Si tenemos conectada la caja registradora al sistema, antes de iniciar la sesión y comenzar las ventas nos aparece una pantalla en la que introducir el dinero efectivo que tenemos en dicha caja al comienzo del día (como ya hemos visto). Esto se puede configurar en el terminal punto de venta creado seleccionando la opción Cajón de monedas en el apartado Proxy hardware. Así, introducimos la cantidad de dinero en efectivo (monedas y billetes) con la que empezamos el día. De esta manera, cuando finalice la jornada y cerremos la sesión, el sistema realiza el arqueo de caja.

Nota 2: para evitar que un producto agotado aparezca como disponible en un terminal o tienda, hay que configurar el producto yendo a la pestaña Ventas y quitar la opción Disponible en el TPV. De este modo, desaparece del TPV al que pertenece.

Generación de informes

En la mayoría de las compañías resulta primordial poseer o generar determinados documentos (facturas, órdenes de pedidos, albaranes, recibos, etc.) e informes (contables, de compras, de ventas, pérdidas y ganancias, balance de situación, etc.), además de contar con la posibilidad de personalizarlos según sus necesidades. Para nuestra empresa no es distinto, también requiere la creación y recopilación de información sobre sus procesos y la marcha del negocio.

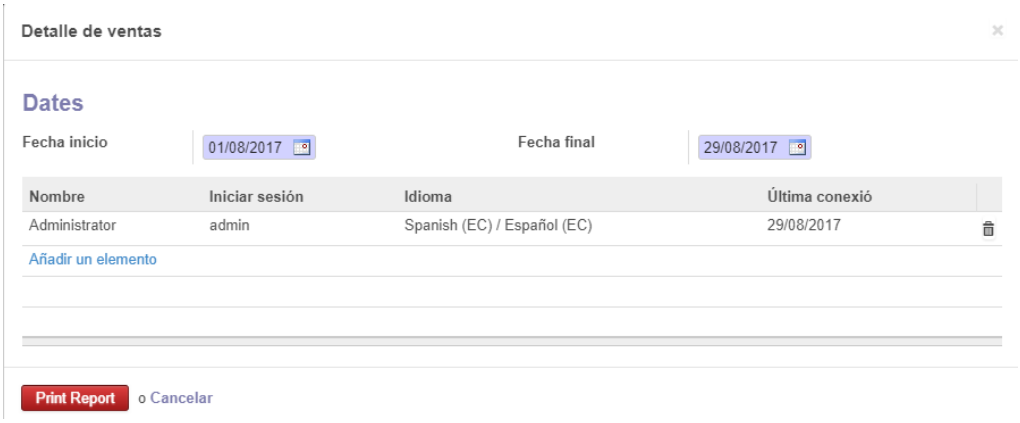
Para llevar a cabo esta tarea, la aplicación ERP Odoo cuenta con un sistema de generación de informes y estadísticas dinámicas basado en el lenguaje de descripción de documentos Report Markup Language (**RML**) y el nuevo motor de plantillas o modelos denominado **QWeb**. Gracias a esta herramienta podemos obtener información precisa de las actividades que realiza la empresa. Cada módulo permite crear y modificar sus propios documentos e informes, además de poder exportar los datos en formatos como PDF o Microsoft Excel. El usuario debe tener en cuenta que no resulta una tarea tan sencilla, aprender a manejar esta herramienta requiere tiempo y contar con unos conocimientos básicos sobre la funcionalidad. En este apartado, intentamos proporcionar las bases para poder trabajar con el sistema.

Por un lado, en el módulo de **Contabilidad**, podemos ir al apartado **Informe** del menú izquierdo, para consultar dos tipos de informes: **genéricos** y **legales**. Dentro de los genéricos tenemos de Saldo, de Saldos vencidos, Libro mayor de la empresa y de Impuestos. Dentro de los legales están los Informes contables y los Diarios. En el primer grupo se encuentran: Libro mayor, Balance de sumas y saldos, Balance de situación, Pérdidas y ganancias e Informe financiero. Por otro lado, si pinchamos en la opción **Informes** del menú superior, vemos los apartados correspondientes a cada módulo. En ellos se encuentran los distintos informes y estadísticas dinámicas sobre las diferentes actividades de la empresa. En nuestra situación, los que más nos pueden interesar son los de pedidos de compras, de ventas y de pérdidas y ganancias.

Suponemos que un momento determinado, el responsable de la tienda quiere conocer los ingresos por ventas durante la última semana del mes de noviembre. Basta con acudir al apartado **Terminal Punto de Venta** y hacer clic en **Detalles de ventas**. Así aparece una ventana donde seleccionar las fechas y el responsable de las ventas. Damos al botón rojo **Imprimir informe** y se genera un documento PDF con toda la información sobre las ventas: fecha, referencia de pedido, producto, precio, cantidad, así como el total de impuestos aplicados, las formas de pago de los clientes y el total de ingresos.

Si además de los pedidos de venta, al usuario le interesa saber cómo han sido las compras de dicha semana, nos situamos en el apartado **Purchase** (Compra) del menú izquierdo y pinchamos en **Análisis compra**. Como ya vimos cuando hablamos de los tableros de información, aparece un listado personalizable acerca de los pedidos. Observamos los filtros introducidos, un tipo de vista en formato tabla y el mapa crítico que nos permite clasificar visualmente los elementos gracias a la variación del color.

Toda esta información puede ser recogida de manera conjunta en el informe de **Pérdidas y ganancias** generado por el sistema. Yendo a la pestaña Contabilidad del menú superior y al apartado Informes citado anteriormente, seleccionamos dicho documento.



Detalles de ventas

Compania:
Todo Criollo
Periodo de Inicio:
01/08/2017

Usuarios:
Administrator
Periodo final:
29/08/2017

Print Date:
29/08/2017

Fecha	Orden	Producto	Precio	Cant.	Disc(%)	Invoiced
29/08/2017 06:11:07	Main/0694	[] AJO LB	$ 3.00	0.25 lb(s)	0.00	VENTA/2017/0627
29/08/2017 06:11:07	Main/0694	[5000000001224] OREGANO HOJA LB	$ 3.50	1.00 lb(s)	0.00	VENTA/2017/0627

Impuestos	
iva0 - IVA 0%(SIN DERECHO A CREDITO TRIBUTARIO)	**$ 0.00**
Pago	
Efectivo	**$ 4.2500**

GENERAR INFORME DE PRODUCTOS CON CÓDIGO DE BARRA

En esta sección de este documento, se procede a realizar el informe que nos permita generar códigos de barras personalizados de acuerdo a los productos y a sus diferentes presentaciones ofrecidas por la empresa Todo Criollo.

Nos dirigimos a la sección de Informe, luego la opción de Impresión de código en Listado de productos

Aparecerá una nueva ventana, si se desea filtrar se ubica la categoría del TPV, a continuación, se añaden los productos en la opción de añadir un elemento, una vez agregados todos los productos, se procede a dar click en la opción cargar y se desplegara todos los productos con sus presentaciones disponibles.

Exportar productos xls

Asistente para la impresion de codigo EAN:

XLS Exportacion

Categoria de TPV

Producto	Descripcion	Codigo Ean	Precio	Cantidad
Añadir un elemento				

Cargar

Close Export

Exportar productos xls

Asistente para la impresion de codigo EAN:

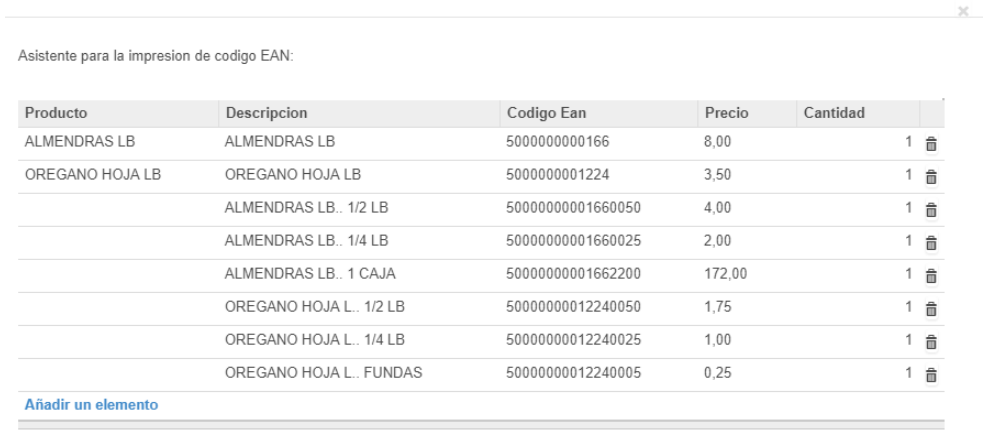
XLS Exportacion

Categoria de TPV

Producto	Descripcion	Codigo Ean	Precio	Cantidad
ALMENDRAS LB	ALMENDRAS LB	5000000000166	8,00	1
OREGANO HOJA LB	OREGANO HOJA LB	5000000001224	3,50	1
Añadir un elemento				

Cargar

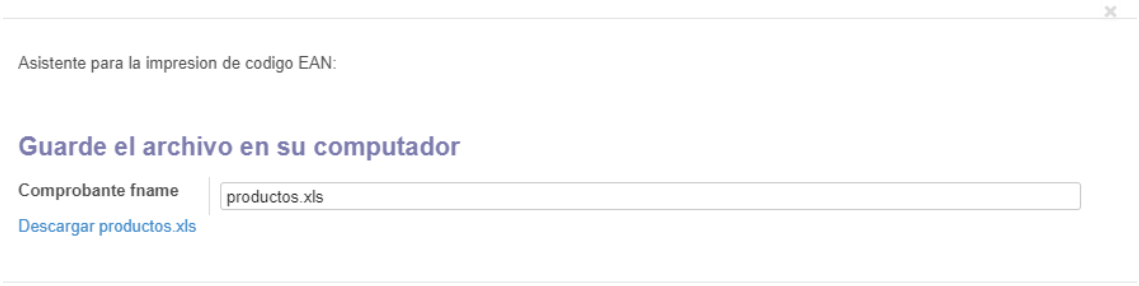
Close Export


Asistente para la impresion de codigo EAN:

Producto	Descripcion	Codigo Ean	Precio	Cantidad
ALMENDRAS LB	ALMENDRAS LB	5000000000166	8,00	1
OREGANO HOJA LB	OREGANO HOJA LB	5000000001224	3,50	1
	ALMENDRAS LB.. 1/2 LB	50000000001660050	4,00	1
	ALMENDRAS LB.. 1/4 LB	50000000001660025	2,00	1
	ALMENDRAS LB.. 1 CAJA	50000000001662200	172,00	1
	OREGANO HOJA L.. 1/2 LB	50000000012240050	1,75	1
	OREGANO HOJA L.. 1/4 LB	50000000012240025	1,00	1
	OREGANO HOJA L.. FUNDAS	50000000012240005	0,25	1

Añadir un elemento

Una vez establecida las cantidades se procede a exportar el archivo, dando click en la opción **exportar,** y aparecerá una nueva ventana para descargar el archivo.

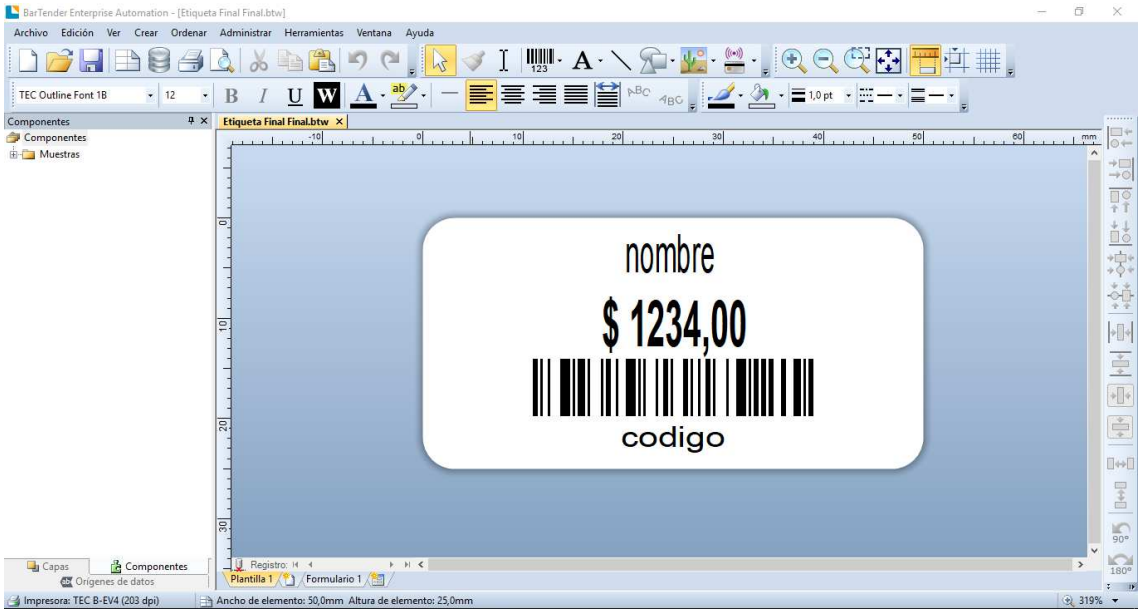

Asistente para la impresion de codigo EAN:

Guarde el archivo en su computador

Comprobante fname | productos.xls

Descargar productos.xls

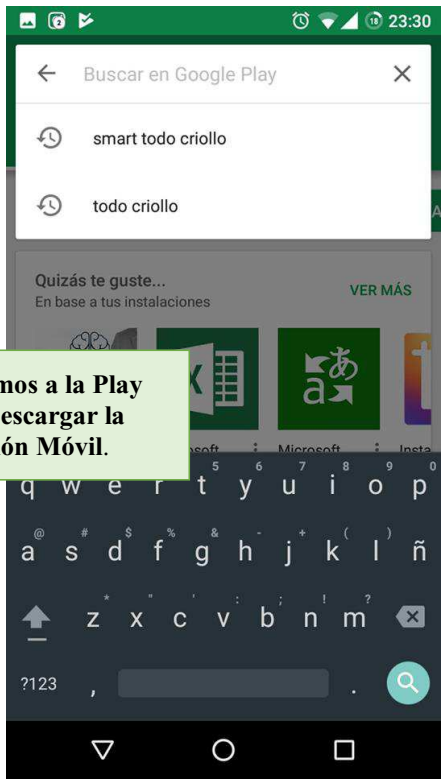
Luego usaremos un programa externo con la plantilla previamente creada, a continuación, se carga el archivo Excel en el programa Bartender y se procede a imprimir.


BarTender Enterprise Automation - [Etiqueta Final Final.btw]
Archivo Edición Ver Crear Ordenar Administrar Herramientas Ventana Ayuda
TEC Outline Font 1B
12
Componentes
Componentes
Muestras
Etiqueta Final Final.btw
nombre
$ 1234,00
codigo
Capas
Componentes
Orígenes de datos
Plantilla 1
Formulario 1
Impresora: TEC B-EV4 (203 dpi)
Ancho de elemento: 50,0mm Altura de elemento: 25,0mm
319%

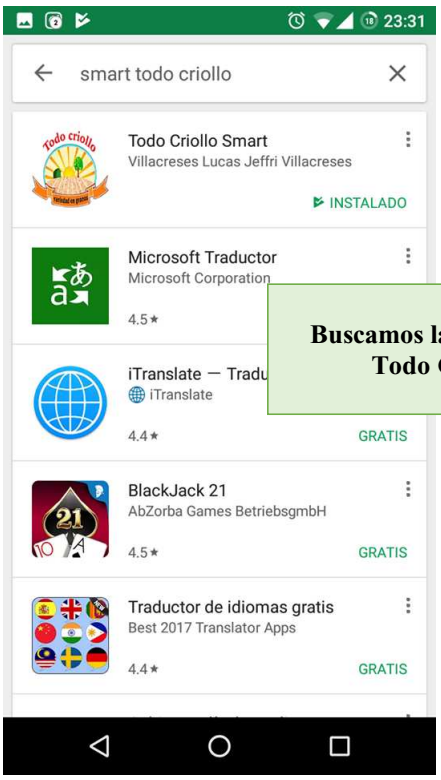
Todo Criollo

MANUAL DE USUARIO

APLICACIÓN MOVIL CATALOGO



Nos dirigimos a la Play Store a descargar la aplicación Móvil.



Buscamos la app Smart Todo Criollo

Presionamos en el botón Instalar.

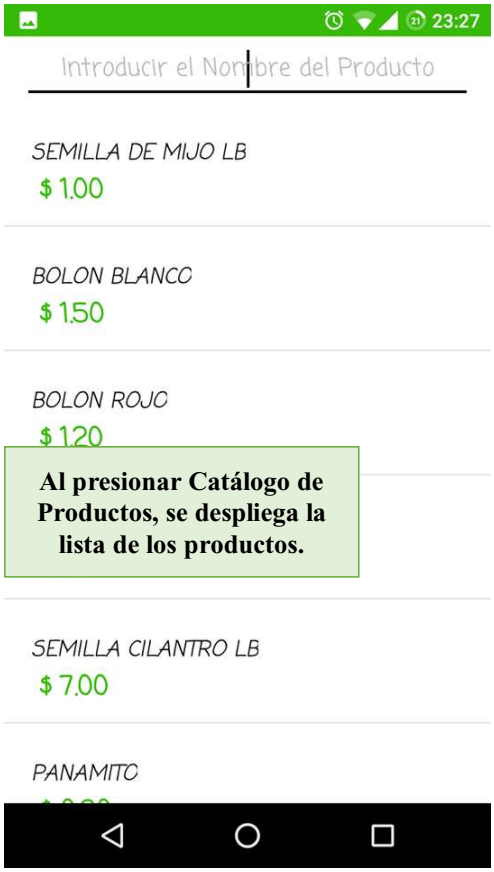
Esperamos la descarga.

Abrimos la aplicación.

Presionamos el botón en la parte inferior derecha y se despliega el menú.

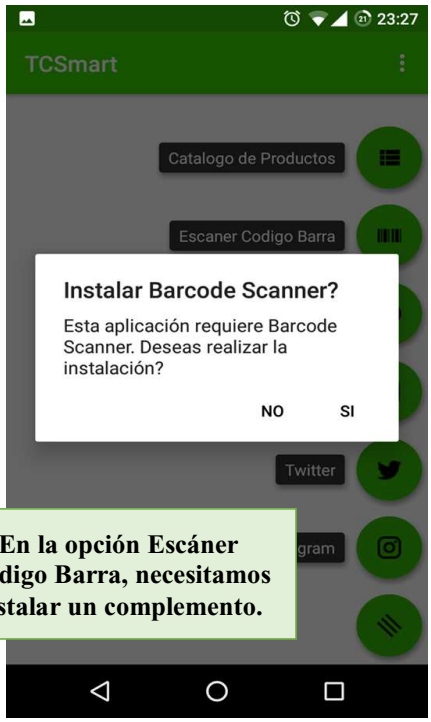
Al presionar en el botón superior derecho, se despliega la opción Acerca De.

Encontraremos la información de la App.



Al presionar Catálogo de Productos, se despliega la lista de los productos.

Se puede filtrar por medio de la caja de búsqueda.



En la opción Escáner Código Barra, necesitamos instalar un complemento.

Presionamos Instalar y esperamos unos segundos.

Comercial "Todo Criollo"
Variedad en granos

Capturamos el código de barras.

Se despliega la información del producto.

Al presionar en Sitio web, se abrirá la página web.

Al presionar en Facebook, se abrirá la red social.

Al presionar en Twitter, se abrirá la red social.

Al presionar en Instagram, se abrirá la red social.