Grado en Ingeniería Informática

Trabajo de Final de Grado

Desarrollo e implantación de Odoo con múltiples puntos de venta

Autor:
Carlos Roca Zaragoza

Supervisor:
Sergio Teruel Albert
Tutor académico:
M. Ángeles López Malo

Fecha de lectura: 16 de julio de 2020
Curso académico 2019/2020

Resumen

Este proyecto se basa en la extensión e implementación de un sistema de gestión empresarial de código abierto. La finalidad del proyecto es actualizar el sistema antiguo de una empresa que se dedica a la venta de productos asociados a los cigarrillos electrónicos y cubrir las nuevas necesidades que le surgen al administrador. Para ello se instalan los módulos que se necesitan del estándar de Odoo y se desarrollan los necesarios para cubrir esas necesidades que no cubre el sistema.

El sistema dispone de herramientas para gestionar las compras, las ventas, las tiendas físicas y su almacén. Se puede gestionar la página web, tanto la apariencia, como las cantidades disponibles para la venta en línea. Para conseguir esto, se ha hecho uso de los módulos que nos ofrece Odoo y la OCA, y se ha desarrollado alguno que cubra alguna funcionalidad no cubierta.

Palabras clave

Odoo, ERP, Python, Aplicación web, JavaScript

Keywords

Odoo, ERP, Python, Web application, JavaScript

Índice general

Capítulo 1

Introducción

1.1. Contexto y motivación del proyecto

Tecnativa [1] es una empresa española que se fundó en el año 2016, a partir de la unión de experimentados profesionales de Odoo. Esta, se dedica a la migración de versiones de los módulos contenidos en la OCA (Odoo Community Association) y a la implantación del ERP (sistema de gestión empresarial) con los módulos necesarios en la empresa que lo requiera. Esta empresa dispone de una oficina en la cual empecé a cursar las prácticas en Burriana, en la que trabajan 2 de sus trabajadores y el resto realizan sus tareas mediante el teletrabajo. Cabe destacar que durante la estancia en prácticas, se ha dado una pandemia de índole mundial, que ha forzado que todos trabajemos desde casa y que termine mi estancia en prácticas mediante el teletrabajo, lo que me ha permitido saber cuál es el procedimiento y la forma de trabajar desde casa.

Para entender qué es Odoo y qué es la OCA, me gustaría hacer un inciso:

- **Odoo** [2] es un ERP de código abierto que cubre las necesidades de áreas como:

  Contabilidad y Finanzas

  Ventas

  RRHH

  Compras

  Proyectos

  Almacenes

  CRM (Gestor de Relaciones con los Clientes)

  Fabricación

  Y muchas más áreas.

  Dentro de la plataforma de Odoo, existen una serie de módulos que forman la llamada versión “enterprise”. Estos módulos tratan de hacer que la gestión empresarial sea más

sencilla. Sin embargo, para adquirir estos módulos hay que pagar una cuota que en algunos casos es elevada.

- **OCA** [3] es una organización sin ánimo de lucro. Tienen la misión de promover el uso de Odoo y fomentar el desarrollo colaborativo de sus módulos. Para ello, disponen de un repositorio en GitHub con más de 900 contribuidores y un sistema muy detallado para contribuir [4].

Por lo que respecta al proyecto, la cadena de empresas Street Vape One, que se dedica a la venta de productos asociados a los cigarrillos electrónicos, quiere informatizar su sistema de ventas, compras, almacenaje y que sus tiendas tengan los pertinentes puntos de venta, por lo que buscan un ERP con varias funcionalidades. El proyecto consiste en la implantación de Odoo en esta cadena de empresas con la funcionalidad que necesitan, y desarrollar aquello que no nos ofrece ni Odoo ni la OCA. De esta forma, tenemos un proyecto de desarrollo y de implantación de Odoo junto con módulos de la OCA a una serie de empresas reales.

Durante muchos años, en esta cadena se utilizó una versión antigua de este ERP, en concreto la versión 8. Al tratar de actualizar el sistema a versión 13.0, se han encontrado con un abismo importante, puesto que no han sido capaces de llegar a tener lo que tenían en la versión antigua. A raíz de esto, decidieron acudir a Tecnativa para que les ayudasen y de esta forma aprovechar para tener la funcionalidad que siempre quisieron.

1.2. Objetivos del proyecto

En este apartado se van a mostrar los objetivos principales que se pretenden lograr con el desarrollo de este proyecto. Además, se presentará el alcance en sus distintos ámbitos (funcional, organizativo e informático).

El objetivo principal del proyecto, es implantar la versión 13 de Odoo al cliente Street Vape One y desarrollar los módulos necesarios para satisfacer las necesidades del cliente. Del mismo modo, hay que buscar que los módulos que se desarrollen cumplan los estándares de la OCA y que sean mantenibles en un futuro. Del mismo modo, en caso que se realice una migración de versión, hay que seguir las indicaciones que nos da la asociación comunitaria de Odoo.

Por lo que respecta al alcance del proyecto, se puede dividir en las secciones que se encuentran a continuación.

1.2.1. Alcance funcional

El ERP que se desarrolla debe cumplir con los siguientes requisitos funcionales, para que el sistema esté de acuerdo con la necesidad del cliente:

- Se desea instalar Odoo en la versión 13, de manera que se cubran las necesidades del cliente.

- Se ha de poder comprar tanto en tiendas físicas como en la tienda *online*. Para ello, habrá que configurar los puntos de venta y la web a la legislación española.
- Al tratarse de una cadena de empresas, hay que configurar la multi-compañía que nos permitirá gestionar la contabilidad, inventariado, las ventas... de forma que funcione todo de forma independiente.
- Se quiere una web, con su tienda en línea que esté vinculada al almacén de la tienda principal.
- El cliente quiere tener un sistema de inventariado de sus almacenes y de gestión de las compras, de forma que no compre demasiados productos y que no se quede sin ellos.
- El cliente quiere ser formado en el sistema para saber moverse por él y ser eficiente a la hora de realizar las operaciones debidas.
- No hay que desarrollar la pasarela de pago, ya que se usará la de Redsys, sin embargo, se debe crear la conexión con el sistema, por lo que esto último sí que entra dentro de nuestro alcance funcional.

1.2.2. Alcance organizativo

El sistema afecta a varios departamentos dentro de las empresas del cliente. En primer lugar, se va a poder llevar el inventario y controlar el almacén haciendo uso de herramientas que mediante las compras y las ventas nos aumenten o nos reduzcan el inventario. Se ha de poder gestionar el aspecto de la página web por parte del administrador de la plataforma. Además, las compras se van a poder realizar desde el sistema de punto de venta en las tiendas físicas y desde comercio electrónico. Así pues podemos concluir que el sistema afectará a los departamentos de las empresas relativas a compras, ventas, almacén y diseño.

1.2.3. Alcance informático

Para cubrir el alcance informático, se van a usar distintas tecnologías que son necesarias para desarrollar en Odoo.

- **Python 3.6**: Lenguaje usado para toda la funcionalidad relativa al *backend*. Se usa para la lógica, haciendo uso de las librerías de Odoo que nos permiten una conexión con la base de datos.
- **PostgreSQL 12**: Gestor de bases de datos relacional. Es un requisito necesario tener montado un servidor PostgreSQL para poder ejecutar correctamente Odoo.
- **XML**: Lenguaje usado para las vistas de Odoo, tanto las vistas del *frontend* como del *backend*. Esta herramienta nos permite crear una conexión entre la lógica y lo que se ve en el sistema.

- **JavaScript ES6**: Este lenguaje se usa para la funcionalidad que no es necesaria que se cargue en el servidor. De esta forma conseguimos que mejore el rendimiento, al cargar datos desde el cliente de forma asíncrona.

Aparte de estas tecnologías con las que hemos desarrollado, para poder hacer uso de ellas, son necesarias algunas herramientas. En mi caso, he usado PyCharm Community para programar, a pesar de tener la opción de usar Visual Studio Code. Esta decisión se debe a la facilidad de ejecución que ofrece haciendo uso de un fichero de configuración. Además de esta herramienta, para la distribución de las actividades y planificación de las tareas, he hecho uso de Odoo.

1.3. Requisito extra

Como se parte desde un proyecto ya creado por el cliente, existe un requisito implícito, que es necesario para que el desarrollo de la funcionalidad que se va entregar sea lo más eficiente posible. Este requisito consiste en analizar los módulos que el cliente tiene instalados en el sistema y decidir si lo que se nos ha pedido puede resultar necesario.

Además de esto, el cliente dispone de la versión "enterprise", que incluye una serie de módulos que amplían la funcionalidad de Odoo. A pesar de que esta versión disponga de varios módulos que nos pueden ser de utilidad, hay que destacar que puede que las funcionalidades que cubran alguno de estos también estén cubiertas por algún modulo de la OCA. En este caso, se debería valorar si el módulo de la OCA cubre mejor la funcionalidad que se espera.

Capítulo 2

Descripción del proyecto

2.1. Situación inicial

Street Vape One actualmente cuenta con 5 compañías, de las cuales una de ellas, va a ser llamada la compañía central, puesto que es la que contacta con proveedores y provee a las otras de productos. Cabe destacar que cada una de las compañías dispone de alguna tienda física. El cliente dispuso durante mucho tiempo del servicio de Odoo, concretamente de la versión 8. Un día decidieron actualizar el sistema a la versión actual, siendo esta la versión 13. Al plantarse ante el nuevo sistema, el cliente sintió que había cosas que antes tenía que ahora no era capaz de encontrar. Por esto pidió ayuda a Tecnativa, para tratar de encauzar el rumbo y el flujo de acciones necesarias para realizar un proceso, y de esta forma aprovechar para añadir alguna funcionalidad extra para su sistema.

2.1.1. Metodología de trabajo en el cliente

Para explicar el modo en el que se trabaja desde Street Vape One (tanto con la versión antigua de Odoo como con la nueva), podemos explicar el recorrido que tienen los productos, el cual se esquematiza en la figura 2.1. Desde la compañía central se compran una serie de productos a proveedores. De esta forma empieza el proceso.

Una vez los productos han llegado a la compañía central, se almacenan hasta ser comprados. A partir de aquí pueden tomar tres direcciones.

En primer lugar, pueden ser comprados por una persona que se presenta a la tienda física que pertenece a la compañía central. En este caso se va a poder pagar en efectivo o tarjeta.

Otra opción, puede ser que un cliente se conecte a la página web, en la que las cantidades que disponen están asociadas a la compañía central. Cuando se realiza una compra de esta forma en caso que supere una cantidad de precio, el envío sale gratuito. Para este tipo de compras solo se puede pagar con tarjeta.

Finalmente, existen varias franquicias que están en el barrio de Pozuelo, el de Tres Cantos, el de Boadilla de Madrid y en Toledo. Estas franquicias compran los productos con los que se suministran a la empresa principal y posteriormente ellos los venden a los clientes finales.

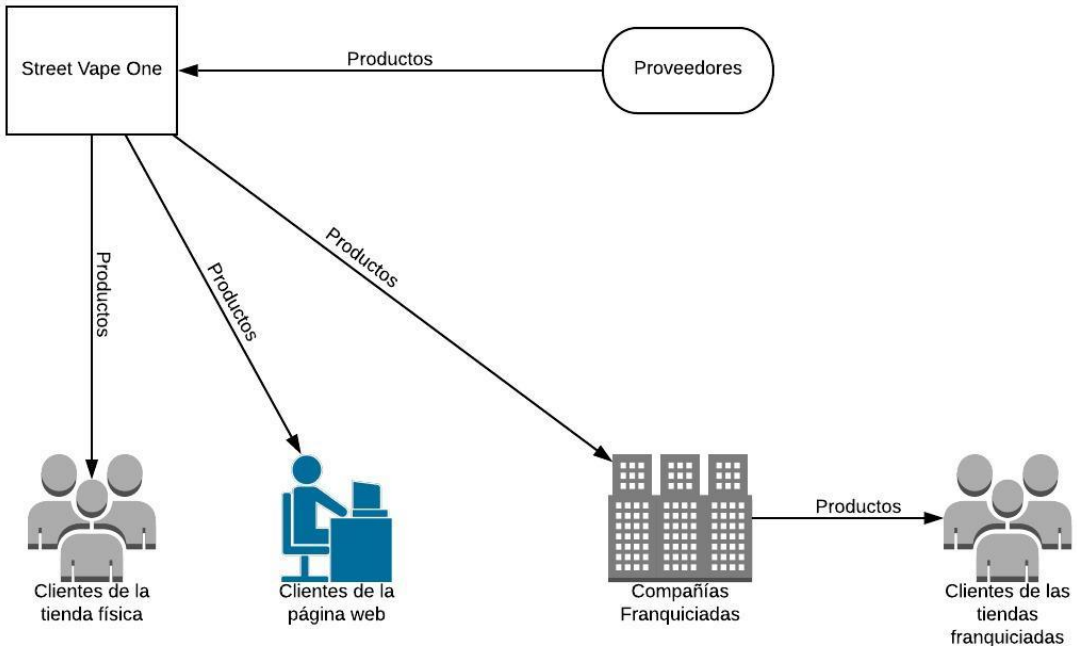

Figura 2.1: Ilustración del modelo de trabajo en Street Vape One.

2.1.2. Gestión de Almacenes

Hablando con el cliente, nos comentó que él era el que se encargaba de gestionar los almacenes de todas las franquicias. Para ello, contaba a mano los productos y para hacer inventario, lo realizaban a mano, para así incluir los productos de más que tenían o que dejaban de tener.

Por ello, me pareció importante implantar en este sistema una forma de automatizar un poco más el proceso. Como es lógico, desde Odoo se ofrece esto, pero el cliente no sabía usarlo, al igual que yo al principio. De esta forma se consigue a partir de las compras que se realicen, llenar los almacenes, y que a partir de las ventas, estos se vacíen. De esta forma, le reducimos la cantidad de veces que va a tener que realizar el inventario del almacén, y le vamos a facilitar el proceso de monitorización de los productos.

2.1.3. Página web

Como el usuario que se ocupa de la administración de un sistema de Odoo dispone de la capacidad de instalar temas en la web, al descargar la base de datos, me encontré con una cantidad desmesurada de dependencias. Esto fue debido a que el cliente instaló muchos temas que luego no fueron desinstalados, por lo que los tuvimos que quitar a mano cada uno de los

temas (aunque posteriormente se añadirían de nuevo), para trabajar con el tema por defecto de Odoo.

2.2. Descripción de la aplicación

Como ya hemos mencionado, el cliente ha querido pasar de la versión 8 a la 13, y esto va a suponer un gran cambio, puesto que hablamos de 5 versiones de diferencia.

Así pues, se van a poder gestionar las compras directamente desde la aplicación, que ya tenían disponible en la versión 8, pero que no estaban usando. De hecho, aunque el cliente no ha querido, hubieran podido tener los pedidos automatizados, según la cantidad de productos en el almacén y realizar un pedido vía correo electrónico directamente al proveedor, esta última funcionalidad la incluye la versión 13, pero como se sale de su proceso de compra, han preferido que no se use.

Por lo que respecta a las ventas, se va a disponer de tres opciones de ventas desde la misma aplicación en línea, así pues, van a poder vender en línea, en las tiendas físicas, o bajo pedido (cuando se trate de ventas entre franquiciados) y de esta forma no va a ser necesario software separado para cada función. Para evitar que los trabajadores que solo pueden acceder al punto de venta, accedan a otros puntos de la aplicación, como puede ser el inventario o la contabilidad, se les otorgan permisos en las aplicaciones.

Las unidades de productos disponibles en la aplicación se van a actualizar automáticamente, puesto que de esta forma evitaremos la interacción continuada con el sistema por parte del usuario. Así pues, se va a implantar un sistema por el que el cliente realizará compras que aumentarán las cantidades y ventas que las reducirán. Complementario a esto, se van a poder sacar informes de las cantidades disponibles de cada uno de los productos que se disponen.

La aplicación nos va a permitir editar la página web a nuestro gusto a partir de un tema. Por lo que el cliente va a tener libertad de editarla a su gusto. Así pues, va a poder insertar diferentes bloques editables que añade Odoo. De esta forma, se ofrece una forma sencilla e intuitiva para que el administrador de la página web añada aquello que le parezca necesario. Además de esto, podrá mover los elementos arrastrándolos por la página web y va a poder editar las gamas de colores de forma que se distribuya la información a su gusto.

2.2.1. Beneficios respecto a la versión anterior

Como hemos dicho, el cliente disponía de una versión 8 antes de decidirse por Odoo 13. Como es lógico se presentan distintas ventajas [5], pero las más destacables son las siguientes:

- Se mejora la velocidad de interacción entre los componentes del sistema.
- El *framework* de desarrollo se mejora, de forma que se puedan realizar personalizaciones más baratas y eficientes.

- El punto de venta ofrece mayores posibilidades, como poder cambiar de cajero al vuelo o gestionar tanto tiendas como restaurantes.

- Se mejora la contabilidad, a través de la eliminación de datos no necesarios y el uso de estructuras más eficientes.

- En las ventas y las compras se unifican los flujos desde el pedido. De esta forma se permite facturar desde el mismo punto de bienes y servicios, con la política de facturación que se desee.

2.3. Tecnologías

Como el proyecto parte de un ERP como es Odoo, ya tenemos las tecnologías que vamos a usar marcadas. A pesar de esto, veo importante entrar en detalle en cada una de ellas, de esta forma entender la lógica de Odoo. Así pues, las tecnologías que se han usado para el proyecto son las que se encuentran en los siguientes puntos.

2.3.1. Python

Lenguaje de programación de alto nivel, cuya principal filosofía es la legibilidad del código. Además de esto, da la posibilidad a los programadores de conseguir una función con menos líneas de código.

Por estas razones es un lenguaje muy bien elegido para la lógica del *backend* de Odoo. Esto es porque al ser un programa *Open Source* colabora mucha gente, por lo que un lenguaje de fácil entendimiento permite a los revisores de código entender de forma más sencilla lo que un programador hace o pretende hacer.

Además de esto, gracias a las librerías que incluye Odoo nos permite enfocarlo al sistema de forma muy sencilla y eficiente.

2.3.2. XML

XML es un lenguaje que se usa para hacer referencia a otro. Se suele usar para el etiquetado de documentos.

En el caso de Odoo, con XML se montan las vistas que después se transforman en html. Gracias a XPATH podemos hacer referencia a los componentes de la vista ya cargados, de forma que con un selector, podemos añadir nuevos campos a un vista sin romperla.

2.3.3. JavaScript y JQuery

JavaScript es un lenguaje que usualmente se usa del lado del cliente, como en Odoo pasa, de forma que se consiguen cosas que no son posibles haciendo uso simplemente de las tecnologías anteriormente mencionadas. Como dinamizar la página web, o incluso hacer test de interfaz de usuario.

Para conseguir acceder a cada apartado de la vista, en este caso, se usa JQuery, que igual que con el XPATH, mediante selectores nos permite acceder a las vistas que necesitamos. De esta forma combinando JavaScript, JQuery y XML podemos cargar vistas con mucha lógica de forma dinámica desde el lado del cliente. Gracias a este hecho ganamos mucha eficiencia y conseguimos no ralentizar la página.

2.3.4. PostgreSQL

Sistema de gestión de bases de datos relacionales. En Odoo se usa de forma que aloja todos los campos declarados en el código Python, por lo que existe un enlace de conexión entre la base de datos y lenguaje del *backend*.

2.4. Software Usado

Para tratar tecnologías que se han mencionado antes, ha sido necesario usar una serie de software específico. A continuación se puede ver cada uno de los que se han usado.

2.4.1. PgAdmin

Con esta aplicación que se ejecuta en el navegador, podemos controlar las bases de datos locales. Así como, crear, borrar, editar las columnas, etc.

En mi caso solo lo he usado para borrar bases de datos y resetearlas, puesto que Odoo se ocupa de todo lo demás.

2.4.2. PyCharm Community

IDE de programación elegido por mi, puesto que realizando un archivo de configuración se facilita mucho la ejecución del entorno de Odoo. Como inconveniente a este programa, es que el tratamiento de los archivos JavaScript no está disponible, por lo que no tenemos corrección de errores, en el caso que quisiéramos tratarlos tendríamos que pagar la versión IDEA.

2.4.3. GitHub y GitLab

Para compartir los desarrollos entre la comunidad de Odoo, se usa GitHub, concretamente el repositorio de la OCA. Pero en el caso de Tecnativa, no se realizan los *Pull Request* (PR) directamente a la OCA, sino que se crea una rama al repositorio de Tecnativa y desde este se realizan las acciones para realizar el PR. En la figura 2.2 se puede ver el repositorio de la OCA.

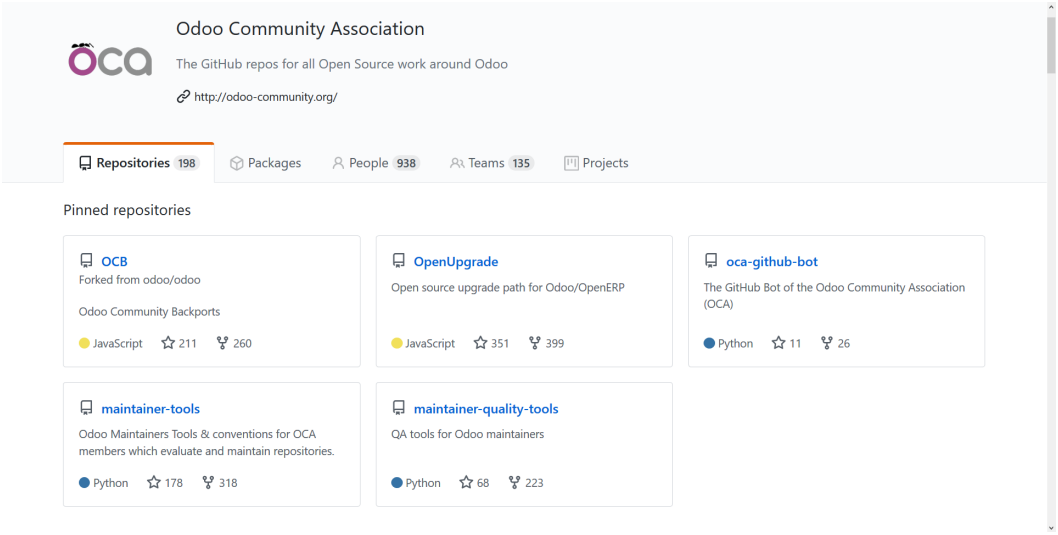

Figura 2.2: Repositorio de la OCA en GitHub

Para los proyectos de Tecnativa se usa GitLab, puesto que se aloja a un servidor privado y permite más privacidad. En este caso no se incluye imagen puesto que contiene los proyectos de la empresa que son privados.

2.4.4. Plugins de navegador

Para facilitar el desarrollo en Odoo se hace uso de algunos plugins que se consideran imprescindibles desde Tecnativa.

- **Odoo Debug**: Este *plugin* nos permite ejecutar el modo *debug* en las aplicaciones Odoo, de forma que podemos ver las opciones de desarrollador que Odoo nos ofrece.
- **Tampermonkey**: Mediante este *plugin* se ejecuta un *script* creado por Tecnativa, por el que se permite relacionar los *Pull Request* de GitHub con las tareas la empresa.
- **Odoo Terminal**: Este *plugin* está creado por Tecnativa dando la posibilidad de ejecutar comandos *shell* desde el navegador, sin necesidad de acceder al servidor.

Capítulo 3

Planificación del proyecto

3.1. Metodología

En este apartado, se va a explicar cuáles son los motivos que se tuvieron en cuenta para elegir la metodología que se ha usado para desarrollar el proyecto. Junto a esto se darán varias ventajas para justificar la elección de la metodología usada frente a otras.

La metodología de trabajo que se va a seguir está basada en la combinación de dos metodologías ágiles, *Scrum* y DRY (*Don't Repeat Yourself*) [6]. Se han elegido estas porque desde la empresa son las que se usan.

A pesar que sabemos que el sistema que se va a usar va a ser Odoo y el cliente presentó una serie de requisitos al inicio del proyecto, cabe la posibilidad que estos cambien, porque el desarrollo de una funcionalidad sea demasiado compleja para el tiempo de prácticas o porque haya cosas que son innecesarias y que pueden descartarse haciendo uso de otra vía. Así pues, podemos ver que a pesar de que se recopile una pila inicial, esta puede cambiar, por lo que una metodología predictiva no tiene cabida en este proyecto.

Así pues, para agilizar la distribución del trabajo se va a crear un *tablero kanban* y de esta forma priorizar el trabajo y ver en todo momento lo que nos queda por desarrollar.

La metodología DRY, la encontramos sobretodo en la especificación en el momento de desarrollar un módulo. Esto es porque hay una amplia serie de funcionalidades que ya están en Odoo, y como ya sabemos tenemos la OCA por el medio, por lo que también tenemos que analizar si el módulo ya está en los repositorios de la comunidad. Esto implica un período algo más amplio de especificación, pero que a la larga nos ahorra mucho tiempo de desarrollo.

Sabiendo esto podemos definir un proceso para el desarrollo del ERP web. En primer lugar vamos a recopilar todo lo que tenemos que hacer en primera instancia y así crear la pila de producto inicial. A continuación, se priorizan las tareas y se escogen las que son relativas al primer sprint (2 semanas). Después, se mira si los requisitos necesarios para el cliente ya están desarrollados en la OCA o se analiza si otro módulo de Odoo puede cubrir esa necesidad. La

intención es no desarrollar aquello que ya está hecho (Metodología DRY). Si se da el caso que esta funcionalidad no se puede cubrir, pasaríamos a producción del modulo que la empresa necesita, siempre orientándolo a que a los usuarios de la OCA les pueda llegar a servir en algún momento. Finalmente se expone qué se ha realizado durante el *sprint* y se planifica el siguiente.

Cabe destacar que existe alguna diferencia respecto a la metodología Scrum en el caso de este proyecto. A pesar de ser un proyecto destinado a un cliente, se trata de un proyecto destinado a las prácticas, aceptado por parte del cliente, por lo que tenemos un *Product Owner* que en este caso es el supervisor de la empresa, Sergio Teruel. Además de esto, el equipo de desarrollo está formado únicamente por mí. Finalmente, para exponer lo que llevamos, no se realizan Scrums diarios, sino que estos se realizan al comienzo y al final de la semana para exponer lo que vamos a hacer y lo que hemos hecho respectivamente.

3.2. Planificación

Sabiendo que la metodología es de tipo ágil, se presenta la planificación del proyecto que se ve en la tabla 3.1.

3.2.1. Pila del producto

Como la metodología de desarrollo es de tipo ágil, dentro de la planificación hay que abordar la pila del producto. Haciendo uso de las historias de usuario que se recopilaron en la toma de requisitos con el cliente, se llegó a la pila que se muestra en la tabla 3.2.

A pesar de tener una pila de producto inicial, durante el transcurso del proyecto se han ido añadiendo historias de usuario que el cliente creía importantes para su plataforma. De esta forma, se ha concluido con la pila del producto que encontramos en la figura 3.3.

3.3. Estimación de recursos y costes del proyecto

Para realizar una correcta estimación de los costes del proyecto se ha de tener en cuenta tanto el coste de personal, como el de software y hardware. Teniendo en cuenta esto nos ha salido un coste total de **3076,92 €** que se divide de forma detallada en los siguientes puntos.

3.3.1. Coste de Personal

Tras realizar una estimación del personal necesario para realizar este proyecto, concluimos que son necesarias 3 personas-mes. Como solo ha habido un desarrollador que ha realizado 300 horas de trabajo, podemos calcular el coste que supondría a la empresa sabiendo el sueldo promedio de los informáticos junior que trabajan con Odoo según Infojobs. Este sueldo ronda a los 14400 € anuales. Si dividimos este salario por las 52 semanas laborales y esto a su vez lo

Nº	Tarea	Tiempo
1	**Preparación propuesta técnica**	**15**
1.1	**Definición de requisitos**	**10**
1.1.1	Reunión con el cliente	2
1.1.2	Reunión con el supervisor de la empresa	4
1.1.3	Especificación del proyecto	4
1.2	**Planificación de la duración de los sprints**	**5**
2	**Desarrollo del proyecto**	**285**
2.1	**Preparación de la pila del producto**	**9**
2.1.1	Declaración de las historias de usuario	3
2.1.2	Priorizar historias de usuario	3
2.1.3	Validar pila del producto inicial	3
2.2	**Análisis**	**8**
2.2.1	Buscar funcionalidad ya desarrollada en Odoo	3
2.2.2	Buscar en el repositorio de la OCA funcionalidad ya desarrollada	3
2.2.3	Analizar dependencias para el desarrollo	1
2.2.4	Validar el análisis	1
2.3	**Desarrollo del proyecto**	**231**
2.3.1	Sprint 1	33
2.3.2	Sprint 2	33
2.3.3	Sprint 3	33
2.3.4	Sprint 4	33
2.3.5	Sprint 5	33
2.3.6	Sprint 6	33
2.3.7	Sprint 7	33
2.4	**Puesta en marcha**	**37**
2.4.1	Implantación	11
2.4.2	Formación	15
2.4.3	Entrega final	11

Cuadro 3.1: Planificación del proyecto

dividimos por las 40 horas semanales, nos sale un precio por hora de unos 7 €. Así pues el coste de desarrollo ascendería a los **7x300=2100 €**.

3.3.2. Coste Software

Como hemos mencionado en la sección 1.2.3 se hacen uso de muchas tecnologías para el desarrollo, sin embargo estas pueden ser tratadas con dos herramientas perfectamente.

- **PyCharm Community**: Es la versión gratuita del programa, la versión de pago es PyCharm IDEA, pero en nuestro caso no era necesario. Por lo que el coste ha sido 0 €.
- **Odoo**: Como con Odoo se gestionan varios departamentos, viendo las funciones que se han usado y usando la herramienta de tarifas de la página web, he calculado que ronda

Pila del producto inicial			
Código	Historias de Usuario	P.H.	Épica
SVO1	Registro de cliente	1	Login
SVO2	Inicio de sesión	1	Login
SVO3	Sistema de Punto de Venta	2	Ventas
SVO4	Sistema de Gestión de Compras	3	Compras
SVO5	Sistema de Gestión de Ventas	3	Ventas
SVO6	Sistema de almacenaje y logística	2	*Stock*
SVO7	Diseño de la tienda *online*	1	Diseño
SVO8	Desarrollo de la tienda *online*	4	Ventas
SVO9	Sistema de envío masivo de correo	2	Comunicación
SV10	Compra de un articulo	3	Compras
SV11	Formación de Odoo del desarrollador	2	Formación
SV12	Formación de Odoo al cliente	3	Formación
SV13	Toma de requisitos adicionales	3	Diseño
SV14	Capacitar varios canales de comunicación	5	Comunicación
SV15	Desarrollar un módulo de packs de artículos	5	Ventas
SV16	Implantación del Sistema ERP	1	Implantación
SV17	Implantación de datos	2	Implantación

Cuadro 3.2: Pila del producto inicial

sobre los 122,50 € al mes que si lo multiplicamos por los 4 meses de prácticas, nos sale 490 €.

Resumiendo, el precio por todo el software usado es de **490 €**.

3.3.3. Coste Hardware

Por lo que respecta al hardware, podemos encontrar los siguientes equipos.

- **Servidor**: tiene un precio de 6 €/mes, lo que supone un coste en los 4 meses de duración de 24 €. Se trata de un servidor con linux instalado para alojar el sistema que se está desarrollando.
- **Ordenador de trabajo**: se trata de un Lenovo con 8GB de RAM, memoria SSD de 256GB, procesador i3 de 2,2GHz con un precio de 515 €. Si tenemos en cuenta que la vida de un portátil con estas prestaciones es de 4 años, podemos calcular el precio relativo a los 4 meses que ha durado el proyecto, siendo **515/4/12x4=42,92 €**. Este ha sido el equipo de desarrollo del proyecto.

En resumen, el coste del hardware ha supuesto un coste de **66,92 €**.

Pila del producto final			
Código	Historias de Usuario	P.H.	Épica
SVO1	Registro de cliente	1	Login
SVO2	Inicio de sesión	1	Login
SVO3	Sistema de Punto de Venta	2	Ventas
SVO4	Sistema de Gestión de Compras	3	Compras
SVO5	Sistema de Gestión de Ventas	3	Ventas
SVO6	Sistema de almacenaje y logística	2	*Stock*
SVO7	Diseño de la tienda *online*	1	Diseño
SVO8	Desarrollo de la tienda *online*	4	Ventas
SVO9	Sistema de envío masivo de correo	2	Comunicación
SV10	Compra de un articulo	3	Compras
SV11	Formación de Odoo del desarrollador	2	Formación
SV12	Formación de Odoo al cliente	3	Formación
SV13	Toma de requisitos adicionales	3	Diseño
SV14	Capacitar varios canales de comunicación	5	Comunicación
SV15	Desarrollar un módulo de packs de artículos	5	Ventas
SV16	Implantación del Sistema ERP	1	Implantación
SV17	Implantación de datos	2	Implantación
SV18	Informes de *stock*	1	*Stock*
SV19	Informes de las ventas	1	Ventas
SV20	Preparar impresora de recibos	2	Ventas
SV21	Restringir acceso de usuarios	2	Login
SV22	Sistema de abastecimiento interno	3	*Stock*
SV23	Inclusión de stock en *e-commerce*	1	Ventas
SV24	Categorías desplegables *e-commerce*	2	*Stock*

Cuadro 3.3: Pila del producto final

3.3.4. Costes indirectos

Aparte de lo mencionado, hay que tener en cuenta la luz, el internet y el agua. Esto se hace llamar costes indirectos, que se traducen de forma genérica en un 20 % de los costes de personal (2100 €).

Quedando un coste total de **420 €** por costes indirectos.

3.4. Seguimiento del proyecto

Por lo que respecta al tiempo que se ha empleado para cada sprint, ha sido lo esperado, a pesar de que las historias de usuario SV09 y SV14 no se han terminado de desarrollar, porque no ha dado tiempo, se le han dado al cliente las indicaciones para poder hacerlo con el estándar de Odoo. Como hemos podido ver en la tabla 3.3, relativa a la pila del producto final, se añadieron varias historias de usuario, esta inclusión se dio en el *Sprint 2*.

3.4.1. Sprint 1

Para este primer sprint elegí solo la historia de usuario SV11, puesto que era mi primer contacto con el programa.

Así pues durante el trascurso de este sprint se me enseñaron las funcionalidades básicas de Odoo y la forma de trabajo de la empresa, junto con la forma de colaborar en la OCA. Una vez visto todo esto realicé mis dos primeros módulos simples.

Uno de estos módulos complementa la realización de informes de inventario. Cuando se realiza un inventario con la aplicación base, esta nos incluye todos los productos siempre y cuando estos tengan existencias en almacén. Sabiendo esto, se realizó un módulo que marcando una opción nos incluye en el inventario los productos que no tienen existencias.

El otro módulo que realicé nos da la opción de incluir una ruta de pedido global, puesto que si hay 50 productos en una factura, hay que seleccionar la ruta de uno en uno. Con este módulo se añade un campo en la cabecera del pedido que nos permite seleccionar una ruta global.

3.4.2. Sprint 2

Durante este sprint terminé con la historia de usuario SV11, y además realicé la reunión con el cliente para conocerlo y conocer de forma más especifica el proyecto relativo a la historia de usuario SV13.

Para terminar con la historia de usuario SV11, realicé un módulo por el cual no se permite a los usuarios devolver más cantidad de la que disponen actualmente. Es decir que si compran 10 unidades y reciben 8, pero tratan de devolver 10 o más no se permita al sistema realizarlo.

Por lo que respecta a la historia SV13, se realizó una reunión con el cliente en la que el cliente nos presentó todo lo que quería para la aplicación, y como se presentaron más historias de las que se me dieron a mi en un principio, se incluyeron en la pila del producto las adicionales.

3.4.3. Sprint 3

Como el cliente ya disponía de una base de datos creada del sistema, pensé que durante este sprint podía realizar las historias de usuario SV16, SV17 y desarrollar el módulo relativo a la historia SV23.

En primer lugar, se contrató un servidor en Hetzner para alojar el proyecto y se vinculó con el proyecto de GitLab (esta tarea la realizó un compañero, puesto que no estaba dentro del área de mi proyecto). Para continuar, se montó la estructura del proyecto mediante el uso de un entorno ya realizado que hace uso de *docker* para recoger los repositorios de GitHub y montar el sistema. Una vez montada la estructura del proyecto, se descargó la base de datos antigua del cliente y encontramos muchos errores relativos a los temas de la página web. Finalmente conseguimos arreglarlo dejando solo activas las vistas del tema por defecto.

Una vez realizadas las historias SV16 y SV17 pasé a empezar el módulo que muestra la disponibilidad en almacén de un producto que está en la página web. Pero casi no me dio tiempo a hacer nada, porque empezó el estado de alerta y era una tecnología que no tenía muy tratada, al ser este un módulo que requería de JavaScript.

3.4.4. Sprint 4

Al empezar este sprint tuve un dilema, por el que me planteé que debía hacer, si elegir solo el módulo que ya había empezado o ponerme una historia más. Al final decidí hacer ponerme como objetivo acabar el módulo que ya había empezado puesto que sabía que podía hacerse un poco cuesta arriba.

Para realizarlo hice uso de JavaScript junto con un controlador hecho con Python, de forma que se accede a la cantidades que tenemos almacenadas sin repercutir en la eficiencia de la página. Esto se debe a que el código JavaScript que se ejecuta del lado del cliente hace una petición get al controlador que le devuelve los datos que necesita para renderizar la vista XML.

Cómo entender todo este proceso, y saber ejecutarlo me llevó bastante tiempo. Fue una buena decisión escoger solo esta historia para este sprint, pero ya sabía que los próximos iban a ser más sencillos, puesto que ya tenía algo de rodaje y se vio un aumento en mi productividad.

3.4.5. Sprint 5

En este sprint decidí abarcar más historias de usuario, puesto que todas ellas estaban cubiertas por el programa principal. A pesar de estar realizadas ya en Odoo, hay que realizar un trabajo de investigación para ajustar el estándar a lo que el cliente nos pide, y poder explicar después, en la formación, de forma correcta cada paso al cliente. Las historias que he realizado son las siguientes: SV02, SV03, SV04, SV05, SV06, SV10, SV18 y SV19.

Al ser Odoo un ERP online, ya lleva incluido un sistema de autenticación para los usuarios, por la historia de usuario SV02, no resultó ningún tipo de problema. Así pues, como el cliente tiene tiendas franquiciadas en distintos lados de Madrid y Toledo había que configurar los puntos de venta de las tiendas, como se marca en la historia de usuario SV03, para que se comuniquen con su respectivo almacén. Para ello instalé el modulo pos de Odoo y en la configuración de los diarios de cada punto de venta, se configuró el almacén pertinente. Aparte de esto, como se trata de tiendas españolas, se tienen que configurar a la legislación del país. Así pues se migró un modulo de la versión 12 a la 13 para que al hacer una venta desde el punto de venta nos saque una factura simplificada, siempre que la cantidad no supere los 400 €, en caso contrario saca la factura de la venta.

Las siguientes cuatro historias dependen una de otra, por lo que se tomaron las tres un poco al mismo tiempo. Así pues, las historias SV04, SV05, SV06 y SV10 se van a explicar un poco a la vez. Para empezar para que hayan unidades de productos en el almacén, has de producir o comprar. Como la empresa Street Vape One se dedica a la compra/venta, para llenar el almacén definido en los ajustes hay que realizar compras. Una vez se realiza una compra de uno o varios

productos, se genera un pedido que cuando recibamos los productos en nuestro almacén hay que validar. Una vez el pedido validado, podemos ver que en nuestro almacén se han añadido las unidades que hemos validado. Una vez tenemos productos en el almacén, podemos realizar ventas, al realizar un presupuesto de venta, se genera un envío que cuando se valida se restan las unidades almacenadas de los productos enviados.

Por lo que respecta a los informes relativos a las historias SV18 y SV19 desde la aplicación inventario de Odoo podemos realizar un ajuste de inventario, que al finalizarlo nos saca el informe del almacén que disponemos. Para el informe de ventas, desde la aplicación de punto de venta podemos sacar informes mensuales de lo vendido tanto con tarjeta como con efectivo.

3.4.6. Sprint 6

Para este sprint me propuse realizar la historia SV01. Desde Odoo la gente se puede registrar, pero no dispone de seguridad para evitar que la gente que use correos fraudulentos. Para evitar esto existe un módulo por el que solo te deja terminar de registrarte en caso que recibas un enlace en el correo electrónico. Cuando empecé a realizar esta historia el modulo no estaba migrado, pero antes que llegara a ponerme a migrar el módulo, se subió al GitHub un PR(*Pull Request*) en el que se migraba a versión 13. Al pasar esto me dispuse a revisar el dicho PR y una vez corregido se incluyó al proyecto. En esta fase se ha visto la metodología DRY perfectamente.

Al suceder lo mencionado, pasé a realizar la historia SV22 que consiste en capacitar a los franquiciados la capacidad de realizar compras entre las compañías. Como Odoo no da una forma de realizar esto, se pasó a usar dos módulos ya migrados que nos permiten realizar compras entre distintas compañías. Al incluirlos al proyecto me di cuenta que el módulo que incluye la facturación entre compañías tenía un error. Para corregir el error, me dispuse a crear un PR para que se corrija y que nadie que vaya a usar el módulo tenga el mismo error que yo tuve. Una vez esto arreglado estudié la mecánica de trabajo para el abastecimiento entre las compañías que me ocupó lo que quedaba de sprint.

3.4.7. Sprint 7

Durante este sprint se han tratado las historias SV07, SV08, SV15, SV20, SV21, SV24. Estas historias las he dejado para el final, porque como sabía que se podían tratar de cierta forma sin necesidad de realizar un módulo. Así pues, por lo que respecta a los packs de productos, se estan desarrollando en OCA una serie de módulos que incluyen los packs de productos, pero como ya hay gente trabajando en ellos y no les ha dado tiempo de realizarlo en el tiempo que ha durado el proyecto se ha buscado una alternativa que consiste en crear productos que sean el pack. Para esto hay que tener cierto cuidado y separar los productos que van a ser parte del pack y las cantidades necesarias para que en caso que se haga un pedido de ese producto o del pack, no interfieran el uno con el otro.

Por lo que respecta a la impresora de recibos, se le pidió al cliente desde el primer momento el modelo de la impresora que usa. A pesar de esto, no me la facilitó, por lo que le di la opción de configurar la impresora desde el navegador y configuré el punto de venta para que saque

automáticamente las facturas simplificadas como pdf y que se impriman por el navegador. Por otro lado para restringir el acceso de los empleados a las aplicaciones de gestión, hay que entrar al perfil del cliente y darle solo permiso de usuario de punto de venta, ya que solo van a realizar ventas en las tiendas.

El cliente ya tenía medio montada la página web mediante el asistente de Odoo, por lo que para cubrir las historias SV07 y SV08 se le ha dado acceso a todos los temas gratuitos de Odoo que son editables, por lo que él mismo puede modificar la web a su gusto. Finalmente, se le añadió una novedad de la versión 13 que nos incluye el mega-menú para crear las categorías desplegables que no ocupen tanto espacio en la pantalla, como si estuvieran en el lado izquierdo fijas.

3.4.8. Historias no acabados

Una vez acabados los sprints, nos quedaron dos historias de usuario que no eran prioritarias para el cliente y que para insertarlas al proyecto por parte de Odoo suponía un coste extra para el cliente y por tanto, este decidió descartar estas funcionalidades destinadas a la mensajería.

A pesar de esto conseguí un proyecto muy completo, con el que el cliente acabó muy satisfecho, por lo que mostró durante las sesiones de formación que se realizaron al terminar todos las historias mencionadas anteriormente.

Capítulo 4

Análisis y diseño del sistema

4.1. Análisis del sistema

Como se ha usado una metodología ágil, la herramienta que se ha usado para el análisis del sistema han sido las historias de usuario, que se pueden ver en el diagrama de casos de uso que se encuentra en la figura 4.1.

A continuación se van a presentar las historias que se han tratado durante todo el proyecto.

- SV01 - Registro de cliente

Como usuario Quiero registrarme Para poder estar identificado cuando realice una compra

- SV02 - Inicio de sesión

Como trabajador Quiero iniciar sesión Para poder acceder a las funcionalidades del sistema

- SV03 - Sistema de Punto de Venta

Como trabajador Quiero vender productos por el punto de venta Para que los clientes tengan los productos que desean

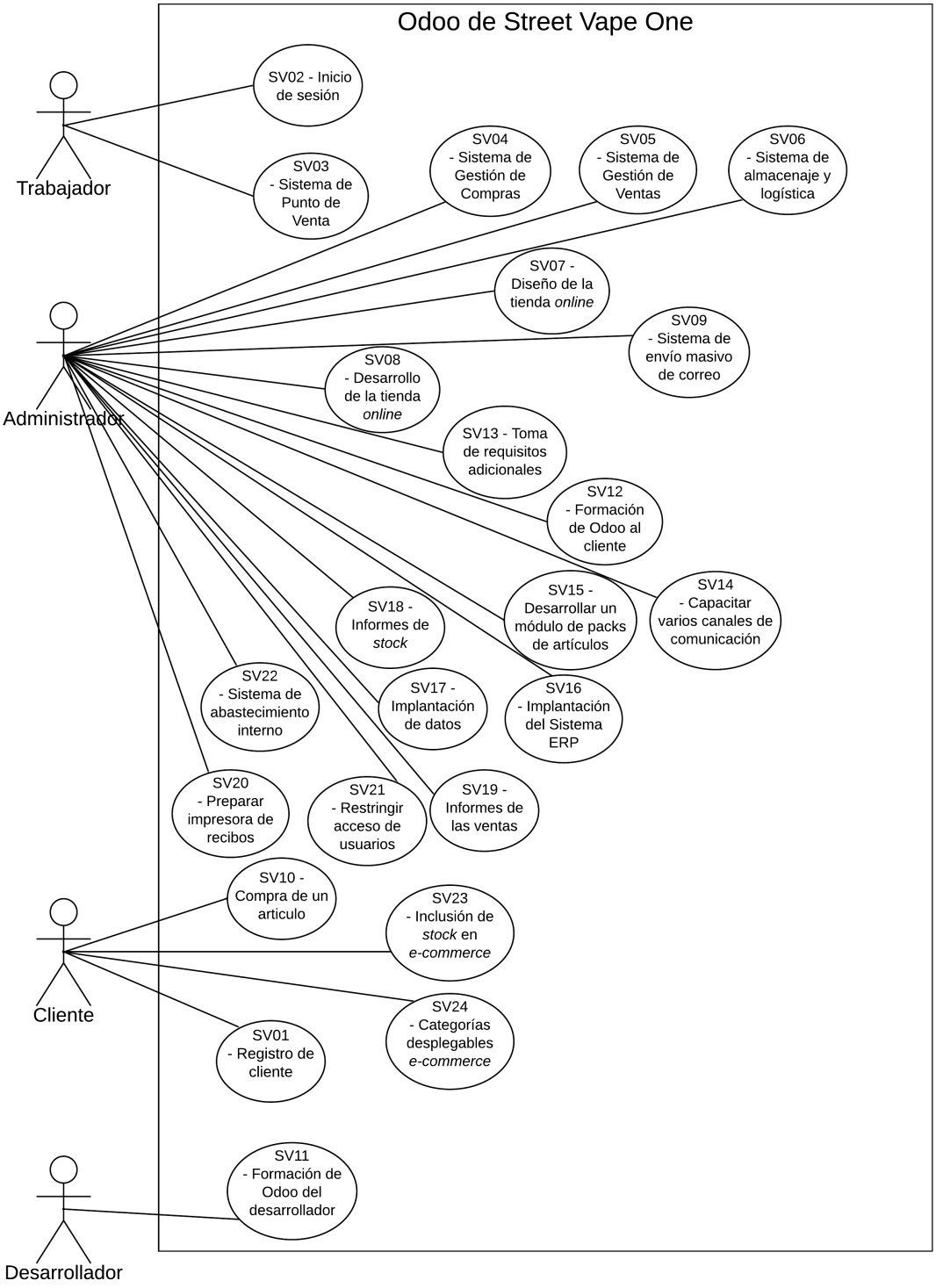

Figura 4.1: Diagrama de casos de uso de la aplicación

- SV04 - Sistema de Gestión de Compras

Como administrador Quiero comprar productos Para tener *stock* en el almacén

- SV05 - Sistema de Gestión de Ventas

Como administrador Quiero vender productos Para que el negocio funcione

- SV06 - Sistema de almacenaje y logística

Como administrador Quiero saber cuantos productos tengo en todo momento Para no quedarme sin *stock*

- SV07 - Diseño de la tienda *online*

Como administrador Quiero diseñar el estilo de mi página web Para que los usuarios naveguen a gusto

- SV08 - Desarrollo de la tienda *online*

Como administrador Quiero decidir que productos se van a vender en la tienda online Para que no se vendan los productos que solo estan disponibles en tienda

- SV09 - Sistema de envío masivo de correo

Como administrador Quiero avisar a mis clientes de las ofertas Para que estén informados de las últimas ofertas

- SV10 - Compra de un articulo

Como cliente Quiero comprar productos Para adquirir un producto que necesito

- SV11 - Formación de Odoo del desarrollador

Como desarrollador Quiero recibir una formación completa de Odoo Para saber cómo realizar las actividades y aconsejar al cliente

- SV12 - Formación de Odoo al cliente

Como administrador Quiero recibir una formación completa de Odoo Para saber usar la plataforma y enseñar a mis trabajadores

- SV13 - Toma de requisitos adicionales

Como administrador Quiero que me tomen otra vez los requisitos Para incluir funcionalidades que me parecen importantes

- SV14 - Capacitar varios canales de comunicación

Como administrador Quiero comunicarme con los trabajadores y clientes por varios canales Para poder estar siempre comunicados en caso de error

- SV15 - Desarrollar un módulo de packs de artículos

Como administrador Quiero varios productos se vendan juntos Para realizar ofertas que llamen la atención de nuevos clientes

- SV16 - Implantación del Sistema ERP

Como administrador Quiero que se instale el sistema en un servidor Para no depender de los servidores de Odoo

- SV17 - Implantación de datos

Como administrador Quiero que se conserven los datos de mi antigua base de datos Para no perder el trabajo que ya había realizado

- SV18 - Informes de *stock*

Como administrador Quiero realizar inventario del almacén Para ajustar las cantidades del almacén y tenerlo documentado

- SV19 - Informes de las ventas

Como administrador Quiero sacar informes de las ventas del PoS (*Point of Sale*) Para tenerlas documentadas

- SV20 - Preparar impresora de recibos

Como administrador Quiero que los recibos se impriman por la impresora que tengo Para no comprar una nueva

- SV21 - Restringir acceso de usuarios

Como administrador Quiero que mis trabajadores accedan solo al punto de venta Para evitar problemas de stock o con la contabilidad

- SV22 - Sistema de abastecimiento interno

Como administrador Quiero que las tiendas franquiciadas se abastezcan de la empresa principal Para que todas compren del mismo sitio

- SV23 - Inclusión de *stock* en *e-commerce*

Como cliente Quiero ver el stock de cada producto en la tienda online Para no intentar comprar algo de lo cual no hay stock

- SV24 - Categorías desplegables *e-commerce*

Como cliente Quiero que las categorías estén plegadas en la parte superior de la web Para poder ver más productos en la vista previa

4.2. Proceso de análisis

Al ser Odoo un sistema tan grande, es imposible explicar la base de datos ni el diagrama de clases. A pesar de ello no deja de haber un proceso de análisis para la realización de las tareas. Como hemos dicho anteriormente, se usa la metodología *Don't Repeat Yourself* que consiste en no volver a desarrollar aquello que ya se ha hecho para otros o por otro.

Esto supone una ventaja a la hora de desarrollar los módulos, pero sí que supone un proceso exhaustivo de búsqueda antes de desarrollar uno nuevo.

Para empezar con el módulo, primero tenemos que analizar qué va a hacer y cómo lo va a hacer. Una vez tenemos claro esto, tenemos que preguntarnos si esto se puede lograr de una forma distinta, en caso positivo, vamos a buscar que módulos cubren esa forma distinta de lograr nuestro objetivo. En caso que encontremos alguno en Odoo que nos valga, dejaríamos de buscar. En caso que este módulo no nos valga o se quede corto, deberíamos buscar en la OCA un módulo que nos valga. Después de la búsqueda por los repositorios de la comunidad, debemos plantearnos qué hacer. Para ello podemos tomar dos caminos.

El primer camino sucede cuando no hay nada que nos valga, en este caso pasamos a desarrollar un módulo que cubra nuestras necesidades. Cabe destacar que cuando hablamos de desarrollar, siempre va enfocado a los repositorios de la OCA, esto se debe a que igual que a nosotros nos ha hecho falta este módulo, a la larga puede hacerle falta a otros usuarios.

El segundo camino se da cuando encontramos algo que nos puede valer. En este caso, debemos de probar todos los casos de uso de la implementación que queremos hacer, para ver si realmente nos vale. En caso positivo, el proceso finalizaría aquí y este módulo se incluiría, en caso negativo, hay que decidir si alguna de las funcionalidades nos vale, en este caso, el nuevo módulo extendería de este, y nos ahorraríamos realizar lo que el padre ya hace. En caso contrario realizaríamos lo mismo que en el primer camino.

Para tratar de esquematizarlo, he realizado un diagrama de actividades con el proceso de análisis que se encuentra en la figura 4.2 y con los caminos que ha tomado cada historia de usuario en la tabla 4.1.

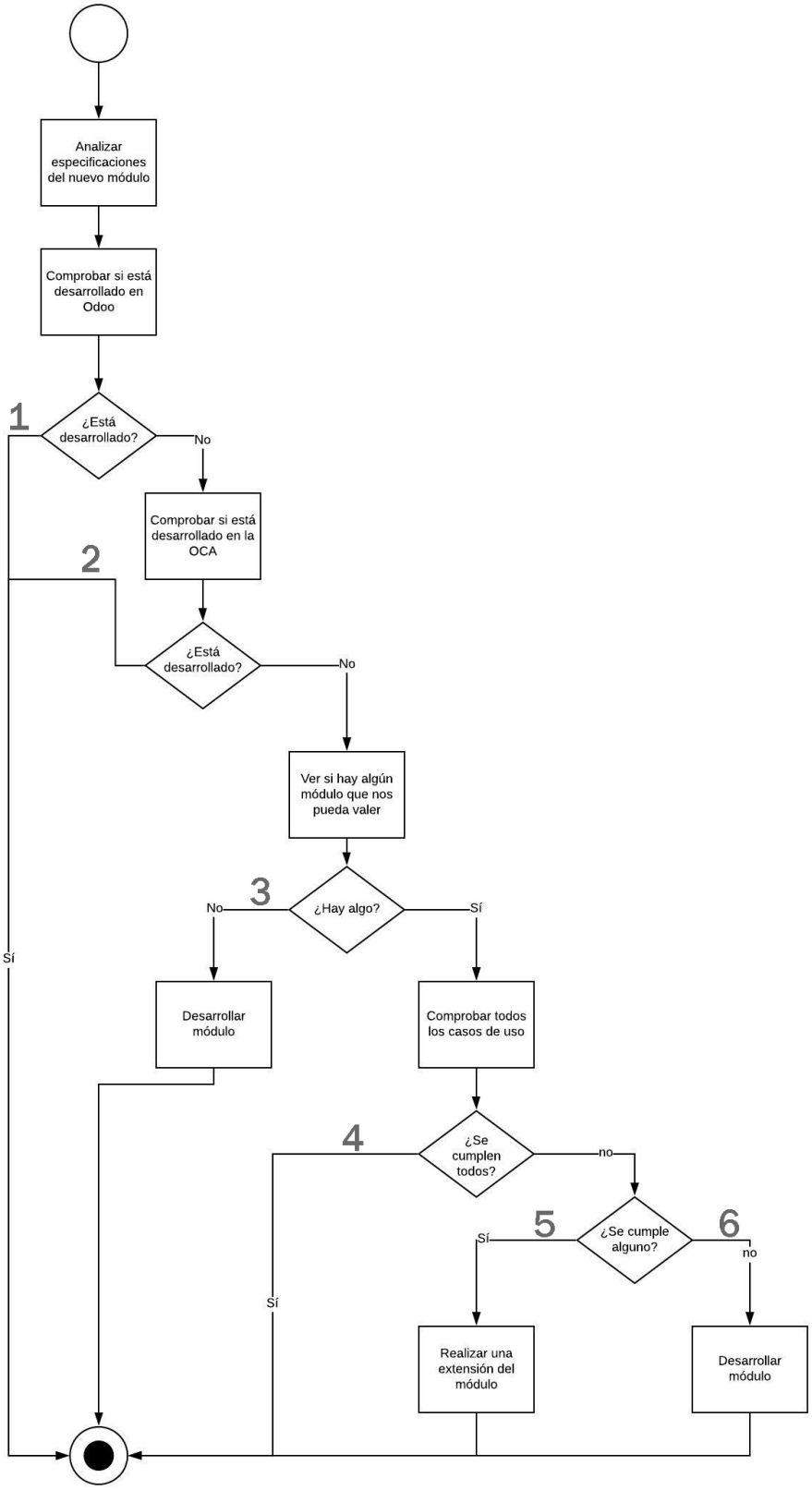

Figura 4.2: Diagrama de actividades del análisis de un módulo.

Caminos para desarrollar un módulo						
Código	**Cam. 1**	**Cam. 2**	**Cam. 3**	**Cam. 4**	**Cam. 5**	**Cam. 6**
SVO1	X					
SVO2		X				
SVO3	X					
SVO4	X					
SVO5					X	
SVO6	X					
SVO7	X					
SVO8	X					
SVO9					X	
SV10	X					
SV11						
SV12						
SV13						
SV14					X	
SV15				X		
SV16						
SV17						
SV18	X					
SV19	X					
SV20						X
SV21	X					
SV22					X	
SV23			X			
SV24	X					

Cuadro 4.1: Caminos que se han tomado para desarrollar un módulo de las historias de usuario

Como podemos ver hay muchas historias de usuario que ya están desarrolladas, pero esto no significa que no haya que hacer nada. Esto se debe a que hay que poner las configuraciones pertinentes para que se ajusten a los gustos del cliente y que cumplan con la legislación del país. Por ejemplo, para realizar compras y ventas hay que tener en cuanta los impuestos del país en el que reside el cliente, en este caso, hay que configurarlo para que incluya el IVA (Impuesto sobre el Valor Añadido) al 21 % por bienes.

Por lo que respecta a la historia relativa a la gestión de ventas, en España existe una ley [7] por la que los recibos para este tipo de locales de menos de 400 € han de ser facturas simplificadas. Por lo que hay que desarrollar un módulo que nos automatice y facilite el proceso. Buscando por la OCA encontré uno que hacía exactamente esto, pero era para la versión 12, así que tuve que migrarlo.

Para cubrir las historias SV09 y SV14 hay que migrar dos módulos de la OCA de la versión 12. Para incluirlos al sistema, el cliente tiene que contratar el servicio de CRM (*Custormer Relationship Management*), pero como por el momento no le resulta una prioridad, se ha decidido dejarlos para el final, aunque por falta de tiempo, no se han llegado a migrar. Para la historia SV15 un colaborador de la OCA esta trabajando en estos módulos durante mi estancia

en prácticas, para cubrir la petición que requiere el cliente se le da una opción por el estándar que cubre la funcionalidad de forma bastante satisfactoria.

Para la impresora de recibos, existen módulos que cubren la funcionalidad que el cliente desea pero para otros modelos de impresora. Como nos ha dado la información relativa a la impresora muy tarde, hemos configurado el punto de venta para que imprima los recibos por el navegador, de forma que solo deberá conectar la impresora a este.

Para el sistema de abastecimiento interno, es decir, entre las compañías, ya existen módulos que se encargan de ello en la OCA, pero uno de ellos necesita una mejora puesto que al mandar una factura que ya existía en una empresa franquiciada, salta un error que hay que arreglar.

Como no existe ningún módulo que incluya el *stock* en la vista previa de los productos, ni en Odoo ni en la OCA, se ha desarrollado desde 0 para cubrir una necesidad de importancia para nuestro cliente.

4.3. Diseño de la aplicación

Como lo que se ha realizado ha sido extender la funcionalidad de una aplicación que está actualmente en el mercado, los diseños ya están predefinidos y por lo tanto debemos seguirlos para añadir las funcionalidades extra.

4.3.1. Diseño de la arquitectura

Odoo es ERP en la nube, por lo que tiene una lógica detrás de cliente-servidor. Esto se puede ver en la figura 4.3.

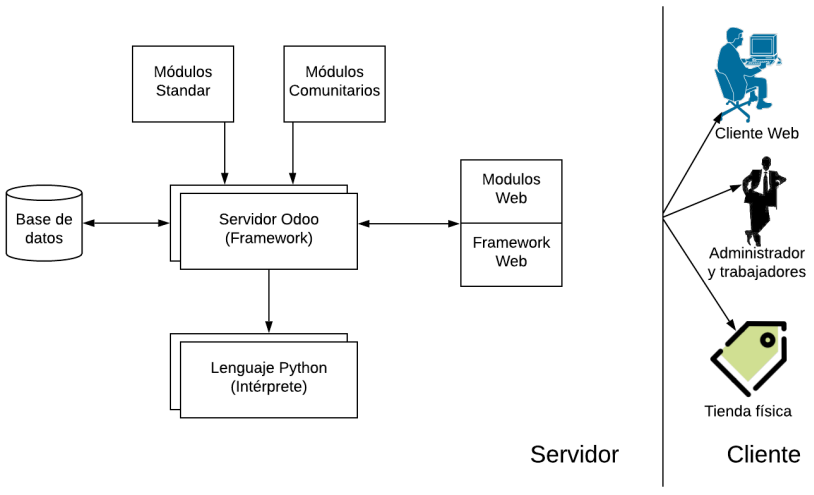

Figura 4.3: Arquitectura de Odoo. [8]

Para entender la imagen, del lado del servidor encontramos el servidor de Odoo, con sus respectivas librerías. Este servidor está formado por los módulos estándar y por los módulos que se hayan instalado de la comunidad. A su vez este servidor necesita comunicarse con la base de datos para que los cambios que se realizan en los datos tengan consistencia. Además existen los módulos web con su respectivo *framework*, que son los que se ven por los trabajadores o clientes de la aplicación. Todo esto lo interpreta Python, que es el lenguaje principal que se usa.

En la parte del cliente, existen varios usuarios que mediante el navegador web se comunican con el servidor para usar los servicios. Estos clientes, serán tanto los usuarios que se conectarán para comprar desde la tienda *online*, las tiendas físicas y el administrador y trabajadores.

4.3.2. Modelo-Vista-Controlador

Como se verá en la implementación, Odoo sigue el patrón MVC (*Model-View-Controller*). Se definen distintos modelos por módulo que incluyen referencias a la base de datos. Las vistas se alimentan de la información que ofrece el modelo.

Por lo que respecta al controlador, se encarga de procesar los datos para las vistas del *frontend*, mediante peticiones REST (*Representational State Transfer*). Para el *backend*, el procesado de datos se ubica en el modelo.

4.3.3. Diseño de la base de datos

Al tratarse de una aplicación tan extensa, existen muchas tablas dentro de ella para gestionar cada apartado de la aplicación. Actualmente al crear una base de datos desde cero, encontramos que la base de datos dispone de 111 tablas. De la misma forma al añadir módulos esta base de datos se va ampliando en caso que se creen clases no guardadas, en caso contrario se extenderían las tablas incluyendo columnas para los campos nuevos.

Para entender de forma centralizada en un caso la disposición de la base de datos, voy a proponer un ejemplo. Imaginamos que queremos hacer un módulo que incluya por cada producto que aparece en la tienda *online* los atributos seleccionables cuando se accede a la ficha del producto. Para ello, se va a extender el módulo relativo a la tienda *online*, pero este nuevo módulo va a usar las tablas relativas a los productos, en concreto, las relativas a *product_template* y a *product_attribute*, que estas a su vez pueden depender de otras tablas. Además de esto, es posible que se necesite crear una tabla para guardar cierta información. Para ver de forma gráfica a lo que me refiero, podemos ver la disposición de tablas madre e hijas de este ejemplo en la figura 4.4.

Tras ver esto, podemos ver que la base de datos va ampliándose conforme se van añadiendo módulos que creen tablas. Por lo que al terminar con el proyecto, podemos ver que disponemos de 485 tablas en total.

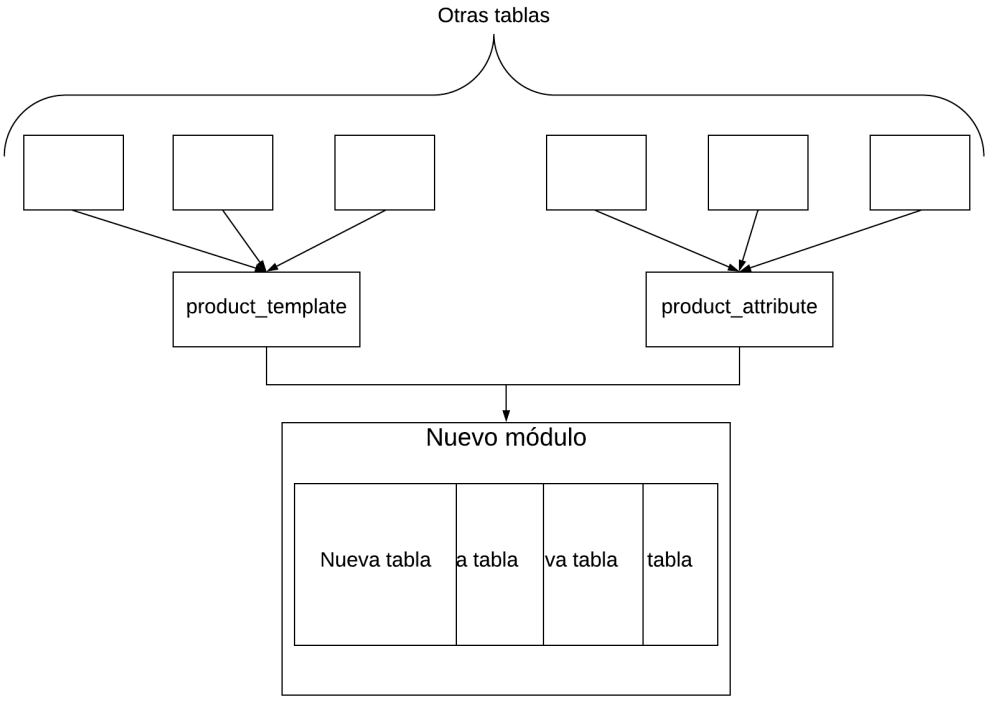

Figura 4.4: Esquema para un nuevo módulo en la base de datos.

4.3.4. Diseño de las interfaces de usuario

Por lo que se refiere a las vistas del proyecto, podemos distinguir entre vistas de *frontend*, relativas a la página web, y vistas de *backend* relativas al ERP de gestión.

Vistas *frontend*

Las vistas del *frontend*, son las vistas que ven los clientes de Street Vape One, por lo que son las vistas relativas a la página web. Estas vistas son editables por el administrador del sistema, de forma que puede arrastrar elementos y colocarlos donde le resulte más conveniente como se ve en la figura 4.5.

Además de esto, se aloja la tienda *online*, desde la que se pueden ver los productos que están publicados, en caso de ser un cliente de la plataforma, esto se puede ver en la figura 4.6. En caso de ser administrador, se pueden ver tanto los publicados como los que no lo están, esto se puede ver en la figura 4.7.

Además de la página web, encontramos en el *frontend* las vistas relativas al punto de venta que tienen un aspecto fijo, aunque se puede elegir como mostrar los productos desde el *backend*. En la figura 4.8 se puede ver el aspecto del punto de venta.

Figura 4.5: Forma de edición web por bloques.

Figura 4.6: Vista de cliente de la tienda *online* con un producto publicado.

Vistas *backend*

Dentro del *backend* podemos distinguir varios tipos de vistas, pero los más usados son la vista *kanban*, la vista *form* y la vista *tree*.

Por lo que respeta a la vista *kanban*, tiene un estilo de tarjetas con imagen, es muy cómoda cuando no estás familiarizado con el sistema y hace la navegación más intuitiva. En la figura 4.9 se puede ver una muestra de este tipo de vista.

La vista *form* es la que se usa para las vistas de creación y edición de entradas del sistema, como el nombre indica es un formulario con los campos que hay que completar. En la figura 4.10 se puede ver una muestra de esta vista.

Figura 4.7: Vista de administrador de la tienda *online* con un producto publicado

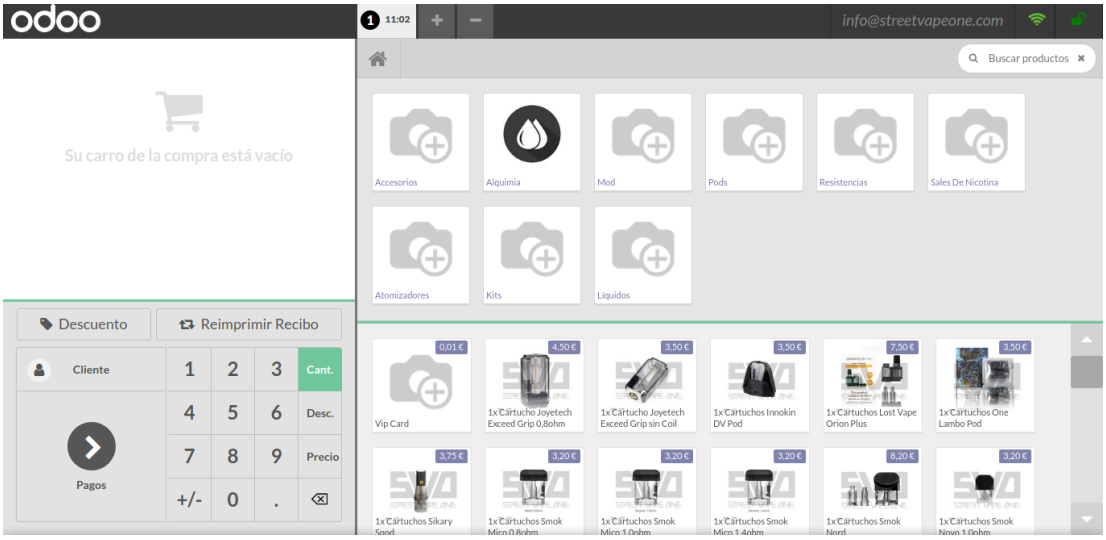

Figura 4.8: Vista del punto de venta de una tienda de Street Vape One.

Las vistas *tree* son las que listan una serie de elementos, sin uso de imágenes. Son vistas destinadas a usuarios expertos en Odoo, puesto que te ofrecen más información que las vistas *kanban*. En la figura 4.11 se puede ver un ejemplo de este tipo de vista.

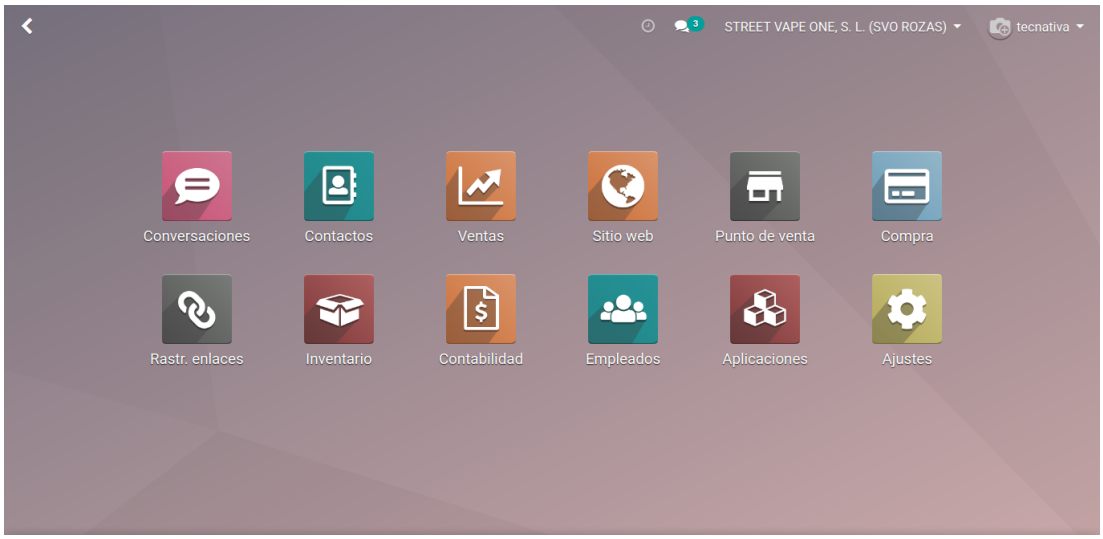

Figura 4.9: Vista *kanban* del menú principal de Odoo.

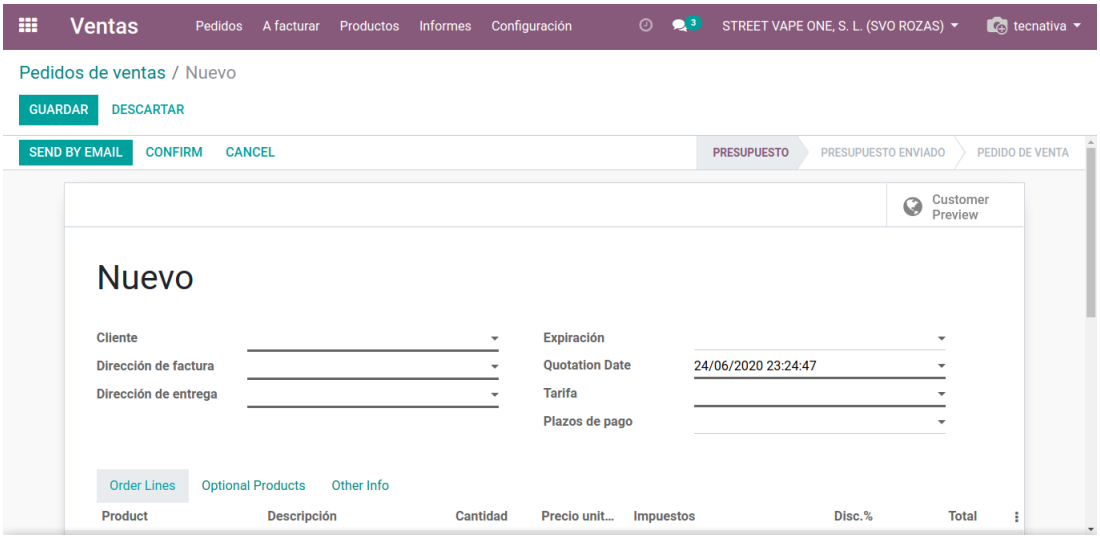

Figura 4.10: Vista *form* de creación de un presupuesto.

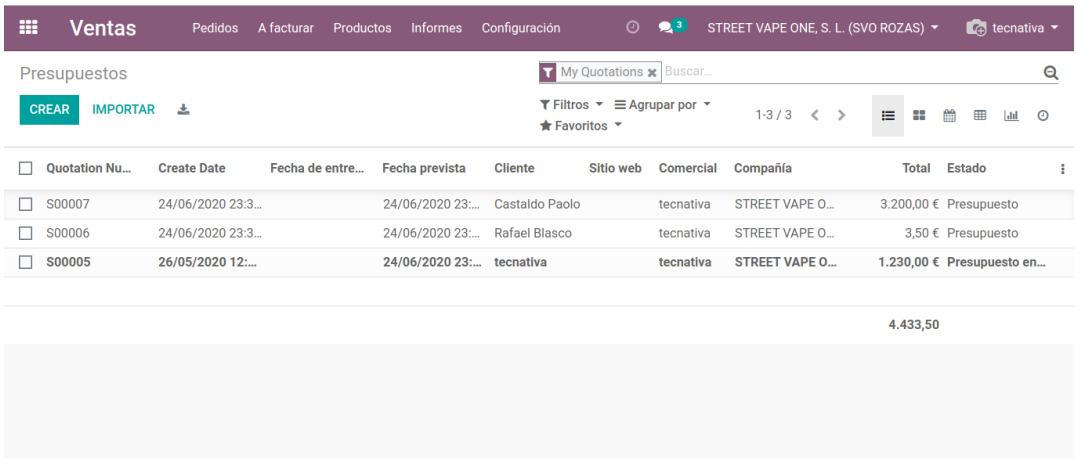

Figura 4.11: Vista *tree* del listado de presupuestos emitidos.

Capítulo 5

Implementación y pruebas

5.1. Estándar de Odoo

Como al cliente se le ha entregado un proyecto en el que está muy presente el estándar de Odoo, vamos a empezar explicando esto y a continuación se dará pie a los módulos que se han desarrollado, migrado o corregido. Así pues, pasaremos a ver cada uno de ellos en detalle.

Como hemos ido viendo durante el transcurso del documento, existe un estándar muy completo en Odoo que nos da mucho juego en el momento de resolver las necesidades de un cliente. Si es cierto que en esta parte no hay que programar, sin embargo, hay que conocerla en detalle para enfrentarnos a los módulos que tenemos que desarrollar.

5.1.1. Conceptos importantes

Aunque resulta lógico, Odoo está formado por muchos módulos. Estos módulos tienen la funcionalidad básica que ofrece Odoo de manera estándar. De esta forma, si queremos realizar un módulo customizado podemos recurrir a estos módulos para ver la estructura.

La estructura de los módulos sigue unas directrices. Aunque según lo que queremos realizar, podemos suprimir algunas de ellas. En la figura 5.1 podemos ver la estructura completa de un módulo de Odoo.

Para empezar, en el archivo `__manifest__.py` encontramos distintos apartados, pero los más importantes son los siguientes:

- **name**: Para indicar el nombre que va a tener el módulo.
- **version**: Indica la versión del módulo, para los módulos de la comunidad hay que empezarla con la versión de Odoo. Por ejemplo, para un módulo de la versión 13 en **version** encontraríamos 13.0.1.0.0.

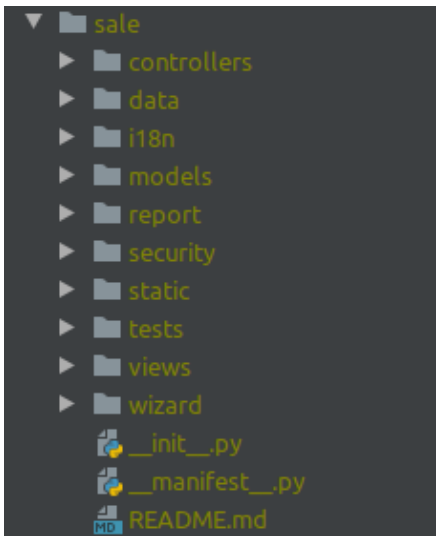

Figura 5.1: Estructura de archivos del módulo *sale* de Odoo.

- **category**: Categoría a la que pertenece dentro de la aplicación.
- **depends**: Se trata de una lista de los módulos de los cuales depende el módulo que desarrollamos.
- **data**: Se trata de una lista de las rutas relativas que ha de cargar el módulo.
- **installable**: Es un booleano que nos permite determinar si aparecerá en la lista de módulos para instalarlos.
- **auto_install**: Se trata de un booleano por el que se decide si se va a instalar el módulo automáticamente al instalar alguna de sus dependencias.

En la carpeta models, encontramos una serie de modelos que incluye un módulo. Aparte de los modelos, se incluye la lógica que se usa para dichos modelos de datos. De esta forma, se acopla el controlador con el modelo. Ahora nos podemos preguntar cual es la función del controlador. Pues la carpeta controllers, incluye la lógica para acceder desde el lado del cliente a los datos del servidor mediante peticiones REST.

La carpeta views, incluye los archivos XML para las vistas que se cargan al instalar el módulo. Desde este archivo, podemos acceder a los valores de los campos que encontramos en los modelos de datos sin necesidad de pasar por un controlador. En caso que se haga uso de JavaScript en este módulo, se incluye un archivo `assets.xml` que carga los archivos *.js. Los *wizard*, son vistas *pop-up*, es decir, ventanas que se ejecutan tras una acción. Si abrimos una carpeta **wizard**, encontramos tanto archivos Python como archivos XML, los primeros para la lógica y los segundos para la vista.

Por lo que respecta a la carpeta **static**, incluye archivos como los JavaScript, los CSS o imágenes que se usen para el módulo. A veces si se carga una vista desde el JavaScript, podemos encontrar también archivos XML. La carpeta test, como el nombre indica incluye los test que se realizan para comprobar que el módulo realiza lo que se desea. A veces hay que hacer test

de interfaz de usuario que se realizan mediante el uso de `tours.js` y como hemos visto, estos archivos se encuentran en la carpeta **static** que son llamados desde un test Python.

Hay algunas carpetas más en el sistema de archivos, pero no las he trabajado para hacer módulos así que prefiero no explicarlas.

5.2. Módulos para aprender

Como he dicho en la sección 3.4.1, para aprender a desarrollar en Odoo desarrollé tres módulos que a pesar que no aparezcan en el proyecto final, me parece importante mencionar cómo los hemos realizado, puesto que aprendí a desarrollar módulos con estos. Para probar que todo funciona bien se realizan test, pero no entraré en detalle para estos módulos puesto que están fuera del proyecto.

5.2.1. Módulo para incluir los productos agotados al ajuste de inventario

El motivo del desarrollo de este módulo, es que al realizar un ajuste de inventario no se incluyen los productos que están sin existencias. Como se ve en la figura 5.2 no encontramos ningún producto que esté en el programa fuera de *stock*. En caso que encontrásemos algún producto del que no tengamos unidades registradas en el ERP pero tengamos en el almacén, deberíamos añadirlo a mano.

Por lo que se propone que el módulo incluya una opción al crear un nuevo ajuste. Esta opción lo que hará será incluir los productos con cantidad 0 en el ajuste de inventario.

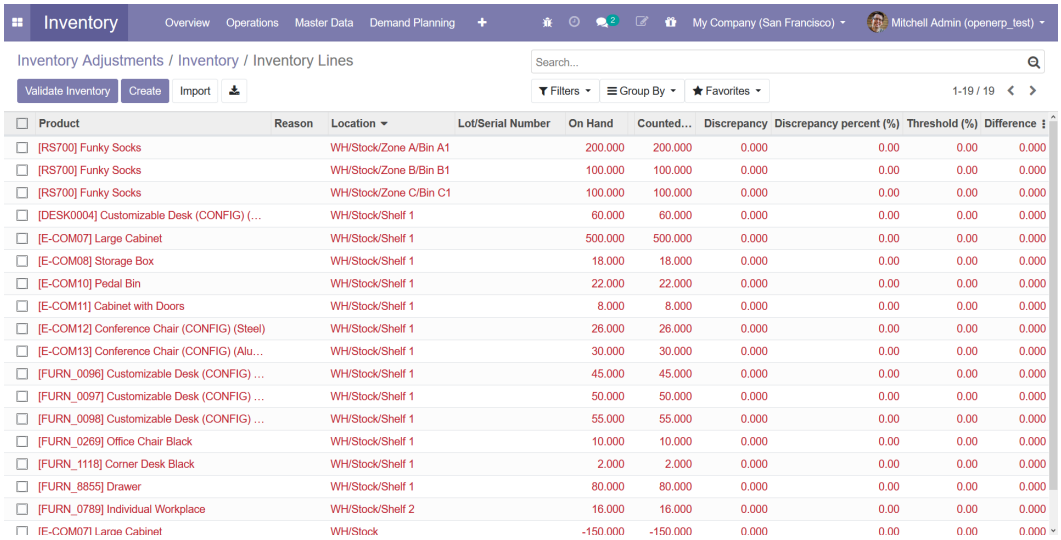

Figura 5.2: Ajuste de inventario estándar.

Para lograr esto, hemos hecho uso de la carpeta models y la carpeta views. En este caso

dependemos del módulo *stock*, puesto que lo que comprobamos y necesitamos son las cantidades de los productos. Dentro del módulo *stock* encontramos un modelo que hace referencia a los ajustes de inventario, por lo que vamos a extender la funcionalidad de este para el desarrollo del módulo.

Para decidir si se van a añadir los productos sin existencias, se va a añadir un campo al modelo *stock.inventory* de tipo *Boolean*. A continuación, se extenderá la función que crea las líneas del ajuste de inventario, para que añada, en caso que el campo que hemos añadido sea *True*, los productos que están fuera de *stock*. Para ver cómo crear un campo y cómo extender funciones, ver en el siguiente fragmento de código.

```
# Copyright 2020 Tecnativa - Carlos Roca
# License AGPL-3.0 or later (http://www.gnu.org/licenses/agpl.html).

from odoo import fields, models


class StockInventory(models.Model):
    _inherit = "stock.inventory"

    include_exhausted = fields.Boolean(
        string="Include Exhausted",
        help="If you select this option, you will receive the "
        "out of stock inventory products",
    )

    def _get_inventory_lines_values(self):
        vals = super()._get_inventory_lines_values()

        if self.include_exhausted:
            domain = [("qty_available", "=", 0), ("type", "=", "product")]
            if self.product_ids:
                domain.append(("id", "in", self.product_ids.ids))
            exhausted_products = self.env["product.product"].search(domain)
            vals_dic = {
                "inventory_id": self.id,
                "company_id": self.company_id.id,
                "product_qty": 0,
                "theoretical_qty": 0,
            }
            vals_dic["location_id"] = (
                self.env["stock.warehouse"]
                .search([("company_id", "=", vals_dic["company_id"])], limit=1)
                .lot_stock_id.id
            )
            for product in exhausted_products:
                vals_dic["product_id"] = product.id
                vals_dic["product_uom_id"] = product.uom_id.id
```

```
                vals.append(vals_dic.copy())

        return vals
```

Mediante el uso de *_inherit* extendemos el modelo *stock.inventory*. Para crear campos definimos una variable de clase que sea igual a fields.[el tipo de datos, en este caso *Boolean*]. Para extender la función creamos una función con el mismo nombre y definimos dentro de ella una variable igual a la llamada a la función del modelo padre. Es muy importante que se haga esto último y que se devuelva siempre esta variable, de forma que no se rompa la funcionalidad que este módulo realiza.

Para que el usuario pueda cambiar el valor del campo, hay que añadir una vista que haga referencia al campo que hemos creado. Para ello extenderemos la vista que añade las opciones en la creación del ajuste de inventario. En el siguiente fragmento de código se puede ver como se realiza esto.

```
<?xml version="1.0" encoding="utf-8" ?>
<!-- Copyright 2020 Tecnativa - Carlos Roca -->
<odoo>
    <record id="view_inventory_form" model="ir.ui.view">
        <field name="name">
            Inventory form view - stock_inventory_include_exhausted extension
        </field>
        <field name="model">stock.inventory</field>
        <field name="inherit_id" ref="stock.view_inventory_form" />
        <field name="arch" type="xml">
            <field name="prefill_counted_quantity" position="after">
                <field name="include_exhausted" />
            </field>
        </field>
    </record>
</odoo>
```

Como podemos ver, se usa la misma nomenclatura para extender las vistas, en este caso extendemos la vista con id *view_inventory_form* que se encuentra en el módulo *stock*. Para añadir el campo, buscamos el campo *prefill_counted_quantity* y lo añadimos debajo suyo.

Finalmente, podemos ver el resultado en las figuras 5.3 y 5.4. Hay que tener en cuenta que en el *runbot* de Odoo están todos los módulos de un repositorio instalados, por lo que se han incluido campos que son de otros módulos, pero podemos ver que se han incluido líneas con 0 unidades en el almacén.

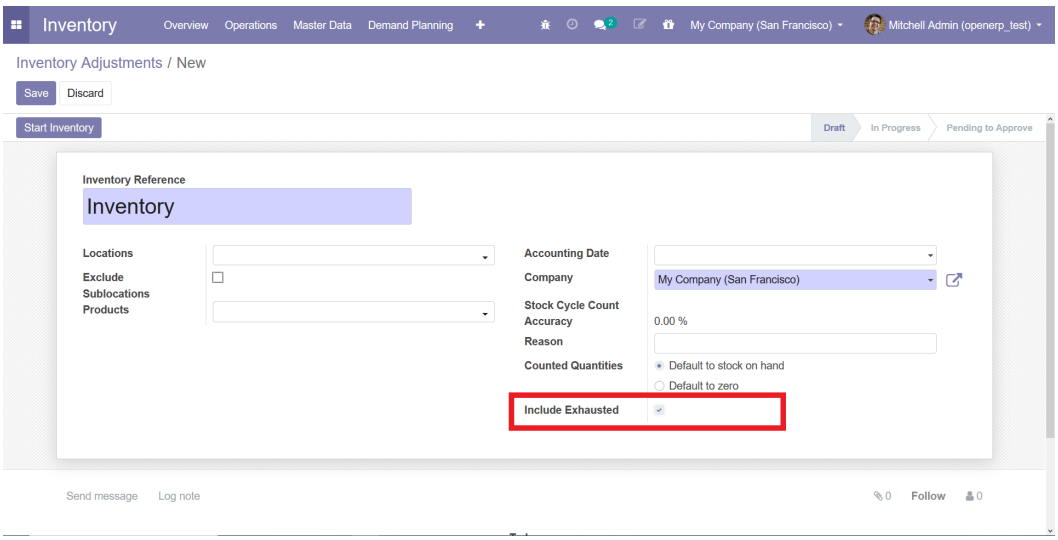

Figura 5.3: Pantalla de creación de un ajuste de inventario con la opción *Include exhausted*.

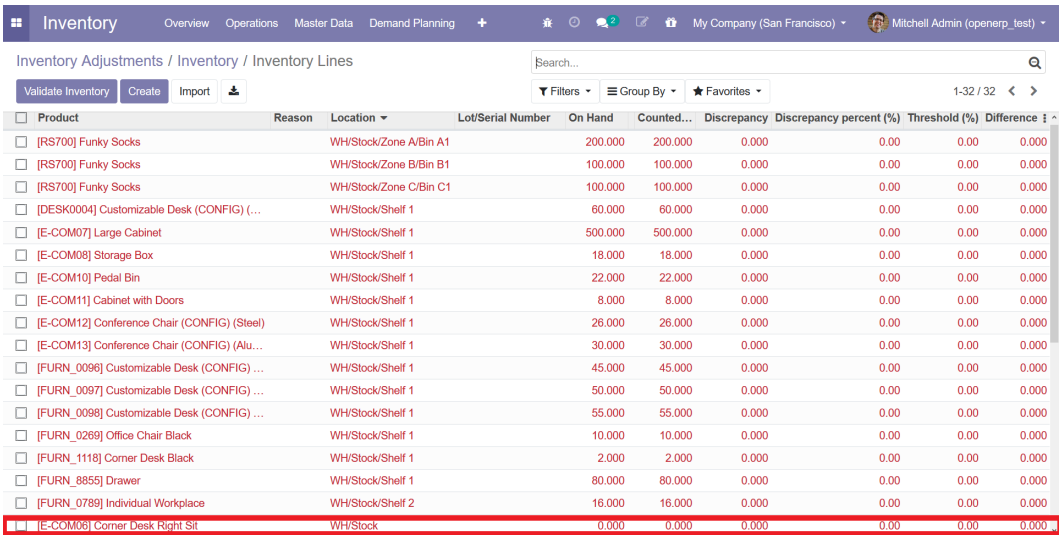

Figura 5.4: Ajuste de inventario con líneas que tienen cantidades registradas 0.

5.2.2. Módulo para incluir una ruta de pedido global

El motivo del desarrollo de este módulo es agilizar el cambio de ruta de cada producto, puesto que por defecto hay que marcar las rutas que deben tomar los productos, de una en una, por lo que si existe un pedido muy grande en el que todos los productos tomarán la misma ruta, puede resultar muy poco productivo. Por lo que se propone crear un módulo que añada un campo en la cabecera del presupuesto para que cambie todas las líneas de este indicando la ruta que ha de tomar de forma genérica.

Para solucionar esto, extenderemos los modelos *sale.order* y *sale.order.line*, de forma que añadamo un campo de tipo *Many2one* que se llame *route_id* en la cabecera. Una vez tenemos esto crearemos una función *onchange* que cambiará el valor del campo *route_id* de cada línea en *sale.order.line*. Para que los productos que se añaden después en las líneas de un presupuesto, habrá que crear otra función *onchange* que coja el valor del campo que hemos incluido en la cabecera.

Por lo que respecta a las vistas, simplemente tenemos que añadir el campo, *route_id* en la cabecera. De forma que el resultado es el que se muestra en las figuras 5.5 y 5.6.

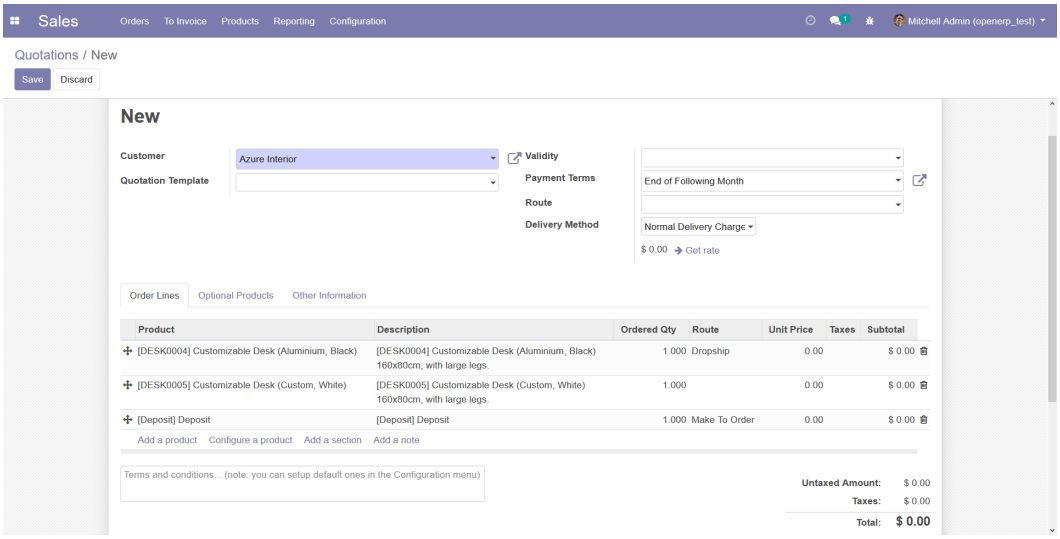

Figura 5.5: Presupuesto con varias líneas sin seleccionar la ruta de la cabecera.

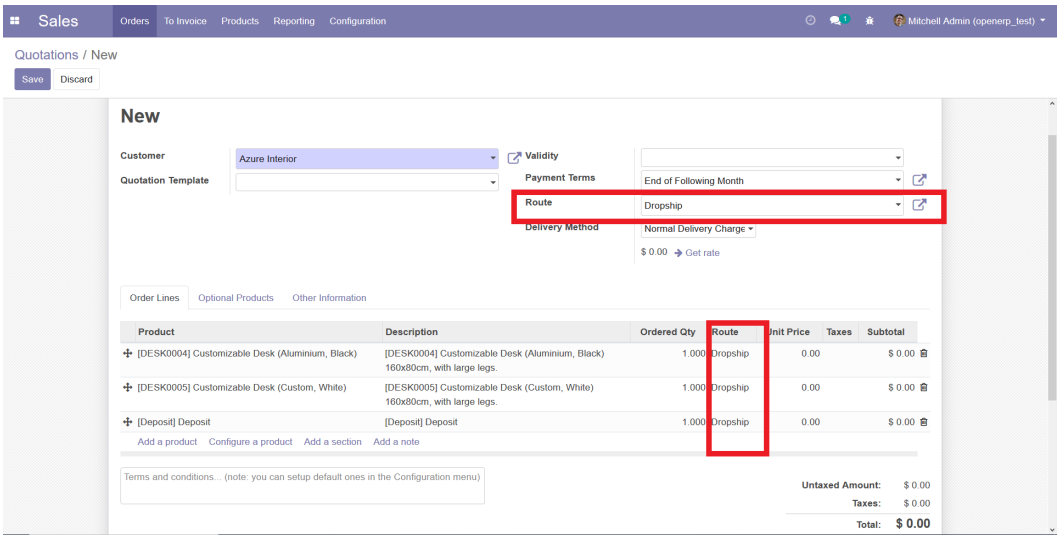

Figura 5.6: Presupuesto con varias líneas seleccionando la ruta de la cabecera.

5.2.3. Módulo que impida devolver más cantidad de la que se dispone

Por lo que se desarrolla este módulo es porque Odoo por defecto te permite crear devoluciones de más cantidad de la que dispones. Este módulo va a depender de `stock` y se va a sobrescribir la funcionalidad del *wizzard* para que salte una excepción al superar la cantidad que disponen los clientes.

Para lograrlo, se extienden los modelos del *wizzard* `stock.return.picking` y `stock.return.picking.line`. Por lo que respeta al primero de ellos, se extenderá una función que pone la cantidad inicial de devolución, por lo que vamos a poner siempre la máxima disponible. Además se extenderá una función que se ejecuta al pulsar el botón para que lance la excepción en caso que se supere el máximo. Por lo que respecta al otro modelo, se añadirá una función `onchange` que en caso que se supere el margen lance la excepción.

En este caso no hace falta extender ninguna vista, puesto que las excepciones se tratan de forma automática por Odoo. Por lo que podemos ver el resultado de las figuras 5.7 y 5.8.

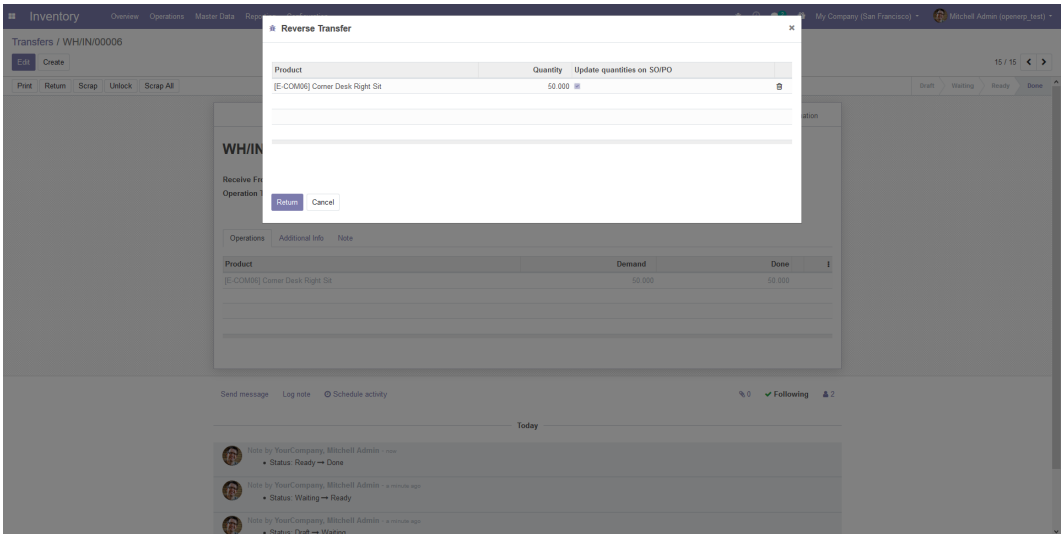

Figura 5.7: *Wizard* de devolución con la cantidad máxima disponible para devolver.

5.3. Módulo que muestra la disponibilidad en la vista previa

5.3.1. Objetivo

El motivo por el que se desarrolla este módulo, es porque en el comercio electrónico de Odoo solo aparece el *stock* de los productos una vez pulsas sobre él, esto se puede ver en las figuras 5.9 y 5.10. Por lo que se pretende incluir las cantidades existentes de cada producto en la página web, pero que no disminuya la eficiencia de la carga de página.

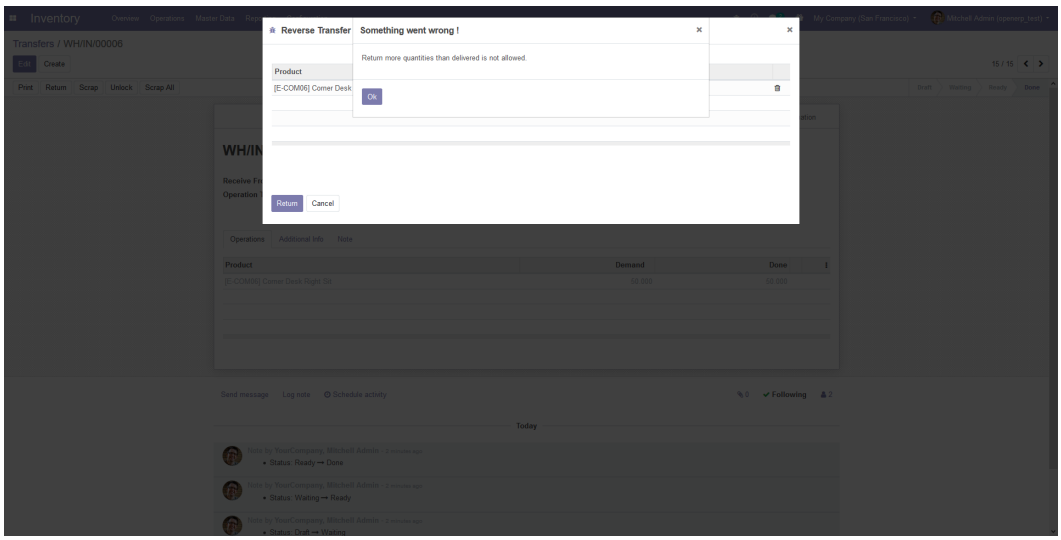

Figura 5.8: Excepción que salta al tratar devolver 53 unidades.

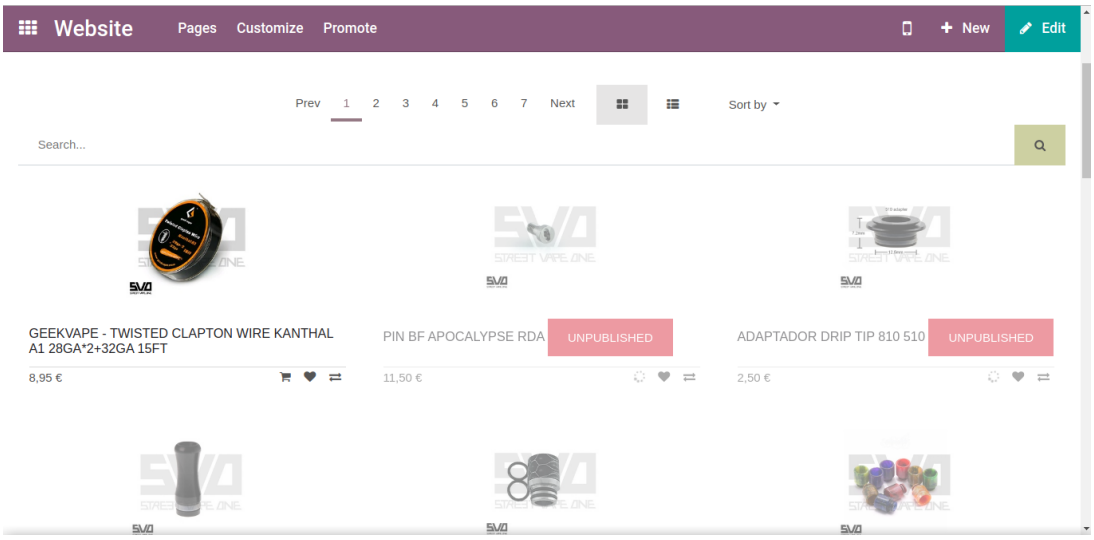

Figura 5.9: Vista previa de los productos de Odoo.

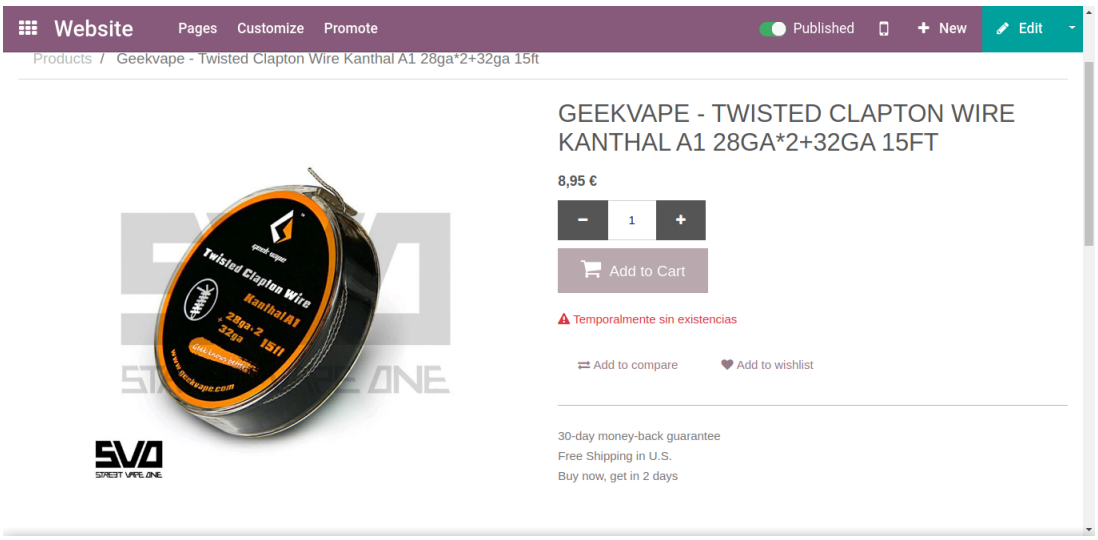

Figura 5.10: Vista del producto que no tiene *stock*.

Gracias a este módulo queremos evitar que los clientes del comercio electrónico vayan a comprar productos que no están disponibles, advirtiendo desde la vista previa de las existencias. El resultado final se puede ver en la figura 5.11

5.3.2. Desarrollo del módulo

Como queremos que el módulo no baje las prestaciones de la página, nos decantamos por que la vista que incluye el *stock* se cargue desde el lado del cliente. Por lo que nos decantamos por un módulo que use JavaScript junto con un controlador. Gracias a esto podremos procesar la cantidad de la que dispone cada producto de forma asíncrona, por lo que no tendremos ninguna disminución de la eficiencia respecto a la versión estándar de Odoo.

En primer lugar crearemos el controlador, al que se le pasarán los id de los productos que ha de calcular. A partir de esto, procesará cada campo para generar una lista de diccionarios que contendrán el identificador, la cantidad disponible y algunos campos que requiere la vista heredada.

```
# License AGPL-3.0 or later (https://www.gnu.org/licenses/agpl).

from odoo import http
from odoo.http import request

from odoo.addons.sale.controllers.variant import VariantController


class WebsiteSaleVariantController(VariantController):
    @http.route(
```

```
        ["/sale/get_combination_info_stock_preview"],
        type="json",
        auth="public",
        methods=["POST"],
        website=True,
    )
    def get_combination_info_stock_preview(self, product_template_ids, **kw):
        """Special route to use website logic in get_combination_info override.
        This route is called in JS by appending _website to the base route.
        """

        current_website = request.env["website"].get_current_website()
        res = []
        templates = (
            request.env["product.template"]
            .sudo()
            .with_context(warehouse=current_website.warehouse_id.id)
            .browse(product_template_ids)
        )
        for template in templates.filtered(lambda t: t.is_published):

            res.append(
                {
                    "id": template.id,
                    "virtual_available": template.virtual_available,
                    "inventory_availability": template.inventory_availability,
                    "available_threshold": template.available_threshold,
                    "custom_message": template.custom_message,
                    "type": template.type,
                    "uom_name": template.uom_name,
                }
            )
        return res
```

Como podemos ver tenemos una cabecera que indica la ruta, el tipo de llamada y el tipo de datos que devuelve, cabe destacar que se ha usado el método POST, puesto que la función que llama al controlador desde JavaScript realiza un POST, por lo que nos fuerza a usarlo.

Como ya hemos dicho, el controlador es llamado desde el JavaScript, por lo que vamos a ver en detalle como funciona cada una de las partes más importantes.

```
odoo.define("website_sale_stock_list_preview.shop_stock", function(require) {
    "use strict";

    var publicWidget = require("web.public.widget");
    var core = require("web.core");
```

```
publicWidget.registry.WebsiteSaleStockListPreview =
                                    publicWidget.Widget.extend({
    selector: "#products_grid",
    xmlDependencies: [
        "/website_sale_stock/static/src/xml/" +
            "website_sale_stock_product_availability.xml",
    ],

    start: function() {
        return $.when.apply($, [
            this._super.apply(this, arguments),
            this.render_stock(),
        ]);
    },
    render_stock: function() {
        const $products = $(".o_wsale_product_grid_wrapper");
        const product_dic = {};
        $products.each(function() {
            product_dic[
                this.querySelector("a img").src.split("/")[6]
                ] = this;
        });
        const product_ids = Object.keys(product_dic).map(Number);
        return this._rpc({
            route: "/sale/get_combination_info_stock_preview/",
            params: {product_template_ids: product_ids},
        }).then(products_qty => {
            for (const product of products_qty) {
                $(product_dic[product.id])
                    .find(".product_price")
                    .append(
                        $(
                            core.qweb.render(
                                "website_sale_stock.product_availability",
                                {
                                    virtual_available:
                                        product.virtual_available,
                                    inventory_availability:
                                        product.inventory_availability,
                                    available_threshold:
                                        product.available_threshold,
                                    custom_message: product.custom_message,
                                    product_template: product.id,
                                    product_type: product.type,
                                    uom_name: product.uom_name,
                                }
                            )
                        ).get(0)
                    );
```

```
                        // With this code we active just the
                        // products that can be sold on website.
                        if (
                            product.virtual_available <= 0 &&
                            (product.inventory_availability == "always" ||
                                product.inventory_availability == "threshold")
                        ) {
                            $(product_dic[product.id])
                                .find(".fa-spinner")
                                .addClass("d-none");
                            $(product_dic[product.id])
                                .find(".fa-shopping-cart")
                                .removeClass("d-none");
                        } else {
                            $(product_dic[product.id])
                                .find(".fa-spinner")
                                .addClass("d-none");
                            $(product_dic[product.id])
                                .find(".fa-shopping-cart")
                                .removeClass("d-none");
                            $(product_dic[product.id])
                                .find(".fa-shopping-cart")
                                .parent()
                                .removeClass("disabled");
                        }
                    }
                });
            },
        });
});
```

Para comzar se define una función en JavaScript que va a englobar todo nuestro código. Dentro de este se definen las dependencias, en este caso *publicWidget* y *core*. A partir de esto, extendemos la funcionalidad del *publicWidget*. Existen más *widgets*, pero este era el más indicado, puesto que nos permite definir un selector sobre el que se va a ejecutar, a esto se le suma que nos permite depender de vistas XML. La primera función que se ejecuta es la *start*, por lo que extendemos esta función y definimos una que tendrá la nueva lógica para la página web.

A continuación, haciendo uso de los selectores de JQuery se seleccionan los productos y guardamos el id de cada producto, que lo encontramos en las imágenes de los productos. Luego llamamos a *_rpc()* que hace la llamada a la ruta que le pasamos junto con los argumentos que deseamos, en este caso la lista de identificadores de productos. Con la respuesta de la llamada a *_rpc()*, construimos las vistas que incluyen el texto de información de las cantidades disponibles.

5.3.3. Dificultades

Como dificultades para este módulo, podría destacar la parte final del código anterior. Puesto que en un principio, no se tuvo en cuenta que en caso que se activase el botón de añadir al carro en la vista previa, los usuarios igual podían comprar. Por lo que se me reportó que debía cambiarlo. Para solucionarlo, se creo una vista que se carga con la instalación del módulo, la cual desactiva el botón y añade un símbolo de espera. Con el final del código, en función de la cantidad que queda, se activa el botón o se mantiene desactivado y se vuelve a mostrar el carro de la compra.

5.3.4. Verificación y validación

Para testear que el código funciona correctamente, se crean datos de prueba desde el archivo Python de test. La creación de los datos de prueba se definen en el setUp. Una vez cargados los datos de test, se lanza un tour test. El tour está escrito en JavaScript y mediante una lista de diccionarios y selectores CSS se marcan los pasos que se van a ejecutar para comprobar que todo está funcionando correctamente.

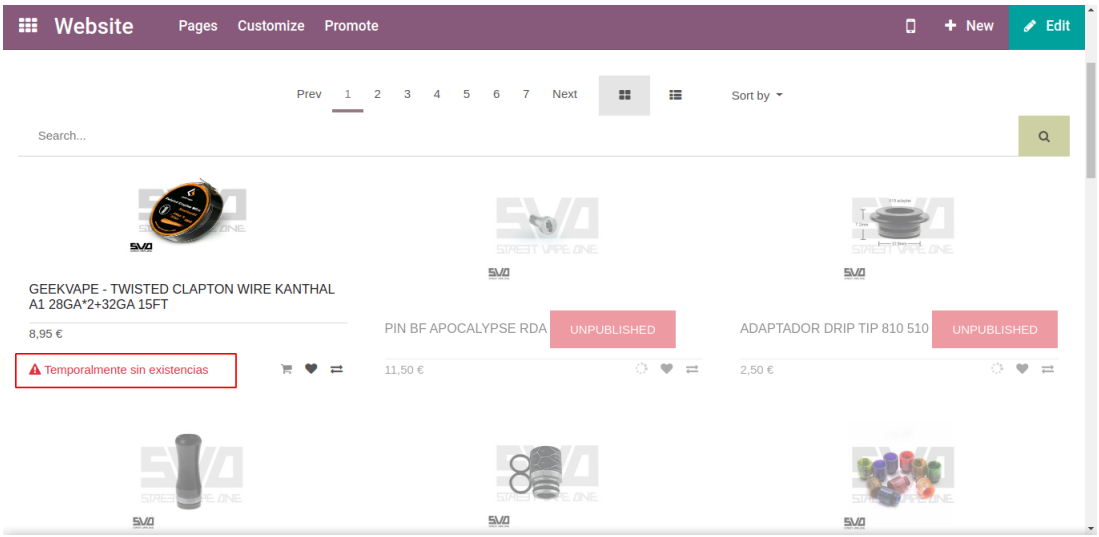

Figura 5.11: Vista previa de un producto que no tiene *stock*.

5.4. Módulo que ajusta el punto de venta a la legislación española

5.4.1. Objetivo

El objetivo de la migración de este módulo es adaptar el punto de venta a la legislación española. Por la que se dice que se deben sacar facturas simplificadas en caso que el importe de

una compra sea menor a 400 €. Del mismo modo, si se supera este importe se debe sacar una factura completa de la compra. En la figura 5.12 podemos ver el aspecto del recibo antes de desarrollar el módulo.

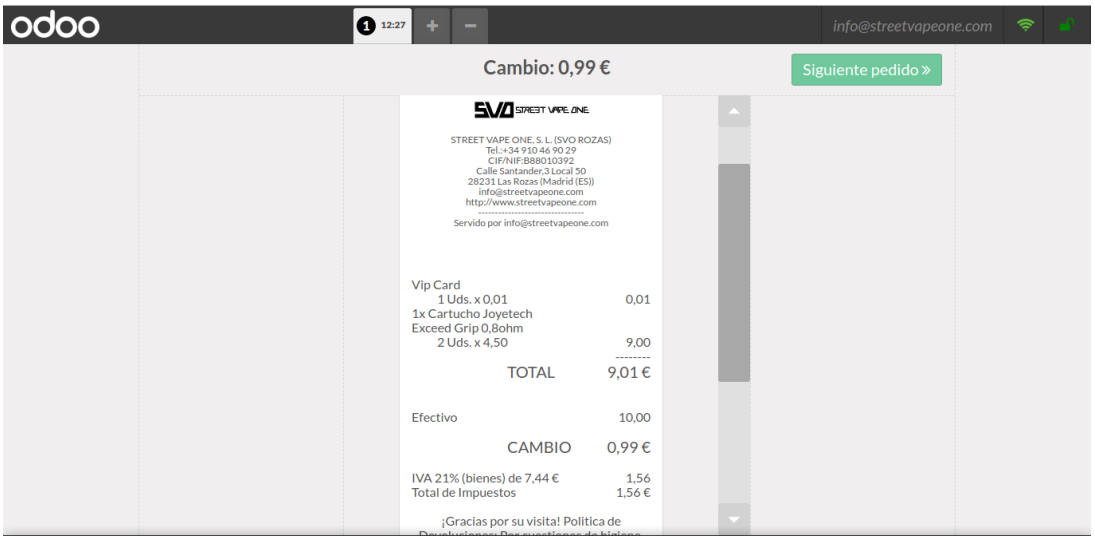

Figura 5.12: Emisión del recibo sin el módulo instalado.

Con el desarrollo de este módulo se espera que se registren las ventas con facturas simplificadas en el sistema, y que saque la factura simplificada en caso que no se supere el límite. Del mismo modo, si se supera el límite se quiere que se registre como factura y que la saque automáticamente. El resultado se puede ver en la figura 5.13, donde se puede ver que en el recibo se señala que es una factura simplificada.

5.4.2. Desarrollo del módulo

Al tratarse de una migración, hay que destacar que la lógica ya está prácticamente hecha, por lo que se parte de algo y lo que se realiza es ajustar el módulo a la nueva situación del estándar. Para aprender a hacer migraciones se ha seguido la guía se nos ofrece en la OCA [9]. Así pues, se han seguido una serie de pasos que se han de realizar siempre. A continuación hay que comprobar si se ha realizado algún cambio mayor, que nos rompa el código. En este caso, había algunas cosas, pero eran minucias, que tras depurar se resolvieron enseguida.

El cambio sustancial llega al empezar con las vistas, puesto que dos vistas que estaban separadas en la versión 12, en la versión 13 se fusionan y cambian algunos nombres de campos. Sabiendo esto, hay que fusionar las dos vistas para conseguir el mismo resultado que en la versión 12.

5.4.3. Dificultades

Las dificultades que he encontrado durante la realización de este módulo, no han sido muchas. Puede parecer que buscar los nuevos nombres es difícil, pero sabiendo el módulo del que se depende, es fácil buscar las nuevas nomenclaturas. Por lo que ha sido un módulo bastante sencillo.

5.4.4. Verificación y validación

En ente caso, los test ya estaban realizados, por lo que solo ha habido que corregirlos para que se adapte a los nuevos nombres de campos. Una vez ajustados los test, se ha pasado a corregir el módulo hasta conseguir pruebas exitosas.

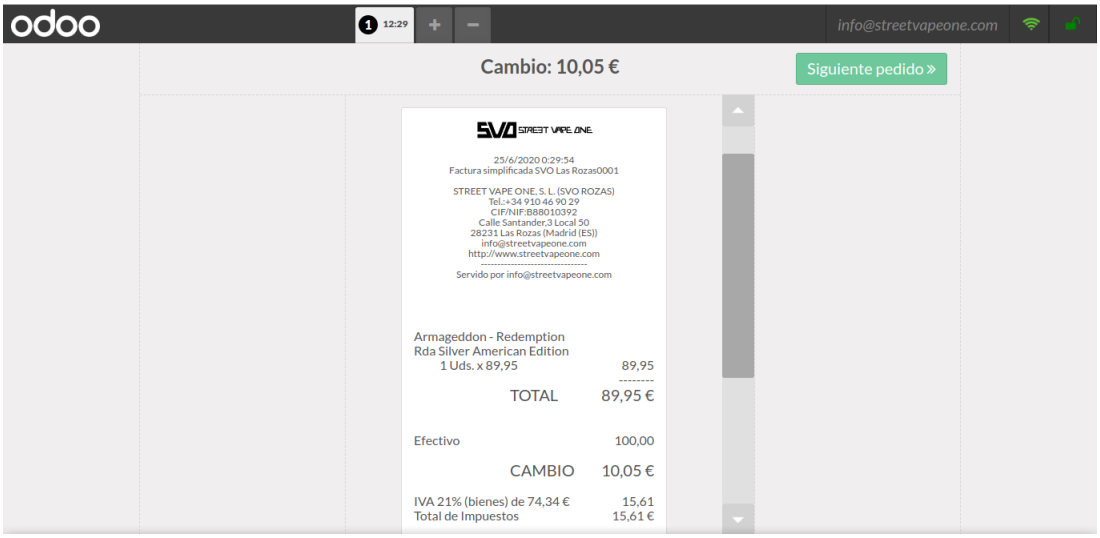

Figura 5.13: Emisión dela factura simplificada con el módulo instalado.

5.5. Módulo para sincronizar las facturas de las franquicias

Con la inclusión de este módulo y un módulo que incluye las ventas entre compañías, el cual depende de este, se ha conseguido mejorar una funcionalidad de Odoo "enterprise". Por lo que se ha sustituido el módulo "enterprise" relativo a esta funcionalidad por los desarrollados por la OCA.

5.5.1. Objetivo

El objetivo de este desarrollo es arreglar un fallo del módulo `account_invoice_inter_company`, por el que no se nos permite volver a enviar una factura que ya estaba creada en el destinatario.

5.5.2. Desarrollo del módulo

El desarrollo, fue difícil a pesar de solo cambiar dos líneas de código, puesto que tuve que buscar de donde surgía el error. Y una vez lo encontré, tuve que requerir ayuda de un compañero para me ayudase, puesto que no conocía como corregirlo.

Finalmente, lo que sucedía era que no existía el campo `move_name` puesto que ha cambiado por `name`. Pero a parte de esto tuvimos que añadir un contexto que forzara la eliminación en caso que la factura existiese.

5.5.3. Verificación y validación

Por lo que respecta a estos test, se añadió un fragmento de código que incluye la prueba de las líneas que contenían el *bug* de forma que se tenga en cuenta en migraciones futuras.

5.6. Pruebas y validación

Como ya se ha comentado, la OCA dispone de un repositorio en el que participan más de 900 colaboradores. Por lo que hay muchos ojos observando y muchas manos trabajando en módulos nuevos o migraciones de módulos antiguos. Para mantener un nivel de calidad alto en los módulos, se siguen unas directrices antes de incluirse un cambio al repositorio.

En primer lugar, los módulos que se incluyan deben tener una cobertura de código elevada, esto significa que se deben incluir test que demuestren que nuestro código está funcionando correctamente. A continuación, 3 colaboradores de la OCA deben revisar y dar el visto bueno al desarrollo que se realiza, de forma que un moderador del repositorio vea que está listo para incluirse.

5.6.1. Validación

Como ya comentamos, en GitLab tenemos el proyecto del cliente, este está vinculado al servidor contratado de forma que cuando se realice algún cambio se vea disponible en el servidor. De esta forma, el cliente puede seguir los cambios que se realizaban y juguetear en el entorno de demostración. Este entorno se construye a partir del entorno de producción, de forma que tengamos una réplica exacta al producto final donde hacer pruebas sin romper nada que sea de importancia.

Capítulo 6

Conclusiones

El desarrollo de este trabajo me ha valido para aprender de un mundo que desconocía, el de los ERP. Además me ha servido para ampliar mis conocimientos en Python. Estoy muy satisfecho con el trabajo realizado y con la empresa Tecnativa. En cuanto al trabajo, he podido ver y aprender en primera instancia los procesos que se realizan desde una cadena de compañías y he podido tratar con un cliente cara a cara. Con Tecnativa, la estancia ha sido muy acogedora, me he sentido desde el primer minuto uno más del equipo de trabajo y sin duda estoy agradecido a cada uno de los integrantes del equipo.

La experiencia que más valoro, es sin duda el trato con un cliente desde dentro de la empresa, creo que es una experiencia perfecta para mi futuro y que sin ella el proyecto no hubiera sido el mismo.

Por lo que respecta a mi supervisor Sergio, me ha enseñado muchísimo y me ha hecho enfrentarme a problemas reales que han hecho que mis capacidades con Odoo aumenten. Y también tengo que agradecer a mi tutora María de los Ángeles por el apoyo que me ha dado con las entregas y las correcciones.

Bibliografía

[1] Tecnativa. Página web de tecnativa. https://www.tecnativa.com/. [Consulta: 08 de Junio de 2020].

[2] Odoo. Manual del desarrollador de odoo para la versión 13.0. https://www.odoo.com/documentation/13.0/. [Consulta: 08 de Junio de 2020].

[3] Odoo Community Association. Repositorio de la odoo community association en github. https://github.com/OCA. [Consulta: 08 de Junio de 2020].

[4] Odoo Community Association. Guía de contribución a la odoo community association. https://github.com/OCA/odoo-community.org/blob/master/website/Contribution/CONTRIBUTING.rst. [Consulta: 21 de Junio de 2020].

[5] Odoo. Notas de versión para odoo 13.0. https://www.odoo.com/es_ES/page/odoo-13-release-notes. [Consulta: 08 de Junio de 2020].

[6] Joaquín Medina Serrano. Metodología de desarrollo no te repitas. http://joaquin.medina.name/web2008/documentos/informatica/documentacion/logica/OOP/Principios/2012_07_30_OopNoTeRepitas.html. [Consulta: 08 de Junio de 2020].

[7] Agencia Tributaria. Legislación sobre las facturas simplificadas. https://www.agenciatributaria.es/AEAT.internet/Inicio/Ayuda/Manuales_Folletos_y_Videos/Manuales_practicos/_Ayuda_Folleto_Actividades_economicas/5__Impuesto_sobre_el_Valor_Anadido/5_9_Las_facturas/5_9_6__Facturas_simplificadas/5_9_6__Facturas_simplificadas.html. [Consulta: 08 de Junio de 2020].

[8] Cubic ERP. Video explicativo de la arquitectura que se sigue en odoo. https://www.youtube.com/watch?v=nTNeogNgkrw. [Consulta: 08 de Junio de 2020].

[9] Odoo Community Association. Guía de migración a la versión 13.0. https://github.com/OCA/maintainer-tools/wiki/Migration-to-version-13.0. [Consulta: 08 de Junio de 2020].