UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN

# COMERCIOS VIRTUALES SIN MARCA IMPULSADOS POR CORNERSHOP

MEMORIA PARA OPTAR AL TÍTULO DE
INGENIERO CIVIL EN COMPUTACIÓN

PABLO IGNACIO ALIAGA VALDERRAMA

PROFESOR GUÍA:
EDUARDO DÍAZ CORTÉS

MIEMBROS DE LA COMISIÓN:
NELSON BALOIAN TATARYAN
HUGO BELTRÁN ALEJOS

SANTIAGO DE CHILE
2022

# Resumen

Los productos sin marca, o *whitelabel*, han formado parte del mercado desde aproximadamente dos décadas y hoy, forman parte de nuestro día a día. Estos pueden verse como mochilas, calcetines, entre miles de otros formatos y son aquellos productos hechos por una empresa pero vendidos por un tercero. Se plantea al iniciar este documento, la existencia de un mercado nuevo en el mundo de la tecnología, relacionado con productos sin marca: *whitelabel e-commerces*, es decir, comercios virtuales sin marca. Estos forman parte de un nuevo paradigma y se entienden como aquellas experiencias de tienda virtual que son, en realidad, provistas por otra empresa. Desde fuera, estas registran elementos visuales de la marca que adquiere el producto y por ende, pareciera ser una experiencia virtual original de la tienda.

En el siguiente documento se verifica la posibilidad de adaptar un modelo de ventas online multi-tienda, como es el caso de *Cornershop*, para generar una experiencia de *whitelabel e-commerce*. La estructura seguida en esta memoria forma parte de un proceso de Ingeniería de *Software*, en específico, estará orientada en la definición de arquitectura de una solución que permitirá resolver la interrogante planteada: ¿Es posible adaptar un modelo de negocios de ventas online multi-tienda, como lo es el caso de *Cornershop*, a una experiencia whitelabel?

*Whitelabel* nace como una adaptación del sistema original de *Cornershop* que, inserto en un eco-sistema masivo de desarrollo TI, se convierte en un conjunto de componentes que permiten ofrecer a una tienda, la posibilidad de montar su propio comercio virtual bajo los estándares de calidad que han representado a una compañía posicionada a día de hoy, como una de las más importantes en el mercado de ventas online en América.

*A mi yo del pasado. Para que todos los esfuerzos rindan frutos.*

# Agradecimientos

Agradezco a mi familia, pues es gracias a ellos que existo; por haberme apoyado en todo lo que he necesitado y por haberme brindado claridad en momentos cruciales. Mis padres, como pilares fundamentales de mi persona, mis hermanos, como compañeros de vida, a quienes no solo considero familia, sino que también amigos.

Agradezco a mis amigos, por ser mi soporte, por haber sido catalizadores de todo el conocimiento aprendido en mi vida. Recordaré por siempre las tardes enteras que pasamos estudiando entre pizarras, entre todos nos permitimos crecer y por eso les doy gracias.

Agradezco a mis profesores, pues es de ellos que he podido presenciar el amor al conocimiento, el generar un ambiente de crecimiento inter-personal que es la universidad y por haberme enseñado, en este caso, cómo ser un ingeniero integral.

Agradezco mi trabajo, pues me ha permitido crecer como desarrollador, como persona y no dejar de ser el soñador que siempre he sido. Es gracias a *Cornershop* que he tenido la posibilidad de desarrollar esta memoria. A mis compañeros de trabajo, no solo he aprendido con ellos, sino que, con el tiempo, se han convertido en amigos y han creado una atmósfera de trabajo ideal, espero poder seguir conociéndoles en un futuro.

Agradezco a la vida, por haberme presentado ocasiones únicas, por haberme dejado vivir y guardar en mí, recuerdos inolvidables; por haberme dejado conocer a personas increíbles que me han enseñado ser quien soy. Le agradezco a mi yo del pasado, pues supo buscar fuerzas para conseguir todas las metas propuestas; pues tuvo la valentía de tomar oportunidades y de vivir sin arrepentimientos; sin mi pasado, no sería la persona que conozco y agradezco entonces todas las alegrías y tristezas que he vivido.

Finalmente, agradezco a quien lee este apartado por dejarme entregar el conocimiento generado en este documento a su repertorio de memorias. Espero la lectura del mismo sea placentera y al finalizar la lectura, pueda entender todo el proceso vivido este proyecto y que fue parte de mí durante esos seis meses.

# Tabla de Contenido

# Capítulo 1

## 1.1. Introducción

En los últimos años, *Cornershop*, empresa dedicada al desarrollo tecnológico y en específico al comercio online, se ha posicionado como el emprendimiento más exitoso de Chile[8]. Si bien se puede especular que el repentino crecimiento de la aplicación fue consecuencia de la situación pandémica vivida en los últimos años, es indudable que al día de hoy la compañía ha alcanzado una posición estable en el marco económico nacional; más aún luego de ser adquirida por *Uber*, una de las empresas más importantes en el ámbito tecnológico a nivel mundial.

Si bien, a primera vista, en el mundo de las tiendas online, lo más relevante pareciera ser el usuario final, es decir, el consumidor, es lógico pensar que dentro de cualquier proceso de compra deben participar esencialmente un proveedor y un consumidor. Es así como dentro de los usuarios principales de *Cornershop* se encuentran los consumidores (usuarios finales) y las tiendas, las cuales tienen la posibilidad de vender sus productos a través de una plataforma que facilita al consumidor la realización de una compra. Es respecto a este último punto que existe un grupo de tiendas, mayoritariamente tiendas pequeñas, que no poseen un comercio virtual propio y por temas de *branding*[1], desean llevar su marca al mundo digital en un comercio virtual propio.

El problema planteado en este documento está directamente relacionado con estos comercios nuevos o pequeños, que buscan impulsar el branding de su compañía con un comercio virtual. *Cornershop*, en la actualidad, trabaja con tiendas que no poseen un mercado virtual y de alguna forma ayuda a digitalizar su catálogo virtual en la plataforma (como es el caso de *AquaChile*, empresa dedicada a la distribución de productos marinos); sin embargo, esto no satisface la necesidad de poseer una experiencia virtual propia por parte de ciertas tiendas.

La necesidad de tener una tienda virtual tiene que ver con algo descrito por el académico Hugo Delgado[9], tener un espacio virtual de tu marca en la web, da la posibilidad de crecer exponencialmente, puesto que genera: una segunda puerta de entrada del público a la tienda, incluso a nivel internacional (dada las cualidades naturales de la web); publicidad constante, dado por el mismo hecho de tener una página o por la posibilidad de ser indexado en algún

[1] *https://www.elisava.net/es/noticias/que-es-branding*

motor de búsqueda web, por ejemplo el de *Google*[2]; o simplemente por temas de *branding*: incrementar la posibilidad de posicionar la marca en la mente del consumidor. Es por estas razones que la mayoría de las tiendas busca la posibilidad de tener una página web y más aún es tentadora la idea de tener un comercio virtual donde los consumidores puedan realizar sus órdenes.

Es así como, en el grueso de este documento, se presentará una solución que logrará solventar el problema descrito anteriormente. Esta solución, naciendo desde la misma empresa, se basa en tecnología que, por un lado, cumple con estándares de software a nivel internacional y, por otro, ha sido desarrollada específicamente para posicionarse como una aplicación consolidada en el mundo de las tiendas online.

La solución planteada - para términos de este documento - será denominada *Whitelabel E-commerce*, o sólo *Whitelabel*. El concepto *whitelabel* ha estado presente a nivel global durante un par de décadas y tiene relación con lo siguiente: un producto *whitelabel*, como es descrito en un artículo de la revista Forbes[3], es un producto hecho por una empresa pero vendido por un tercero. Un *e-commerce whitelabel* busca ser un comercio virtual provisto por una empresa pero presentado, desde el punto de vista del usuario final, como una experiencia propia de otra tienda. Este proyecto busca proveer la experiencia de *Cornershop* presentada como si fuera de una tienda en específico.

De este modo, *Whitelabel* busca solventar la necesidad de las tiendas de poseer un comercio virtual propio; y a consecuencia de esto, dar a las tiendas la capacidad de escalar su comercio en cuanto a cifras de ventas e incluso en el posicionamiento de su marca.

[2] *https://google.cl*

## 1.2. Marco teórico

En esta sección se describe el marco tecnológico, en que se encuentra la solución. Para comenzar con el documento, se plantea la siguiente pregunta a modo de guía:

**¿Es posible adaptar un modelo de negocios de ventas online multi-tienda, como lo es el caso de Cornershop, a una experiencia whitelabel?**

Para responder esta pregunta se revisará de manera amplia algunas soluciones similares que han sido implementadas en el mercado y también se presentará una breve descripción del sistema actual de *Cornershop*. Sin embargo, dado que esta memoria se centrará en procesos de Ingeniería de Software y en específico de arquitectura, es fundamental definir qué proceso de diseño de arquitectura se va a utilizar en este proyecto.

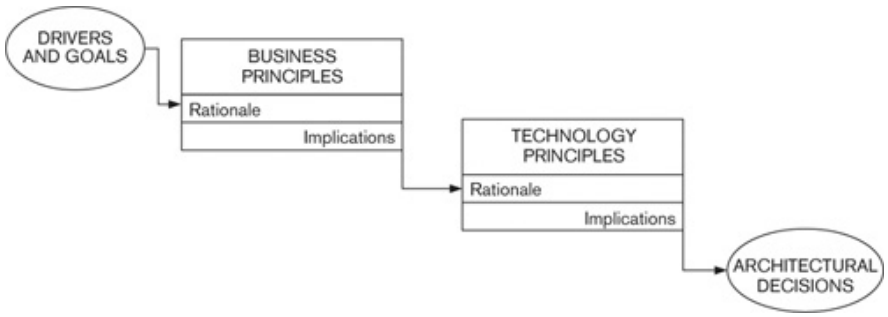


Figura 1.1: Diagrama de proceso de diseño de arquitectura
Fuente: *Software Systems Architecture*[1]

Este proceso estará basado en el definido por el libro *Software Systems Architecture*[1] por *Rozanski* y *Woods*. El cual nos plantea un acercamiento lógico para el diseño de una arquitectura de *software* enfocado en los requerimientos iniciales de los *stakeholders* de un proyecto. A niveles generales y, tal como se muestra en en la figura 1.1, el proceso consta de los siguientes pasos:

1. Se definen los *drivers and goals*, los cuales representan los requerimientos e inquietudes iniciales de los *stakeholders*. Lo más importante respecto a estos es que, en su totalidad, representen lo que los *stakeholders* esperan del proyecto.
2. Se definen los principios de arquitectura, los cuales pueden ser de negocios o tecnológicos. Estos representan las bases para la definición de la arquitectura y pueden ser las creencias o enfoques que el arquitecto define para el proyecto. Estas deben estar fundamentadas por las inquietudes iniciales del proyecto.
3. Finalmente se especifican las decisiones de arquitectura, estas se verán reflejadas al consolidar la arquitectura del proyecto. De manera análoga al paso anterior, estas decisiones deben estar fundamentadas por principios de arquitectura, o en casos excepcionales, por un subconjunto de inquietudes.

Tal como se aprecia en la secuencia, el proceso descrito por *Rozanki y Woods* asegura un razonamiento lógico para la definición de cada una de estas piezas y, por ende, otorga la ventaja de tener trazabilidad respecto a las decisiones tomadas en el transcurso del proyecto.

Además, es importante destacar respecto al punto 1 que el proceso mediante el cual se identifican las inquietudes y requerimientos iniciales del proyecto, fue una actividad realizada, en conjunto, por el arquitecto del proyecto y los respectivos *stakeholders*, siendo estos últimos responsables de validar que estos reflejaran sus puntos de vista. Estos resultados pueden ser encontrados en la tabla 1 del anexo.

Desde este punto del documento, en adelante, se hará referencia a estas piezas definidas utilizando la siguiente notación: [**referencia de ejemplo**]. Estas representarán, inquietudes [**C**], principios [**P**] o decisiones de arquitectura [**A**] definidos para el proyecto.

A continuación, se presenta en la tabla 1.1 un aglomerado de los principios fundamentales que se tuvieron a la hora de plantear el proyecto. De esta forma podremos utilizar estas definiciones en el restante del proyecto.

| Principios de arquitectura | |
|---|---|
| P1 | El producto en cuestión debe ofrecer la capacidad al cliente de montar un mercado virtual propio. Entre las cualidades principales se encuentran: presencia de un catálogo online, compra de productos, creación de órdenes; entrega de órdenes y servicio al cliente. [C1, C3-C6] |
| P2 | Las opciones de personalización del producto a ofrecer serán básicas. Se espera que el cliente en cuestión sea capaz de colocar su marca (colores y logos) en el frontis de la aplicación. [C2, C8] |
| P3 | Para garantizar el apoyo al branding de las tiendas, este producto debe ser accedido utilizando un dominio web provisto por el cliente. No es necesario que el proceso de adquisición y configuración del dominio sea supervisado por Cornershop. [C2, C7] |
| P4 | Mantener estándares de seguridad de la industria sin que esto afecte las funcionalidades básicas del sistema a desarrollar, es decir, las relacionadas con generar la experiencia de un mercado virtual propio: compras y deliveries. [C1, C2, C13] |
| P5 | El desarrollo de esta solución debe seguir los estándares internos vigentes en Cornershop; cuyos objetivos apuntan a garantizar escalabilidad y mantenibilidad del software generado en la compañía. [C12-16] |
| P6 | Además, la solución debe seguir ciertos estándares externos de desarrollo de software / web que garanticen un estándar de producto a nivel internacional; tales como PEP8, W3C [C10, C18, C19]. |
| P7 | La solución debe ser capaz de recolectar información básica sobre las órdenes realizadas utilizando esta solución. [C9, C11] |

Tabla 1.1: Principios de arquitectura del proyecto
Fuente: Elaboración propia

### 1.2.1. Soluciones similares

Actualmente, existen solo dos productos que se han posicionado como principales en el negocio de las experiencias single-store o whitelabel en Chile:

1. *MercadoShops*: producto relativamente nuevo de *MercadoLibre*, empresa y aplicación con una trayectoria de más de una década dedicada al comercio virtual en Latinoaméri-

ca, que busca ofrecer una experiencia *whitelabel*; con la restricción de que esta tienda debe estar previamente en su catálogo. En la figura [1] es posible apreciar la tienda virtual de *Sennheiser* en Chile, sitio que usa *MercadoShops* y que a raíz de esto comparte en su mayoría elementos visuales con el sitio base de MercadoLibre; aun así, se siente como una tienda propia de la marca. En esta experiencia no se usa la sesión de usuario y la compra efectiva - el carrito[3] se maneja en el mismo sitio - se realiza una redirección hacia el sitio base MercadoLibre manteniendo el carrito armado en la página *whitelabel*.

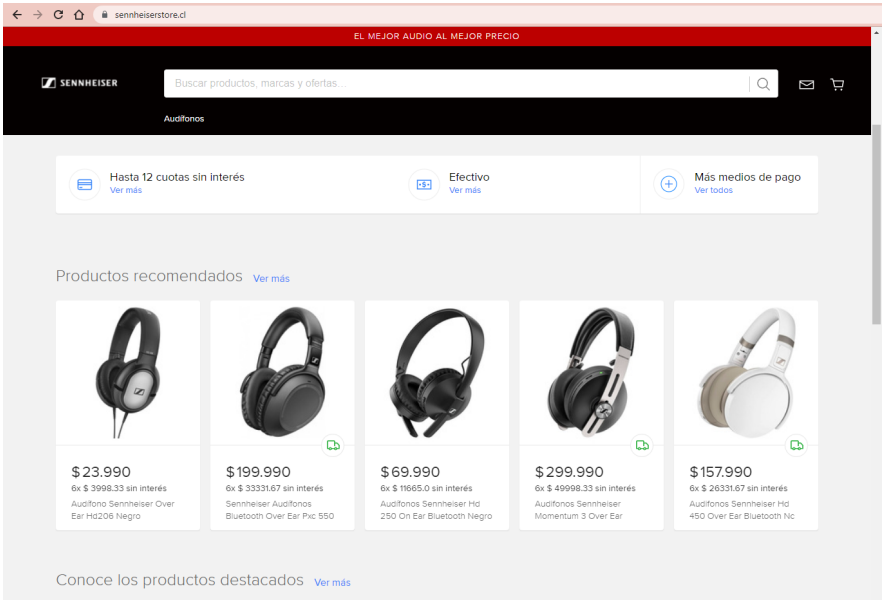


Figura 1.2: Captura de pantalla de la tienda *Sennheiser* Chile
Fuente: Elaboración propia

2. *GetJusto*: emprendimiento chileno que es un producto whitelabel enfocado en restaurantes, que otorga, además de un comercio virtual propio, los servicios de *delivery*[4] para este tipo de comercios. Al igual que en el caso anterior, entre sitios basados en *GetJusto* se pueden apreciar similitudes claras, ya que nacen de un producto central; aun así, cada una de estas páginas tiene su personalidad y estilo propio. A modo de ejemplo, basta mencionar a *RockTheFood* y *SushiOK*, ambos restaurantes ofrecen su sitio web utilizando el producto *GetJusto*.

Si bien ambas soluciones son similares en cuanto a el término *whitelabel*, cada empresa implementa este tipo de productos de manera distinta. En términos de negocio, *MercadoLibre*, trabaja sobre la venta online de productos más no insumos y la entrega del producto la debe realizar el proveedor hasta un punto central donde luego *MercadoLibre* realiza la entrega al usuario; por otro lado, *GetJusto* trabaja sobre restaurantes y se hace cargo de la entrega asumiendo que el pedido ya está envuelto y listo para su entrega en el local. Cabe destacar que, en el apartado 2 presente en el anexo, se pueden encontrar referencias de todos los sitios o aplicaciones mencionados en este documento.

[3]Componente encargado de almacenar los productos que el usuario desea comprar.
[4]Término anglosajón para referirse al proceso de entrega de un pedido.

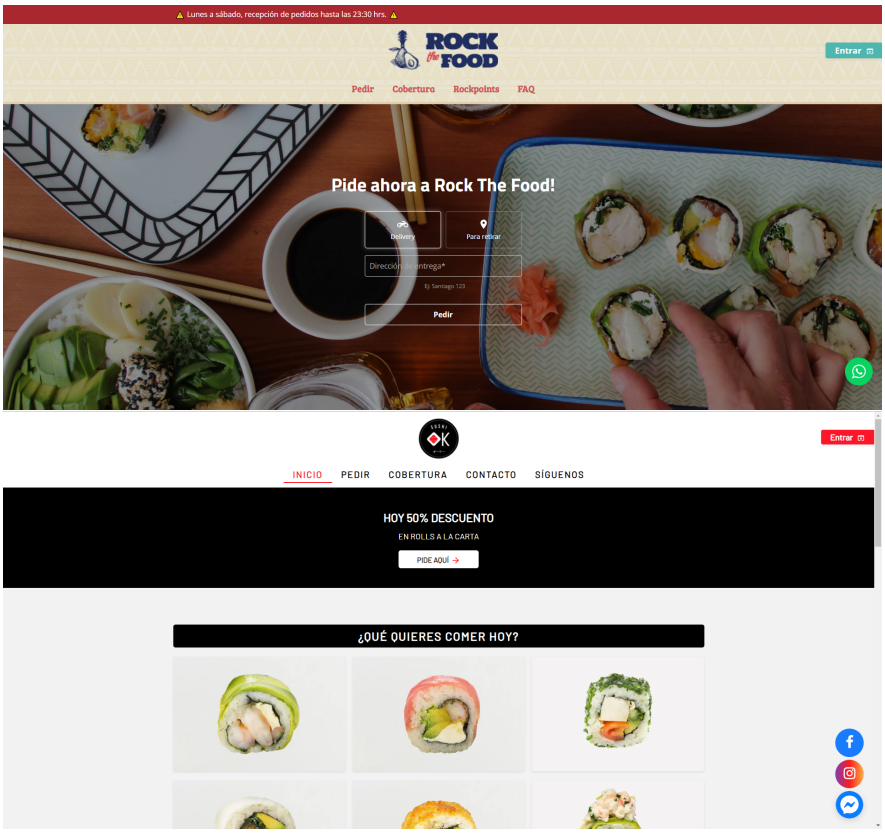


Figura 1.3: Capturas de pantalla de *RockTheFood* y *SushiOk*
Fuente: Elaboración propia

Teniendo en cuenta estas diferencias, vale la pena notar que ninguna de ellas trabaja sobre el área de las *groceries* o insumos, mercado en el cual *Cornershop* ha sabido liderar y por ende, tiene la capacidad de diferenciarse de estos otros productos. Por otro lado, el proceso de entrega de Cornershop es propio de la empresa y en esta el *driver*[5] se encarga de recolectar el pedido y hacer la entrega al usuario, esta es otra cualidad que le permite adelantarse a la competencia, puesto que un proceso de entrega más flexible le permite trabajar sobre una mayor variedad de tiendas.

### 1.2.2. Vista general del sistema de Cornershop

De los párrafos anteriores se desprende la idea de que es posible extender un modelo de negocios basados en el servicio online de múltiples productos, tiendas o restaurantes para su funcionamiento en una tienda en específico. Es por esto que, como parte del marco teórico, se definirá el estado actual de la aplicación principal de *Cornershop*, es decir, *Cornershop Web*. Esta es un comercio virtual multi-tienda, enfocado inicialmente en la venta de insumos, pero que, con el paso del tiempo, se ha ido expandiendo a la venta de gran variedad de productos. Este modelo de negocios ofrece tanto la compra del usuario, la recolección del pedido por parte de un *picker*[6], como también la entrega de la orden por parte de un *driver*; en este sentido, se podría decir que *Cornershop Web* ofrece una experiencia de comercio virtual completa.

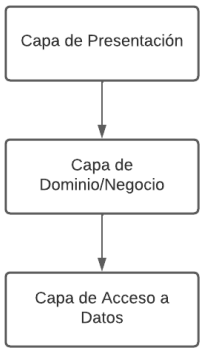


Figura 1.4: Ejemplo de arquitectura de N-capas
Fuente: Elaboración propia

Respecto a la arquitectura general de esta aplicación, es importante mencionar que esta sigue una arquitectura de aplicaciones de "N capas", tal como se describe en un artículo de la organización Red Hat[4], es decir, se divide en una capa de servicios de *backend* y otra de vistas *frontend*; esta última es la encargada de presentar la página web al usuario. A continuación se mencionarán un conjunto de lenguajes de programación y *frameworks*[7] que son utilizados en el sistema de *Cornershop*, referencias a sus sitios web pueden ser encontrados en el apartado 2 del anexo. En la figura 1.4 es posible apreciar un ejemplo de este tipo de arquitectura.

[5]Persona encargada de entregar la orden, desde la tienda al usuario final.
[6]Persona encargada de armar el pedido en la tienda.
[7]Herramienta de código que puede ser utilizada como base el desarrollo de otro.

Por un lado, el *frontend* de la aplicación está desarrollado principalmente utilizando la librería *React* del lenguaje de programación *JavaScript*; aun así, debido al legado de haber utilizado *Django* como *framework* único en el inicio del proyecto, existen ciertas vistas frontend que aún están almacenadas en el *backend*, por ejemplo: registro de usuario, ingreso de usuario y recuperar contraseña. Dentro del flujo normal, el *frontend* accede a los servicios del *backend* a través de API - según *Amazon* [11], mecanismo que permite a dos componentes comunicarse entre sí - que se emplean no solo en *Cornershop Web*, sino también en las aplicaciones móviles de la misma compañía.

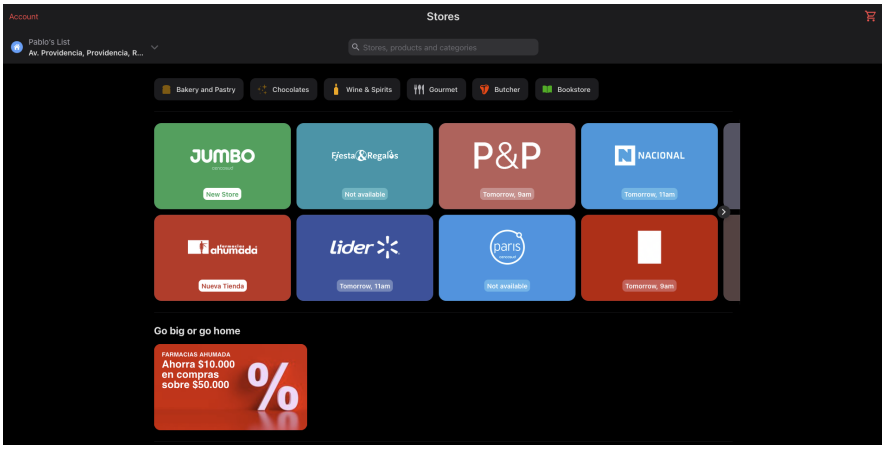


Figura 1.5: Captura de pantalla de *Cornershop Web*
Fuente: Elaboración propia

Por otro lado, la capa de *backend* - desarrollada sobre el *framework Django* - sigue una mezcla de dos patrones de arquitectura, descritos en el artículo antes mencionado [4]: de manera superficial es una aplicación monolítica, es decir, todos los servicios están alojados en una misma aplicación; sin embargo, de manera interna - utilizando un componente que emula servicios en *Django* llamado *apps* - se organiza en micro-servicios y estos, a su vez, se organizan según el modelo de "N-Capas", las cuales se dividen en capas de modelos, proveedores, servicios y vistas. La capa de **modelos** está encargada de generar tablas en la base de datos, traduciendo objetos/clases de la aplicación a tablas almacenadas, a esto se le conoce como *Django ORM*; la capa de **proveedores**, encargada de realizar consultas a la capa de modelos (o consultas directas a la base de datos); la capa de **servicios** encargada de servir a otros micro-servicios con la información o lógica de negocios que estos necesiten; y finalmente la capa de las **vistas**, las cuales sirven como puerto para comunicar al *backend* con el exterior, más específicamente con el *frontend*.

Para motivos de este proyecto se describirán ciertas características específicas del software de *Cornershop*, estas características fueron elegidas porque guardan relación con las inquietudes iniciales del proyecto, definidas en el apartado 1 y que serán de bastante utilidad en las secciones posteriores de este documento:

- El manejo de sesiones de usuarios de la aplicación web se realiza a través de *cookies*, las cuales, según los desarrolladores de Mozilla[18], son una pieza de información manejada en la web y anclada al dominio de un sitio web. De manera opuesta, para las aplicaciones

móviles de *Cornershop*, el manejo de sesiones de usuario se efectúa con base en *Oauth2*[8] utilizando una librería externa de *Python Django Oauth Toolkit*. [C7]

- Dado que se plantea usar el producto desde dominios diferentes[C7], se tienen ciertas restricciones respecto a estas peticiones web entre dominios, es importante mencionar, entonces, el mecanismo CORS[9] que, según Mozilla[17], es un encabezado HTTP que permite a los servidores definir los orígenes desde los cuales se espera recibir peticiones; es así como, siguiendo el mecanismo de seguridad mencionado, un navegador web bloquea las peticiones HTTP hacia servidores que no han registrado el dominio (desde donde se hace la petición) en su *whitelist* o lista de orígenes admitidos de CORS. Para el uso de este componente de seguridad, el *backend* hace uso de un *middleware*[10], llamado *Django-cors-headers*. [C7][C19]

- Como último punto relevante a mencionar sobre el sistema de *Cornershop* se tiene el proceso de creación de órdenes. Siendo esta una aplicación destinada al comercio online, es lógico que las órdenes son la pieza fundamental en el proceso de compra. Luego de que el usuario solicite una orden en la aplicación web, se inicializa un modelo intermedio llamado *Checkout*; este modelo tiene el potencial de convertirse en una orden y para ello existe un componente asíncrono[11] encargado de convertir este modelo en una orden efectiva, verificando que todo el sistema esté preparado para servir esa orden; cuando esto ocurre, se le notifica al usuario que su orden ha sido creada exitosamente [C1]. Un diagrama de este proceso puede ser encontrado en la figura 1.6.

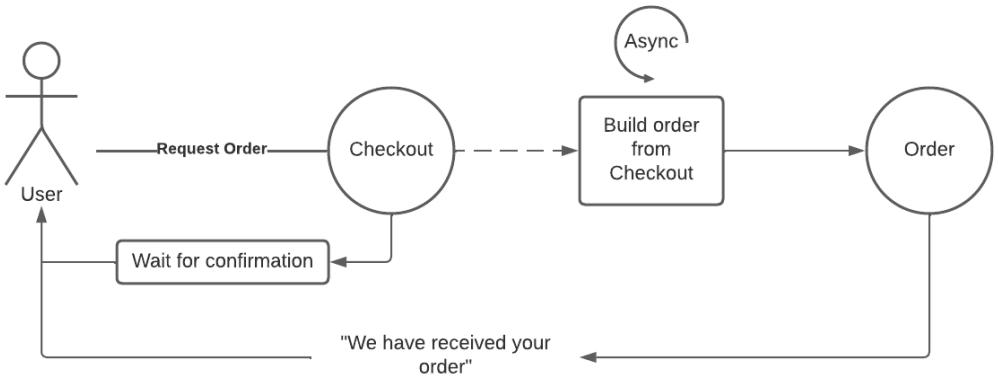


Figura 1.6: Diagrama de proceso de creación de órdenes
Fuente: Elaboración propia

[8] https://oauth.net/2/
[9] *Cross origin resource sharing*
[10] Capa de software intermedia entre las capas de aplicación e inferiores (sistema operativo y red).
[11] Código que puede ejecutarse en paralelo con otros

## 1.3. Objetivos

### 1.3.1. Objetivo general

Ser capaces de ofrecer una solución a aquellas tiendas sin presencia en el mundo digital, que deseen impulsar su *branding* al tener un comercio virtual propio. Esta solución, dado que *Cornershop* ya tiene experiencia en esta área, debe estar nivelada a los estándares de la compañía.

### 1.3.2. Objetivos específicos

Los siguientes objetivos, si bien son limitados, estarán plenamente relacionados con las características e inquietudes del proyecto, definidas en el anexo 1. Es a partir de estas que se definirán, de igual manera, la evaluación de este trabajo.

1. Generar la experiencia completa de un comercio virtual. Este objetivo deriva directamente de los puntos [C3][C4] del apartado de 1. Las características requeridas de una tienda en línea son:
   (a) Ofrecer un catálogo online.
   (b) Realizar compras (en el momento o programadas).
   (c) Ofrecer un servicio de *delivery* para la misma orden.
2. Generar opciones de personalización básica para la aplicación. De esta forma impulsar la idea del *branding* de la tienda. Esto está relacionado con los puntos [C2][C6] de 1. La personalización básica de la aplicación incluiría logotipo y colores, dos fundamentos para establecer la idea de *branding*.
3. Ser capaces de alojar la aplicación en un dominio web provisto por la tienda. Esto está relacionado con los puntos [C2][C5] y sigue la línea de los objetivos relacionados con el *branding*.
4. Ser capaces de recolectar información respecto al uso de esta solución. Específicamente, de las órdenes generadas en la plataforma.
5. Generar producto mínimo viable de esta solución en alguna tienda. De esta manera poder probar el funcionamiento de la solución en un escenario real y se puede verificar el interés en los clientes.

### 1.3.3. Evaluación

Teniendo en cuenta que el proyecto es fundamentalmente sobre desarrollo de *software*, y que este se desarrolla a partir de la guía descrita por *Nick Rozanski* y *Eoin Woods* [1], la evaluación general estará enfocada principalmente en los objetivos descritos, pero siempre respaldada por las cualidades esperadas del producto (anexo 1).

Es así que, para estimar el éxito o fracaso del proyecto *whitelabel* se presentan dos evaluaciones, una enfocada en los requerimientos del proyecto y la otra, enfocada en pruebas de calidad, en entornos reales. Estas son:

1. Se realizará una evaluación de características de la solución siguiendo las cualidades esperadas definidas en el anexo de inquietudes 1. Respecto a este punto, vale la pena destacar que el cumplimiento de estas características implica el éxito en los puntos 1, 2, 3 y 4 de los objetivos específicos, puesto que estos están basados plenamente en estas inquietudes.
2. Se evaluará el funcionamiento de la aplicación en un caso real, con alguna tienda que pueda estar interesada en el producto. Esto se utilizará como prueba de concepto para evaluar la salida del producto al mercado.

# Capítulo 2

# Desarrollo

## 2.1. Diseño de la solución

Luego de comprender las causas por las cuales surgió el proyecto, fue necesario definir cuál sería la solución que se abordaría en los pasajes de este documento. Esta solución - *Whitelabel E-commerces* - guarda relación con la implementación de una tienda virtual a un comercio que desee integrarse al mundo digital.

*Cornershop Whitelabel* se basa en la idea de ofrecer una experiencia de comercio virtual personalizada para una tienda, de manera tal, que esta pueda ser identificada como una tienda online propia por el comercio que desee utilizarla. Para ello - *whitelabel* - cuenta con tres características fundamentales que definirían el valor del producto a desarrollar y otras cuatro relacionadas con estrategias de negocio y tecnología de *Cornershop*; todas estas características se encuentran documentadas en la tabla de principios 1.1.

Respecto a las cuatro cualidades cruciales de la solución, se tiene que: esta debe implementar una experiencia básica de un comercio virtual, basada en la misma idea de presente en la aplicación principal de la compañía, esto es, una aplicación que permite por lo menos mostrar un catálogo de productos, realizar compras, crear órdenes, llevar las órdenes al consumidor y tener herramientas para ayudar al consumidor en caso de ser necesario (servicio al consumidor) [P1]; para garantizar que el cliente pueda impulsar el *branding* de su compañía la solución deberá ser personalizable, al menos mostrando el logotipo y colores representativos de la marca [P2]; por último, siguiendo la línea del *branding* y teniendo en cuenta que el usuario final debe "sentir" que se encuentra en un espacio virtual o en un dominio de la empresa, *whitelabel* debe ser accedido usando una dirección web propia de la tienda [P3].

En cuanto a estrategias de negocio de la compañía, se tiene que esta se considera una empresa orientada al mundo de la tecnología y el internet, es por esto que, uno de sus focos fundamentales es la obtención y manejo de datos generados en sus productos. En efecto, otro de los requerimientos del proyecto es que la solución debe ser capaz de generar información a partir de las órdenes generadas bajo esta plataforma [P7].

Las dos siguientes cualidades de la solución guardan relación con la estrategia tecnológica

| Decisiones de arquitectura | |
|---|---|
| A1 | La solución aprovechará en la medida de lo posible el desarrollo que ya se tenga en la aplicación principal, ya que esta provee las cualidades necesarias para generar una experiencia de tienda online. [P1][P4][P5][P6]+[C7.] |
| A2 | La solución será almacenada en un servidor central, el cual podrá ser accedido desde otros dominios a través de registros DNS. [P3] |
| A3 | La personalización de whitelabel será representada en un modelo de datos nuevo en el ecosistema de Cornershop y esta, a su vez, por tienda, almacenará información de su logotipo y dos colores, uno principal y otro secundario. [P2] |
| A4 | Respecto a la entrega del pedido y el servicio al consumidor, estos seguirán siendo administrados por la marca Cornershop. De esta manera, la tienda virtual pertenece a la tienda en cuestión, pero la experiencia en su interior será impulsada por Cornershop. [P1] |
| A5 | Dentro de la información almacenada respecto a las órdenes se podrá identificar si la orden es whitelabel y a que tienda corresponde. [A1][P7] |

Tabla 2.1: Decisiones de arquitectura del proyecto
Fuente: Elaboración propia

presente en la empresa y es que, todo desarrollo generado en *Cornershop*, debe estar a la cabeza del estándar internacional [C10]. Por un lado, la solución debe regirse por los estándares internos a la compañía, definidos en el documento "*Well-sized services*", estos buscan garantizar la escalabilidad y mantenibilidad de todo *software* desarrollado [P5]. Por otro lado, la solución debe cumplir con los estándares globales de las tecnologías que se utilicen y, dado que el *software* debe estar escrito en *Python*[C16] y, que la solución debe ser accedida desde internet, los estándares externos que debe cumplir son PEP8 y W3C, respectivamente [P6]. Estas dos cualidades, en conjunto, permiten asegurar una calidad de producto comparable con la misma de *Cornershop*, el cual, como ha sido mencionado previamente, es uno de los emprendimientos mejor catalogados a nivel latinoamericano.

Si bien, ya se describieron las características fundamentales y estratégicas del producto, existen ciertas restricciones que deben ser tomadas en cuenta en cualquier proyecto de ingeniería, estas son las relacionadas con aspectos de la vida real (en el libro [1] *real-life constraints*), dentro de las cuales se considera: el tiempo, el presupuesto, las capacidades técnicas del equipo, entre otras. Los detalles de estas restricciones se encuentran en el apartado [C20-23] del anexo 1, aun así, dentro de este proyecto, lo más relevante tiene que ver con que tener un presupuesto reducido y fechas límites acotadas: equipo de diez personas y un tiempo límite de seis meses para probar este producto en público.

### 2.1.1. Definición de la arquitectura

A continuación se describe el proceso que se llevó a cabo para diseñar la solución. Teniendo en cuenta la naturaleza del proyecto, el desarrollo de esta será principalmente relacionada con la arquitectura del *software* a realizar. De manera similar a lo planteado en la tabla 1.1, se especificarán ciertas decisiones de arquitectura - en la tabla 2.1 - que guiarán el diseño de esta.

Para hablar del diseño de la solución de *Whitelabel* es necesario comprender que, dado

| WhitelabelConfig | |
|---|---|
| UUID | Identificador único del modelo. |
| store_id | Identificador numérico de una tienda. |
| store_uuid | Código UUID para identificar a la tienda. |
| redirect_store_url | Dirección web de origen de la tienda. |
| light_primary_color | Color primario del whitelabel en hexadecimal. |
| light_secondary_color | Color secundario del whitelabel en hexadecimal. |

Tabla 2.2: Campos del modelo Whitelabel Config
Fuente: Elaboración propia

que el *software* debe reusar componentes de la empresa ya creados [A1], este no es un desarrollo de una aplicación desde cero, sino que será construido como extensión de un sistema complejo y masivo. Es por esto que, para entender la arquitectura del sistema de la solución, se definirán dos componentes principales: el primero tiene que ver con el producto principal o el sistema al cual accede el consumidor (visto de manera más simple, la página web), esto es, la **arquitectura lógica** de la aplicación; el segundo, en contraste, tiene que ver con cómo conectar la aplicación a la web, esto es la **infraestructura** de la aplicación.

### 2.1.2. Arquitectura lógica de la aplicación

La arquitectura lógica de la solución corresponde a todos los componentes que definen el sistema de manera interna, sin tener en cuenta las conexiones con el exterior. Este gran componente es el que otorga el valor principal al producto y para su definición se usará el modelo de arquitectura de 3 niveles explicado por *IBM*[15], es decir: capa de datos, capa de aplicación y capa de vistas.

#### Modelo de datos

Respecto a la capa de datos, la extensión que se realizó desde el modelo de datos de *Cornershop* se divide en dos cambios principales: uno relacionado con la configuración de la experiencia *Whitelabel* y otro relacionado con poder marcar las órdenes según desde qué plataforma se han realizado.

Por un lado, se añade una tabla - llamada *Whitelabel Config* - que permite almacenar la información necesaria para montar esta experiencia [A1], esto es, la personalización por tienda [A3] y de alguna forma identificar a cuál corresponde, según el origen web del usuario [A2], es decir, la dirección web provista por la tienda. En la figura 2.2 se especifica el detalle de cada uno de estos campos.

Por otro lado, para conseguir identificar las órdenes que se efectúan en esta experiencia [A5], se añade un campo extra en los modelos encargados de crear órdenes en *Cornershop*, (como fue definido en la sección Marco teórico 1.2, el modelo *Checkout* y el modelo *Order*), este nuevo campo recibe el nombre de *marketplace tenant* y representa desde qué plataforma o *marketplace* se genera la orden. Para propósitos de este proyecto, este campo puede tener dos

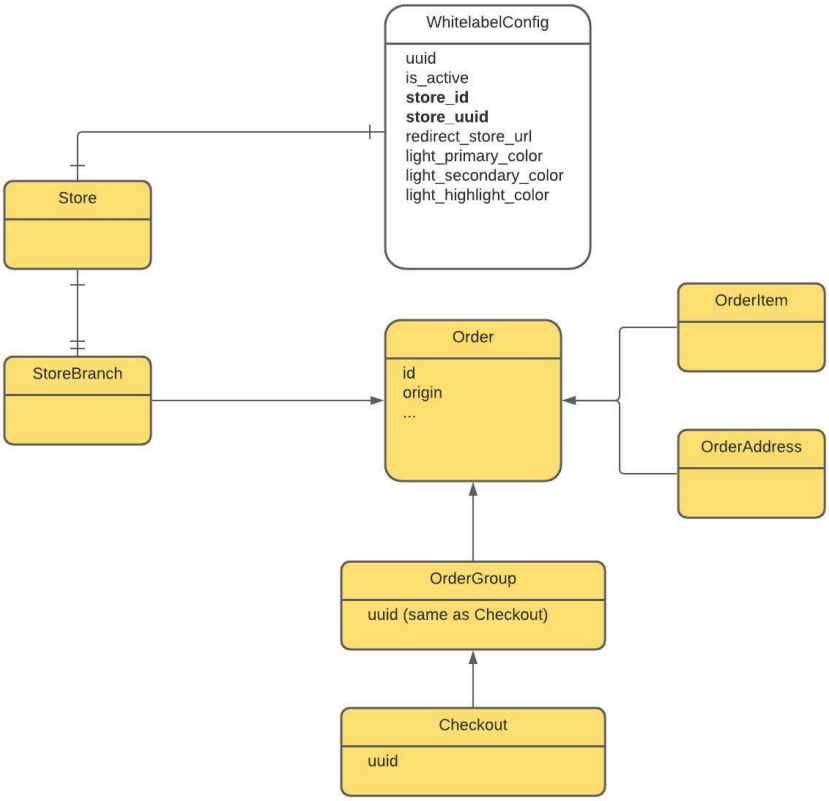


Figura 2.1: Modelo de datos de *Whitelabel*
Fuente: Elaboración propia

posibles grupos de valores: "*customer_app*" o un identificador de la aplicación *Whitelabel*. De esta forma se puede añadir esta información a las estadísticas (y datos en general) generadas en *Cornershop*.

En la figura 2.1 es posible identificar los cambios mencionados en cuanto a datos, cabe destacar que las tablas amarillas corresponden a datos que existían previamente en el sistema y que aportan información valiosa a la solución, como es el caso de la relación entre una configuración *Whitelabel* y la tienda correspondiente (relación 0 a 1).

### Capa de aplicación

La capa de aplicación, que para este proyecto representa la extensión al *backend* de *Cornershop*, corresponde a la lógica de negocios o servicios desarrollados para montar la aplicación *Whitelabel*. Es importante destacar que los detalles presentes en esta sección corresponden a una extensión del sistema de *Cornershop* y por ende no corresponde a la totalidad de los servicios de esta capa.

A nivel lógico, teniendo en cuenta la arquitectura previa de la aplicación de *Cornershop*, tal como se puede apreciar en el diagrama de la figura 2.2, la capa de aplicación de *Whitelabel* corresponde, por una parte, al *backend* de la aplicación principal (conjunto previo de API) y,

por otra, a una nueva aplicación o, utilizando el léxico de *Django*, una nueva *app* creada en esta capa: *Whitelabel App*. Si bien, en realidad corresponde a una API y servicios de lógica por separado (esto, pues forma parte de los estándares de código de la compañía [P5]), por simplicidad, en los detalles de este documento se considerará como una *app* única.

Para definir entonces las características funcionales de la capa de aplicación se considerarán dos grupos fundamentales: la funcionalidad de *Whitelabel App* y los cambios realizados en el *backend* previo de Cornershop.

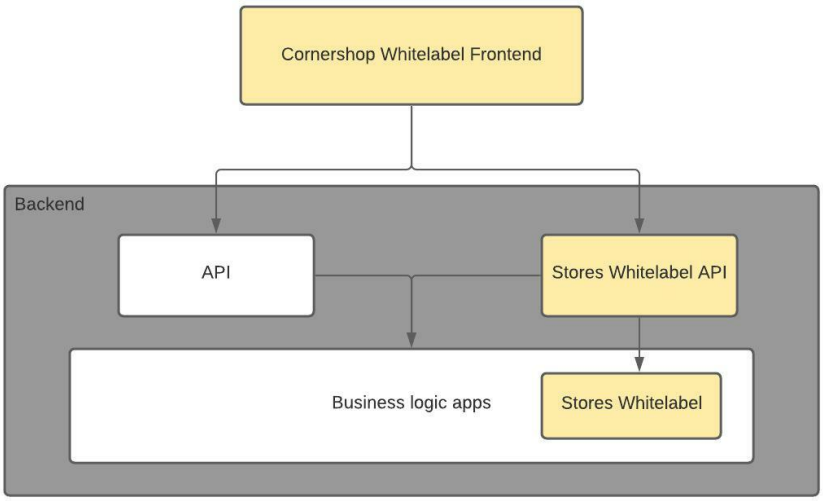


Figura 2.2: Diagrama de la arquitectura lógica de *Whitelabel*
Fuente: Elaboración propia

Esta nueva micro-aplicación es la encargada de toda la lógica necesaria, en cuanto a aplicación, para los *Whitelabel E-commerces*. La funcionalidad más valiosa de la *Whitelabel App* es que permite mostrar toda la configuración necesaria para montar la experiencia personalizable por tienda [A2], este servicio es el llamado "*whitelabel bootstrap*" y accede, a partir del origen de la petición web (esta información puede ser obtenida en el encabezado HTTP *Origin* [19]), a la información guardada en el modelo de datos descrito anteriormente; para asegurar tiempos de respuesta [P5] según el estándar de la compañía, este servicio utiliza un *caché*, es decir, una capa de almacenamiento de datos de alta velocidad [10]. Un diagrama UML respecto al servicio mencionado pueden ser encontrados en la figura 2.3, además en la figura 1 se encuentra documentación sobre la firma del mismo.

Otros dos servicios fundamentales de esta aplicación corresponden a los relacionados con consultas rápidas sobre el nuevo modelo generado: por un lado, un servicio que consta de un *caché* de todas las direcciones web que se tiene para la experiencia *Whitelabel*, esto será utilizado para la infraestructura de la aplicación, en secciones siguientes del documento, y es una pieza fundamental para cumplir con los estándares inter-dominio al navegar por la web, más específicamente para solventar el problema asociado a CORS, mecanismo definido por el W3C [C6]; por otro lado, un servicio que permite identificar a qué tienda corresponde la creación de una orden utilizando el origen de la petición web (de manera análoga al caso del *bootstrap*), esto será explicado en el párrafo siguiente y guarda relación con el marcado de órdenes.

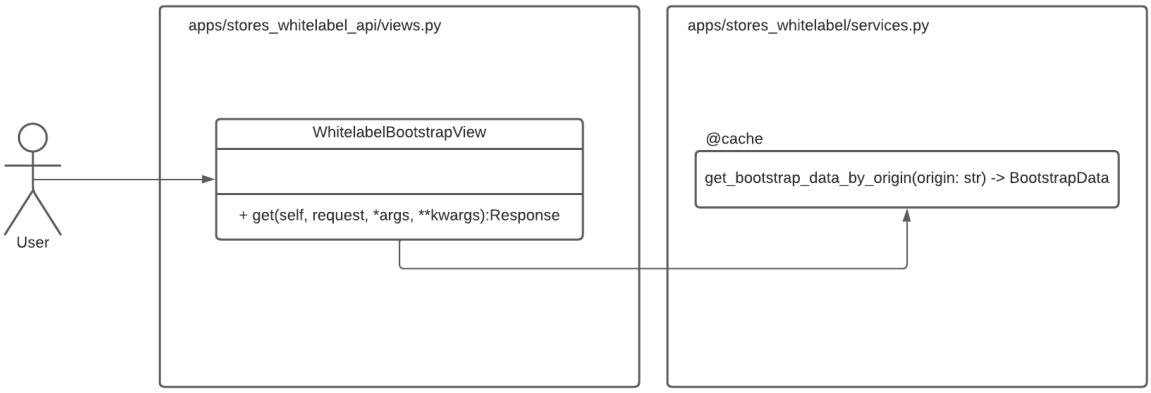


Figura 2.3: Diagrama UML de los servicios de *bootstrap*
Fuente: Elaboración propia

Respecto a los cambios realizados en el *backend* original de *Cornershop* se tiene solamente el relacionado con el marcado de órdenes [A5], para ello se modificó el flujo de creación de órdenes de *Cornershop* para, por un lado, poder identificar si es que la orden se está realizando desde una experiencia *Whitelabel* (utilizando el servicio descrito en el párrafo anterior, que chequea el origen de la petición web), y, por otro lado, para agregar esa información al flujo de datos donde se genera realmente la orden, es decir, a los modelos *Checkout* y *Order* ya existentes en el eco-sistema de la compañía. De esta forma, teniendo en cuenta este cambio, es posible realizar estadísticas en el sistema, pudiendo identificar si es que la orden fue generada en una experiencia *Whitelabel* y en caso de serlo, incluso identificar en cuál de ellas [P7].

### Capa de vistas

Con respecto al *frontend*, la aplicación *web* de *Whitelabel* es una versión (viene desde un *fork*[1] de *git*) de la aplicación *Cornershop* [A1] que, a vista del consumidor, funciona como si fuera una experiencia *single-store* (como si fuera una *app* de una tienda única), para lograr esto se utiliza toda la información entregada por el *bootstrap* de la capa de aplicación, esto es, el nombre de la tienda, iconos, logotipo, colores, entre otras características adicionales que fueron añadidas en el transcurso del proyecto. De esta manera, la experiencia *Whitelabel* representa la marca de la tienda, al menos dentro de ciertos parámetros básicos de la línea de *branding* [P2].

Si bien, como fue descrito en el párrafo anterior, la capa de vista de *Whitelabel* logra presentar un producto que se siente como una experiencia de tienda única, existe en esta capa otra solución adicional que busca solventar uno de los problemas generados al permitir que esta experiencia sea accedida desde un dominio externo al de la aplicación principal de Cornershop [A2], esta guarda relación con el manejo de sesión de usuarios en la aplicación *Whitelabel*. Es importante notar que, tal como fue mencionado en el apartado Marco teórico1.2,el manejo de sesión de usuarios por medio de *cookies* (medio de autorización que se usa en la aplicación web) no puede ser utilizado, ya que estas, por motivos de seguridad y por

[1]Copia de un repositorio Git.

definiciones de protocolos web [18], solo pueden ser accedidas desde el mismo dominio web en el que fueron generadas; por esto, de manera similar a como se procede en la aplicación móvil de *Cornershop*, el manejo de sesiones se hace a través de un cliente *Oauth2*.

*Oauth2*, es un protocolo de seguridad, que permite a una aplicación acceder a un recurso de otra en nombre de un usuario final, en este caso, autorizar a *Whitelabel* para acceder a los recursos de *Cornershop* en nombre del consumidor. Para explicar este protocolo, se definen 4 agentes:

- *Resource Owner* o usuario final: agente dueño del recurso al cual se va a acceder, normalmente es el usuario final y el recurso puede ser, por ejemplo, la información básica del usuario.
- *Client* o cliente: aplicación que desea acceder a un recurso de la aplicación externa.
- *Authorization Server*: capa encargada de gestionar la autorización usando el protocolo. Es ella la que autoriza al *Client* para acceder al recurso final.
- *Resource Server*: aplicación que almacena el recurso. Para permitir el acceso al recurso, este consulta al *Authorization Server*.

La autorización en *Oauth2*, de manera simple, se gestiona a partir de un *Access Token* o token de acceso, esto es un componente de seguridad usado para autorizar alguna acción, normalmente es utilizado en forma de texto, encriptado. Luego de que el cliente envíe este token, el *Resource Server* realiza una consulta al *Authorization Server* para validar el permiso de la aplicación externa.

En la actualidad, este protocolo define 7 modos para obtener el Access Token[21] (aunque 2 de ellos se encuentran obsoletos), estos son los llamados *grant types*, aun así, para propósitos de este proyecto basta definir el *Authorization Code Flow*, el cual es el flujo de obtención de token utilizado en la Whitelabel App. Este tipo de autorización, tal como es definido en el documento RFC del protocolo [2] provee más beneficios, en cuanto a seguridad respecto a los otros, pues evita que la transmisión del *token* de acceso se realice directamente al navegador del *Resource Owner*, lo cual potencialmente puede exponer el *token* a terceros; en vez de eso, se pasa el *token* al *Client*.

Este flujo consta de 9 pasos, tal como se puede apreciar en la figura 2.4, para la autorización de recursos:

1. *Resource Owner* desea acceder a información de su cuenta en el *Resource Server*. Para este caso en específico, hace clic en un enlace de *login* con credenciales *Cornershop*.
2. *Client* redirige al *Resource Owner* a un *endpoint* de autorización del *Authorization Server*. Para indicar que aplicación solicita el acceso al recurso se utiliza un *Client ID* (ID de cliente) definido en el protocolo.
3. *Authorization server* redirige al *Resource Owner* a un sitio donde, el usuario autenticado en el servidor, puede dar consentimiento para que el *Client* acceda a su información.

4. *Resource Owner* autenticado en el *Authorization Server* da el consentimiento para que el *Client* acceda a sus recursos.

5. *Authorization Server* devuelve un *Authorization Code* al *Client*.

6. Con el *Authorization Code* el *Client* solicita al *Authorization Server* el *Access Token*, para ello, envía el *Authorization Code* recibido en el paso 5; además, envía credenciales *Client ID* y *Client Secret* para identificarse ante el servidor.

7. *Authorization Server* verifica la información enviada por el *Client*.

8. *Authorization Server* envía información al *Client* autorizándolo para usar los recursos solicitados. Para ello, entrega el *Access Token* y un *Refresh token* con el cual, en caso de que el *Access Token* expire, es posible solicitar un nuevo *Access Token*.

9. Ya con el *Access Token*, el *Client* puede hacer peticiones al *Resource Server* en nombre del *Resource Owner*. En este punto, la *Whitelabel App* puede acceder a los *endpoints* de *Cornershop* en nombre del usuario.

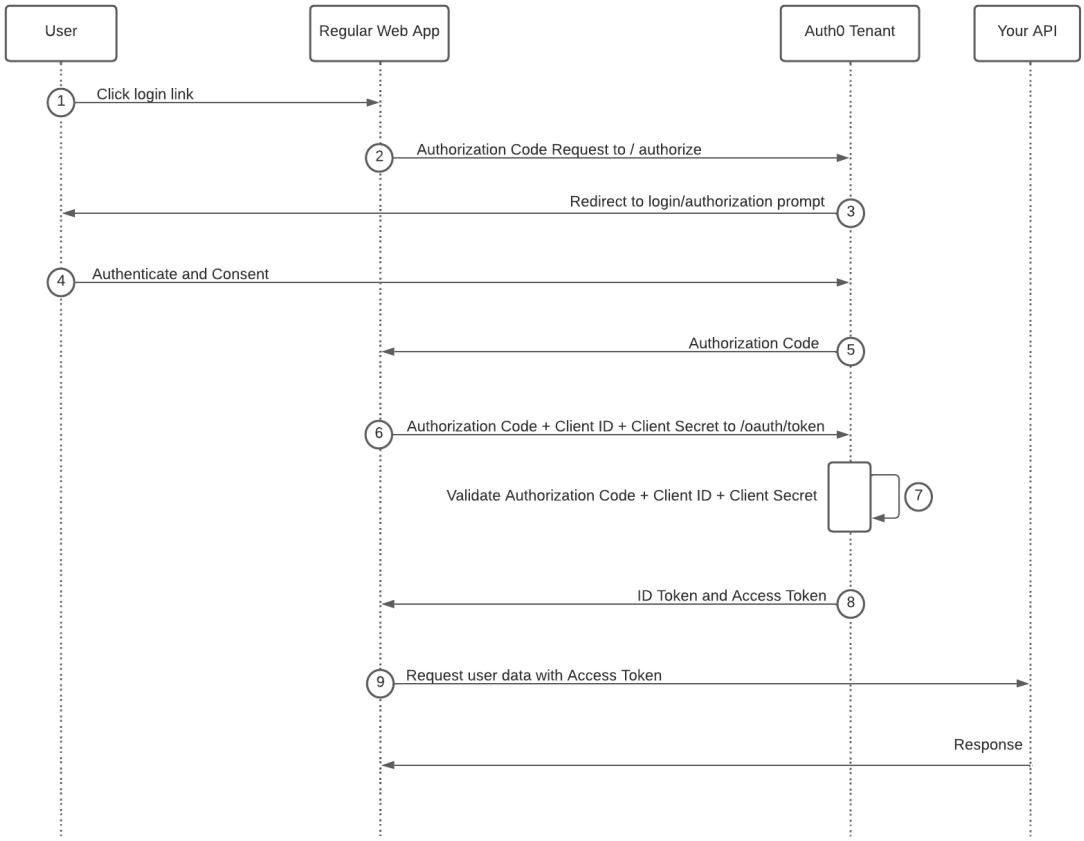


Figura 2.4: Diagrama de flujo de *Authorization Code grant type*
Fuente: Elaboración propia

Con este flujo en mente, el *frontend* de *Whitelabel* define un componente adicional de código que permite utilizar este protocolo como medio de autorización de usuario. Esta pieza fue diseñada para solventar únicamente la autorización mediante los sistemas internos de *Cornershop* y por ende solo implementa el flujo necesario de *Oauth2.0*, en específico, el *grant*

*type Authorization Code* mencionado en los párrafos anteriores. Este componente reemplaza el manejo de sesiones de usuario de la *Customer App* de *Cornershop* por el definido en *Oauth2.0* y define 4 funciones básicas basadas en el flujo mencionado anteriormente:

- Un mecanismo que, de manera similar al proceso de ingreso de usuario, redirige al usuario al *endpoint* de autorización de *Oauth2.0* (paso 2).
- Una vista para recibir el *Authorization Code* por parte del *Authorization Server* (paso 5).
- Configuración de la petición del *Access Token* luego de haber recibido el *Authorization Code* (paso 6 y 8).
- La petición del *Access Token* a partir del *Refresh Token*, este paso no está mencionado en el flujo, pero es parte de la extensión del protocolo que maneja los *tokens* con tiempo de expiración.

Como información adicional, para hacer uso de un cliente *Oauth2* en el sistema de *Cornershop* se debe inicializar una nueva instancia de cliente a través de la librería existente en el sistema previo. Este cliente, como fue mencionado, utiliza el tipo de concesión *Authorization Code*.

Si bien *Oauth2* no es parte directa de la arquitectura lógica de la aplicación, es fundamental haber definido esta pieza, pues es mediante este protocolo que, la capa de vistas de *Whitelabel*, puede acceder a los servicios del *backend* de *Cornershop*, tal como lo estipula la arquitectura definida.

Luego de definir la arquitectura lógica de *Whitelabel*, muchas de las inquietudes iniciales del proyecto fueron resueltas, sin embargo, aún no se resuelve una de las problemáticas iniciales del proyecto, cómo conectar la aplicación con un dominio web externo [A2].

### 2.1.3. Infraestructura de la aplicación

La infraestructura de *Whitelabel*, en oposición a la arquitectura, guarda relación con aquellos componentes que permiten llevar el producto final al consumidor, o, como es definido por RedHat [14], los elementos necesarios para operar y gestionar entornos de *software*. Teniendo en cuenta que esta solución debe ser accedida desde distintos orígenes web [P3], resulta lógico concluir que es fundamental para el éxito del proyecto definir una "arquitectura" de infraestructura robusta y suficiente.

#### Whitelabel Cloudflare DNS

Para poder acceder desde dominios web externos, *Whitelabel* hará uso de *Cloudflare*[2], el cual es un *CDN*[3], es decir, un proveedor de contenido web. Esta aplicación funciona como

[2]https://www.cloudflare.com/es-es/
[3]*Content delivery network*

una puerta intermedia entre la dirección web provista por el cliente y el sistema interno de *Cornershop*, tal como se puede apreciar en la figura 2.5. Dentro de las ventajas que ofrece esta aplicación se tiene, por un lado, su fuerte foco en seguridad de datos online y escalabilidad, además un conjunto de funcionalidades relacionadas con la entrega de contenido en internet. Es en específico una de estas funcionalidades la cual es fundamental para el proyecto, esta corresponde al servicio *DNS* provisto por *Cloudflare*, el cual permite configurar los flujos de peticiones web a los servidores de *Whitelabel*, de este modo es posible acceder al servidor central (en este caso la aplicación de *frontend*) desde un conjunto de direcciones web, los cuales en teoría son los orígenes provistos por los clientes de *Whitelabel*. Tal como fue mencionado a través del documento, esta característica lograda con *Cloudflare* es uno de los focos fundamentales del proyecto [A2]. Cabe destacar, respecto a esta aplicación, que *Cloudflare* ya formaba parte del conjunto de herramientas de infraestructura utilizadas por *Cornershop* y, por lo tanto, no añadía complejidad técnica o monetaria al acceder a estos servicios.

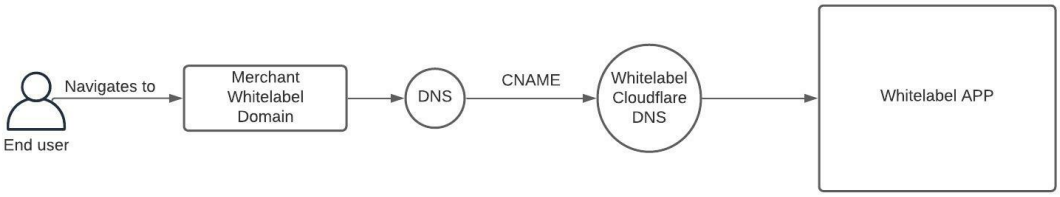


Figura 2.5: Diagrama de flujo de infraestructura de *Whitelabel*
Fuente: Elaboración propia

Respecto a las configuraciones realizadas en *Cloudflare*, estas fueron asistidas por un equipo encargado de infraestructura de la empresa, aun así, es importante entender el funcionamiento de estas para poder asegurar que se cumpla la funcionalidad esperada del producto. Así entonces, el flujo conseguido a través de las configuraciones, es el siguiente:

1. El usuario navega al sitio web de la tienda.
2. El sitio web, por su parte, configurado a través de un DNS, debe apuntar, utilizando un registro *CNAME*[4] a los servidores DNS de *Whitelabel* provistos por *Cloudflare*.
3. *Cloudflare*, en su configuración *DNS*, redirige las peticiones que lleguen a partir de un registro *CNAME* con cierto formato ("*.whitelabel.superpal.com") a la dirección oculta donde se encuentra el servidor de la aplicación *Whitelabel*, es decir, al *frontend*. Respecto al formato mencionado, esta es una expresión utilizada para chequear patrones en texto, que comienza con un *wild card*[5] y por consiguiente, acepta cualquier petición que finalice en ".whitelabel.superpal.com", a modo de ejemplo, "anypage.whitelabel.superpal.com", sí sería aceptado por el DNS para este flujo.
4. Finalmente, el usuario accede a la aplicación personalizada *Whitelabel*.

[4]Registro *DNS* que permite apuntar un dominio web a otro.
[5]Símbolo utilizado para representar, en patrones de texto, cualquier secuencia de caracteres.

Sobre el flujo recién mencionado, cabe destacar que este redireccionamiento, provisto por el servidor DNS, mantiene la información web (encabezados de mensajes HTTP) respecto al origen de la petición web, con lo cual, como fue visto en la sección de arquitectura lógica 2.1.2, *Whitelabel* puede ofrecer la experiencia personalizada por tienda.

Como ventaja adicional de la herramienta, *Cloudflare* puede ser configurado para proveer certificados SSL en sus conexiones. De esta forma, el usuario puede navegar a sus sitios utilizando el protocolo HTTPS, este es una versión más segura del protocolo base del internet moderno HTTP, a diferencia de su antecesor, este busca asegurar que la conexión establecida entre el usuario y el servidor sea privada (para ello utiliza TLS, un protocolo criptográfico) [P6]. Es tanta la importancia de poder asegurar una conexión que use este protocolo, que desde hace algunos años, los navegadores web más conocidos, como es el caso de Google Chrome [6], marcan las conexiones no HTTPS como conexiones inseguras, es así, como a día de hoy, no es una buena estrategia de *branding* que el sitio web de una empresa tenga esta advertencia [C2].

Si bien, por medio de *Cloudflare*, ya se resuelve uno de los principios fundamentales de este proyecto, el del acceso mediante direcciones web externas [P3], aún queda una problemática relacionada con el mismo principio y que nace como una mezcla de dos características del software a desarrollar:

- *Whitelabel* reusa los servicios existentes en el *backend* de *Cornershop* (y los extiende).
- La solución, a través del DNS y *Oauth2*, puede ser accedida desde un dominio externo.

Dado que estamos reusando la capa de aplicación de Cornershop [A1] y esto existe en un dominio distinto al del producto final de *Whitelabel*, que puede ser accedido desde sitios web externos, surgen problemas dado que se harán peticiones web entre dominios. Por ende, la solución *Whitelabel*, debe ser capaz de adaptarse al mecanismo de seguridad entre peticiones web inter-dominio CORS, mencionado en el apartado Marco teórico 1.2. Para poder introducir la solución relacionada con este mecanismo es crucial entender qué es el CORS y cómo funciona.

### Whitelabel CORS Middleware

CORS es un sistema que, mediante un intercambio de encabezados HTTP, determina si un navegador web permite que *JavaScript* acceda a la respuesta obtenida de una petición web inter-dominio. Este es un mecanismo que permite a un servidor elegir desde que orígenes se le pueden realizar peticiones HTTP. En la actualidad es tan relevante que la mayoría de los navegadores web implementan este protocolo[6]; de hecho, en el momento en que se escribe este documento, solo *Opera Mini*, un navegador para móviles, no implementa este sistema. CORS es entonces un mecanismo de seguridad adicional, que busca proteger al usuario y servidor de peticiones inter-dominio. En caso de que se realice una petición web a un servidor el cual no tenga registrado el dominio de origen en su lista blanca[7] el navegador bloqueará el acceso a su respuesta y lanzará un error en la consola, tal como se muestra en la figura 2.6.

[6]https://caniuse.com/cors

[7]Listado de direcciones que se admiten para el uso del protocolo.


Figura 2.6: Ejemplo de error de CORS en navegador
Fuente: Elaboración propia

Respecto al funcionamiento del sistema descrito, lo más básico que se espera en el flujo de CORS es que en el intercambio de encabezados HTTP por parte del servidor se añada el siguiente: "Access-Control-Allow-Origin", especificando, como valor, el origen del cliente en caso de que esté permitido (a modo de ejemplo: "https://foo.example"). En muchos casos, como en la figura 2.7 se utiliza el valor "*" indicando que la petición puede ser realizada desde cualquier origen. Solo para agregar un poco más de contexto sobre el sistema, tal y como es definido por Mozilla [17], CORS especifica casos de uso más complejo que el que se muestra en este documento, incluso cuenta con otros encabezados HTTP que añaden más funcionalidad a este sistema, entre ellos los más conocidos son:

- *Access-Control-Allow-Origin*: permite definir los orígenes respecto del cual el servidor espera recibir peticiones, tal y como fue mencionado previamente.
- *Access-Control-Allow-Methods*: utilizado en peticiones de *pre-flight*[8], permite al servidor especificar los métodos HTTP que permite las peticiones inter-dominio.
- *Access-Control-Allow-Headers*: utilizado en peticiones *pre-flight*, permite al servidor especificar los encabezados HTTP permitidos para enviar en las peticiones al servidor.

Más detalles sobre el protocolo completo y todas las funcionalidades adicionales que provee pueden ser encontrados en el documento escrito por Mozilla [17]. Sin embargo, para propósitos de este documento basta comprender el caso básico, el cual se encuentra representado en la figura 2.7.

Como fue descrito en la sección Marco teórico 1.2, el sistema actual de *Cornershop* ya cuenta con una herramienta para implementar CORS: *django-cors-headers*. Esta librería, si bien implementa el protocolo de manera robusta, no satisface una de las necesidades del producto; esta librería define los orígenes admitidos para CORS de manera estática, es decir, como un parámetro adicional en las configuraciones, directo en el código que usa *Django*. Teniendo en cuenta que cada tienda nueva de *Whitelabel* va a registrar un dominio nuevo en la aplicación, utilizar esta configuración implicaría realizar cambios directos en el código de la aplicación. Ya que existe un proceso para subir cambios al código, por los estándares de la compañía y por las herramientas de integración, sería un proceso que no soportaría grandes magnitudes de nuevos usuarios en la aplicación y, por lo tanto, no es escalable.

Es por la problemática anterior que se decide generar un nuevo componente que pueda hacerse cargo de las peticiones web inter-dominio en este proyecto, de modo que estas sean escalables y eficientes: *Whitelabel CORS Middleware*.

[8]Pequeña petición web que se envía antes de la petición real. Utilizada para obtener información del servidor.

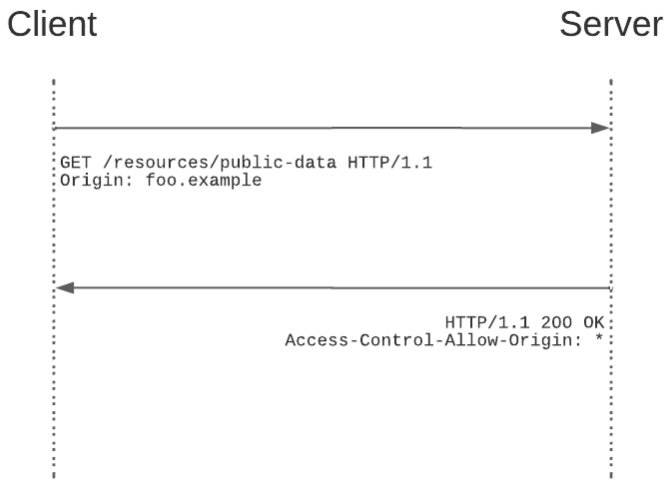


Figura 2.7: Ejemplo básico de petición CORS
Fuente: Elaboración propia

Antes de definir el funcionamiento del componente, es importante comprender cuál es el flujo de una petición web en *Django*, ya que es bajo este *framework* que es definido el *backend* de *Whitelabel*. Este flujo se encuentra representado en la figura 2.8 y consta de los siguientes pasos:

1. El cliente intenta acceder a un recurso a través de una petición web.
2. La petición web es servida entonces por el servidor web, componente encargado de responder peticiones web de clientes en internet. En este caso, el servidor web por defecto para *Django*, es *Apache*[9].
3. La petición pasa a través de un servidor WSGI, el cual es una interfaz que asegura que las peticiones web en *Python* cumplan con un estándar [23]. *Django* tiene por defecto un servidor WSGI propio.
4. La petición es revisada por el *middleware* de Django, el cual consta de piezas de código intermedias, encargadas de manipular o verificar las peticiones web del sistema a un bajo nivel. Una instancia de *middleware* utilizado en el sistema de *Cornershop* y mencionado anteriormente es *django-cors-headers*.
5. Luego de recibir la petición, el componente *router* asigna la petición a una vista (*view*) registrada en las rutas de *Django*.
6. La vista es la encargada de crear una respuesta para el usuario, a través de los servicios provistos por el *backend*. En la figura 2.8, se simplifica la funcionalidad de la vista y se muestra una conexión directa a *Django ORM*, cuando, en muchos casos, se accede a más componentes en el sistema. La respuesta, en caso de ser un archivo HTML, se arma partir de los *templates* de *Django*, estos permiten armar páginas *web* de manera dinámica.

[9]https://httpd.apache.org/

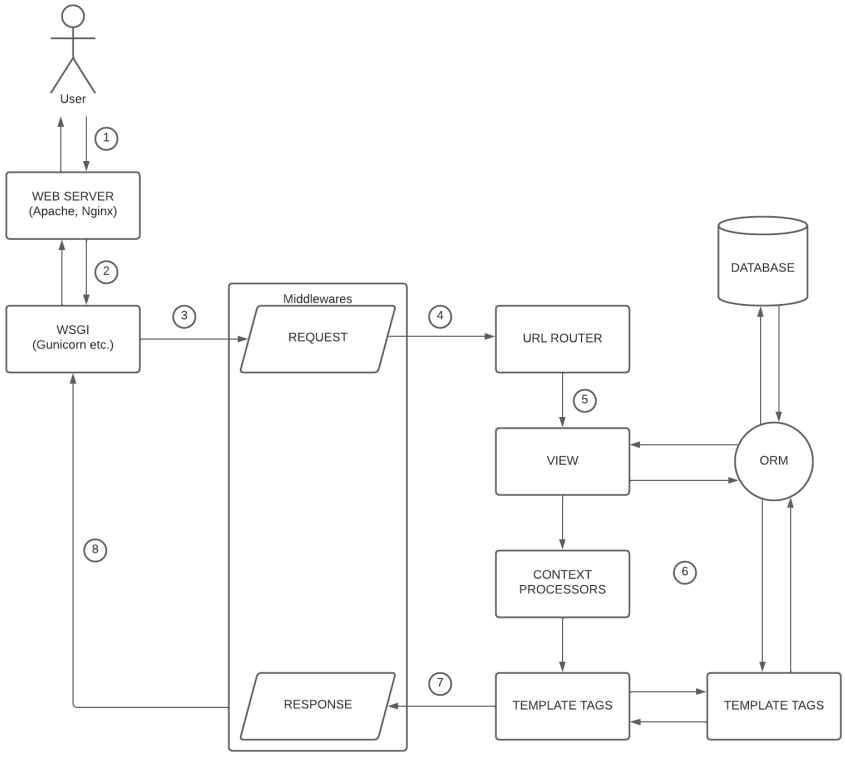


Figura 2.8: Diagrama del flujo de una petición web en *Django*
Fuente: Elaboración propia

7. La respuesta pasa nuevamente por el *middleware*, donde el servidor de *Django* tiene acceso de bajo nivel a la respuesta de la petición web. En este instante, el *middleware* CORS agrega los encabezados HTTP necesarios para el funcionamiento del protocolo.

8. Finalmente, la respuesta es entregada al cliente, pasando por WSGI y el servidor web.

Luego de entender cada uno de los elementos que componen el flujo, se decide crear un componente que, de manera análoga a la herramienta *django-cors-headers*, formará del *middleware* que conforma la aplicación.

*Whitelabel Cors Middleware*, componente originado bajo este proyecto, está encargado de servir las mismas necesidades de un proveedor CORS, pero, a diferencia de la librería que ya posee *Cornershop*, el componente permite al servidor adaptar dinámicamente el listado de orígenes admitidos por el servidor. Para verificar este lista dinámico, el *middleware* utiliza el servicio que provee el listado de las direcciones web de las tiendas *whitelabel* descrito en el apartado de aplicación 2.1.2.

Si bien esto soluciona el problema inicial por el que se profundizó en el tema CORS; un cambio en un componente del *middleware*, teniendo en cuenta el flujo de una petición en *Django*, afecta todas las peticiones web que reciba la aplicación y a su vez, no todas las rutas ("backend.example/route") de peticiones que pueda recibir el *backend* serán usadas por *Whitelabel*, puesto que, como fue mencionado anteriormente, la capa de aplicación de Cornershop es un sistema masivo y complejo, que sirve a un grupo de aplicaciones como lo

es *Cornershop App*, *Stores Center*, *Shopper App*, entre otras aplicaciones relacionadas con la experiencia *Cornershop*. Es entonces que, otra característica de este nuevo componente, es que está diseñado para actuar únicamente sobre las rutas que son utilizadas por *Whitelabel*, para ello se hace uso de otra característica de *Python*: *decorators*. Los decoradores, si bien nacen de un patrón de diseño con el mismo nombre [22], es una forma en *Python* para agregar funcionalidad a diferentes objetos y; para el caso de *Whitelabel*, el decorador creado, de nombre "*whitelabel_view*", marca las vistas de *Django* que son parte de este proyecto. Se adjunta a continuación extractos del código relacionado con el componente nuevo.

```
1
2 def whitelabel_view(obj: Callable) -> Callable:
3     setattr(obj, CORS_WHITELABEL_CHECK, True)
4     return obj
5
6 @whitelabel_view
7 class FakeView(View):
8     # do something
9     ...
```

Listing 2.1: Código del decorador y ejemplo de uso
Fuente: Elaboración propia

Posterior al desarrollo del *middleware* se realizó una investigación para identificar todas las rutas del *backend* de *Cornershop* que son de utilidad para *Whitelabel*, si bien este documento es privado, es posible mencionar que en él se pudieron identificar un total de 89 vistas que eran usadas por *Whitelabel* y por ende debían ser marcadas con el decorador.

De esta forma, el *middleware* es capaz de adaptar los orígenes permitidos por el servidor (CORS) a los sitios web que las tiendas hayan definido para su experiencia *Whitelabel*, evitando alterar el funcionamiento de otras partes del sistema de *Cornershop* que no sean relevantes para este proyecto. Así, este *middleware*, en conjunto con el DNS de *Cloudflare*, permiten que la experiencia *Whitelabel* sea completamente funcional al ser accedida de un dominio web externo [P3].

La arquitectura del sistema presentada, como una composición de la arquitectura lógica más la infraestructura, permitieron definir las cualidades esperadas de *Whitelabel E-commerce*. En la siguiente, se definen ciertos desafíos técnicos que se presentaron a la hora de implementar la solución.

## 2.2. Desafíos Técnicos

A la hora de desarrollar una solución de *software*, aunque ya se haya definido la arquitectura general de este, es esperable encontrar ciertas problemáticas o complicaciones que tienen que ver con cómo se desarrolla técnicamente el software, es decir, desafíos que surgen en paralelo a la arquitectura de la solución. En esta sección del documento se describirán, entonces, los principales desafíos técnicos encontrados al desarrollar *Whitelabel*.

Uno de los primeros desafíos que se encontró tiene que ver con el acoplamiento a los estándares internos de *Cornershop*. Como fue descrito en el apartado Marco teórico 1.2, la arquitectura lógica esperada de las soluciones, en cuanto a Ingeniería de Software, se encuentra en un documento llamado "Well sized services"[C15]. Esta arquitectura, tal como ya fue descrito, es definida siguiendo una estructura de N-capas, siendo estas: API, servicios, proveedores, y modelos de datos. Esta guía, que busca asegurar un código mantenible y escalable a través de la empresa, es la primera barrera de desafíos que se debió superar a la hora de implementar la solución.

Otro desafío encontrado al desarrollar la infraestructura del producto está relacionado con: el DNS de *Whitelabel*, el protocolo CORS y el *middleware* posteriormente creado. Estos tres, en conjunto, permiten acceder a la aplicación desde dominios externos, así, el desafío planteado en este caso surge al tener que desarrollar un componente que cumpla con los estándares planteados en el documento [17], esto requirió, tanto lectura, como análisis e implementación de una pieza de software que funcionará con otras aplicaciones que implementan el protocolo CORS, como es el caso de la mayoría de los navegadores web que existen a día de hoy[10]. Cuando se trabaja con algún estándar o protocolo de internet, siempre hay que probar los casos de uso en un entorno real, puesto que, tal como lo menciona un *meme* de internet, o tal como el homónimo *podcast* [12]: "*Internet is made of duck tape*". Existen muchas herramientas de internet que, dependiendo de quién las haya programado, funcionan de distintas formas, es por esto que, luego de leer el estándar y desarrollar el componente, se tuvo que probar en distintos navegadores la solución del *Whitelabel CORS*.

De forma análoga, vale la pena destacar la implementación en la vista de un componente que permitiera adecuarse al protocolo *Oauth2.0*, en específico, al *Authorization Code Grant*; esta implementación, si bien era simple en su desarrollo, era suficientemente robusta para funcionar con el servidor *Oauth2.0* presente en el sistema de *Cornershop*, de esta forma, los usuarios de *Whitelabel* podían utilizar su usuario *Cornershop* para navegar por la aplicación.

Los siguientes desafíos tienen que ver estrictamente con asegurar la escritura de un buen *software*, es decir: escalable y seguro. La cualidad mantenible no será considerada en los siguientes párrafos, puesto que dado que el código cumple con los estándares internos de la compañía [P5], se considera que estos aseguran esta cualidad.

Respecto a la escalabilidad del sistema, si bien, parte de lo que se considera al desarrollar *software* en la compañía es revisar que el código asegure el rendimiento del sistema, vale la pena mencionar 3 soluciones del proyecto que son cruciales para el mismo, dado que están relacionados con funcionalidades del sistema de gran cadencia de uso. Se tiene entonces 2

[10]https://caniuse.com/cors

soluciones relacionadas con optimizar consultas de datos y 1 solución relacionada con limitar el alcance, respecto a rendimiento general, de *Whitelabel*, en el sistema de *Cornershop*.

De las soluciones relacionadas con la optimización de consultas, se tienen los siguientes casos:

- Al generar una orden *Whitelabel*, se chequea constantemente que el origen de esta corresponda a una dirección registrada en el modelo, de esta forma es posible marcar las órdenes generadas en esta experiencia. Teniendo en cuenta que la mayoría de las operaciones a este modelo serán de búsqueda sobre el campo "*redirect_store_url*", se decide agregar un índice a este en la base de datos. Es así que, consultas de búsqueda al modelo *WhitelabelConfig* por su origen, aumentan notablemente su velocidad y uso de recursos, según documentos de la compañía IBM [16].
- De manera similar, se tiene un caso de optimización relacionado con el componente *Whitelabel CORS Middleware*. Este, para obtener el listado de orígenes admitidos por el protocolo, utiliza una consulta sobre todos los dominios web almacenados en la tabla *WhitelabelConfig*, sin embargo, dada que esta consulta es pesada en tamaño y que tendrá una gran cadencia de peticiones, se decide agregar esta consulta, a un *caché* del sistema interno de *Cornershop*, así se evita sobrecargar la base de datos con consultas que, dada su naturaleza, son recurrentes y seudo-estáticas, es decir, que no cambian su respuesta en periodos cortos de tiempo.

Tal como fue descrito en el apartado 2.1.3, añadir un *middleware* al sistema, implica, inicialmente, un software con la capacidad de afectar todas las peticiones web recibidas al sistema de *Cornershop*, es así que, como exigencia de infraestructura de la compañía y con la intención de limitar el alcance del proyecto en el sistema global, se diseña el componente para operar únicamente sobre un subconjunto del total de *endpoints* del sistema: solo las rutas utilizadas por *Whitelabel*. Tal como fue mencionado en la sección mencionada, estas corresponden a 89 vistas de la totalidad del sistema de *Cornershop*.

El pilar fundamental de esta solución, respecto a seguridad, guarda relación con el uso de *Oauth2* para las sesiones de usuario, pues es a través de esta que se guarda, por un lado, la identidad virtual de los usuarios y por otro, la seguridad de los datos en el sistema, pues, todos los *endpoints* capaces de modificar u obtener datos cruciales, como es el caso de la información del usuario o la creación de órdenes, requieren a un usuario autenticado. La elección de *Oauth2* como medio de autorización para *Whitelabel*, si bien nace de la comodidad de ser un componente preexistente en el eco-sistema de *Cornershop*, va acompañado del gran uso que se hace de este a nivel global, teniendo compañías de gran renombre que ofrecen autorización o autenticación de usuario a través de este protocolo, solo por mencionar una, se tiene el caso de *Google*, que incluso presenta una guía de su cliente *Oauth2* [13].

Aunque el sistema sigue estándares de seguridad utilizados masivamente en el mercado actual, existe un punto negativo respecto a la elección de *grant type* de este protocolo, en otras palabras, se identifica una falencia relacionada al tipo de concesión utilizada en *Oauth2*. A día de hoy, se especifica en el documento publicado en IETF[11] [7], que *first party apps*, es

[11]https://www.ietf.org/

decir, aplicaciones que pertenezcan a la misma empresa encargada de suministrar el servidor *Oauth2*, deben implementar y usar *Proof Key for Code Exchange* (PKCE [5]), una extensión al protocolo que previene ataques donde el *Authorization Code* es interceptado por un agente malicioso (que se ha verificado correctamente con las credenciales del *Client* original). Esta misma idea se menciona en el sitio oficial de *Oauth2* [20], donde se especifica la deficiencia de seguridad en aplicaciones de navegador respecto a que estos no tienen una forma segura de guardar las credenciales de *Client* para el protocolo. Si bien desde un inicio se identifica la necesidad de utilizar este tipo de autorización para *Whitelabel*, por decisiones de costo [C21] y teniendo en cuenta que la versión de la librería utilizada para montar un servidor *Oauth2* (tal como fue mencionado en el apartado teórico 1.2,*django-oauth-toolkit*) no implementaba este protocolo, se decide dejar esta deficiencia de seguridad como una deuda técnica para posteriores versiones de la aplicación.

## 2.3. Estado actual de la solución

Para finalizar la sección destinada al desarrollo del proyecto, se mencionará en los siguientes párrafos el estado actual de la solución en relación con las inquietudes iniciales del mismo.

La solución *Whitelabel* implementa todas las características definidas en su diseño (infraestructura y arquitectura lógica). Revisando las decisiones de arquitectura definidas en un inicio (en la tabla 2.1), se tiene que la solución implementada cuenta con lo siguiente:

- Dentro de la arquitectura lógica de la aplicación, un servidor de servicios o *backend* que hace uso y extiende los componentes existentes de Cornershop y una capa de vistas, que es una adaptación de la aplicación de la compañía pero enfocada en una experiencia de tienda única [A1].
- Un servidor central, donde se provee la solución, que puede ser accedido desde distintos dominios [A2], este es una mezcla entre el *Whitelabel CORS Middleware* y el uso de*Cloudflare*.
- Un sistema de personalización de cada experiencia, destinado a impulsar el *branding* de cada tienda [A3], esto fue logrado a través del modelo de datos de la aplicación, el cual permite almacenar una configuración básica de cada tienda a partir de la dirección web de cada una.
- Servicio de entregas de pedido impulsado por la compañía base, es decir, hace uso de los *pickers y drivers* de *Cornershop* [A5].
- Un sistema para marcar las órdenes provenientes de la experiencia *Whitelabel*, de forma que sea posible realizar estadísticas adicionales respecto a estas mismas [A5]. A modo de ejemplo, una de las estadísticas esperadas al lanzar este proyecto era: "Cantidad de órdenes realizadas en *Whitelabel* en una tienda X v/s cantidad de órdenes realizadas en la aplicación *Cornershop* para la misma tienda".

Aun cuando todo lo respecto al diseño fue cubierto, existe una característica deseable, derivada de una de las inquietudes iniciales, que no llegó a ser diseñada, pues era un requerimiento de proyecto para estados más avanzados del mismo. Esta tiene que ver con el desarrollo de una plataforma que permitiera a las tiendas configurar su experiencia *Whitelabel* de manera personal [C8], esto es la implementación de una interfaz de usuario que permitiera exponer un *CRUD*[12] a las tiendas usuarias del producto. Lo esperado bajo esta característica es servir las mismas funcionalidades que se realizan de manera manual en el estado actual de la aplicación en otra de las micro-aplicaciones del eco-sistema de *Cornershop*, esta podría desarrollarse como una extensión de *Stores Center*.

*Whitelabel*, como proyecto, fue probado en entornos de prueba en múltiples ocasiones, esto con el afán de asegurar su funcionamiento como un producto que ofreciera la experiencia de mercado online [P1]. Para ello, tal como se puede apreciar en la figura 2.9, se utilizaron tiendas (en este caso, esta tienda se llamaba *Whitelabel Store*) en conjunto con direcciones

[12]Interfaz que reúne las operaciones básicas de base de datos sobre un modelo. *Create*, *Read*, *Update*, *Delete*.

web adquiridas para realizar la conexión a través del servidor DNS. De esta forma, y con ayuda de equipos destinados a probar la aplicación (QA[13]), es posible asegurar *Whitelabel* como un producto listo para ser usado.

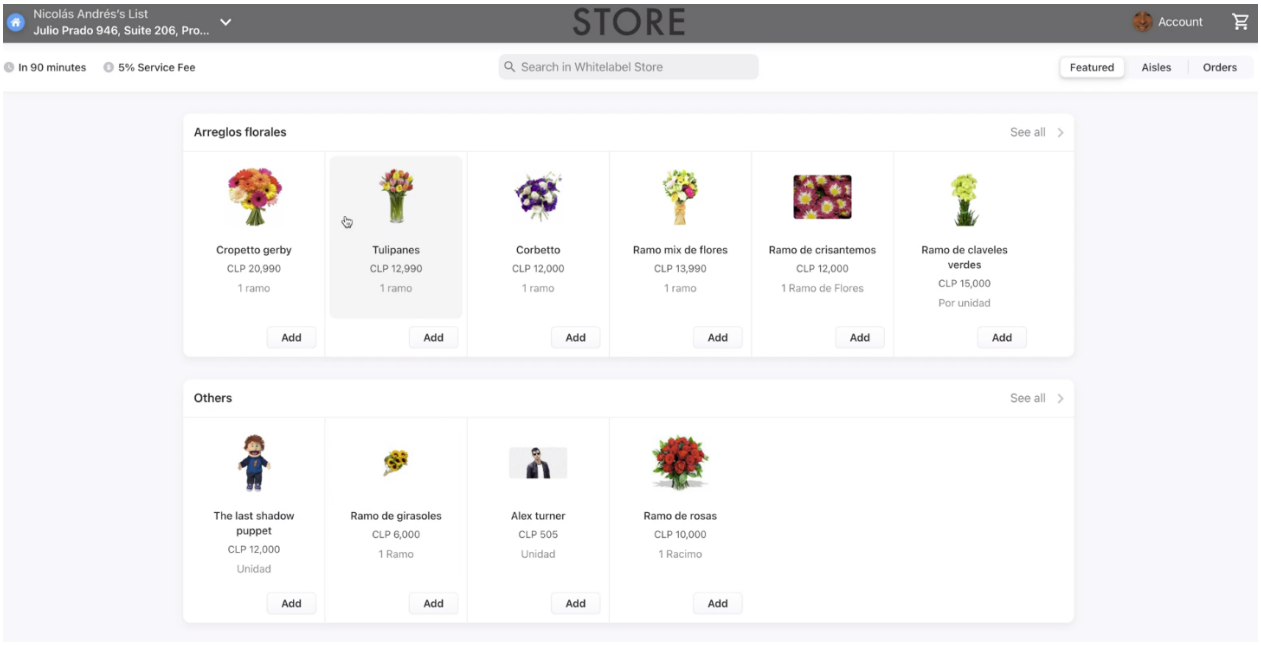


Figura 2.9: Captura de pantalla de tienda *Whitelabel* de prueba
Fuente: Elaboración propia

Por último, teniendo en cuenta el estado actual de la solución, es importante recalcar que, si bien se evaluó la solución en un ambiente de prueba y uno real (esto será revisado en la sección evaluación del documento), dado el estado de negocio de la compañía y por motivos internos, se decide priorizar otros proyectos en la empresa. Es así que, si bien *Whitelabel* se encuentra en un estado suficiente para ser usado por *Cornershop*, se encuentra como producto, en estado de pausa indefinida.

[13] *Quality assurance*, encargados de asegurar la calidad de un producto

# Capítulo 3

# Evaluación

Tal como fue especificado en secciones anteriores, la evaluación de este proyecto se dividirá en dos partes fundamentales, un que evaluará las inquietudes iniciales del proyecto, mediante la cual se verificará que la solución cumpla con las características esperadas del mismo; y, por otro lado, una prueba de concepto en un escenario real, mediante el cual es verificará que, la solución funcione y, más importante aún, mediante la cual los *stakeholders* del producto podrán tener información sobre el mismo, antes de lanzar el producto en el mercado.

## 3.1. Evaluación de características

Para esta evaluación, se hará uso de las inquietudes iniciales del proyecto, las cuales están especificadas en la tabla 1 del anexo. A partir de estas, se verificará una a una si estas fueron logradas (L) o no. Muchas de estas inquietudes fueron resueltas teniendo en cuenta que esta solución está basada en el desarrollo original de *Cornershop*, para este caso, se dice que estas inquietudes son resueltas por transitividad (T). Por otro lado, se ha realizado una encuesta a los *stakeholders* del proyecto de modo de medir su conformidad ante ciertas características esenciales del producto, las inquietudes aprobadas a partir de la encuesta se dirán como válidas por *stakeholders* (S). Finalmente, existen algunas inquietudes que no pueden ser explicadas por estos dos principios, estas serán acompañadas por comentarios al costado derecho de la tabla 3.1 para probar que las inquietud mencionadas fueron logradas satisfactoriamente.

La encuesta realizada a los *stakeholders* (con sus resultados) puede ser encontrada en el anexo 2. Respecto a esta, vale la pena destacar que estas preguntas están hechas a partir de un subconjunto reducido de las inquietudes iniciales del proyecto, en específico, aquellas que pueden ser probadas u observadas a nivel de usuario, es por esto que, por ejemplo, quedan afuera las inquietudes relacionadas a las estrategias de negocio o tecnológicas.

| Id | Resumen de inquietud | L | T | S | Comentarios |
|---|---|---|---|---|---|
| C1 | El sistema desarrollado debe ofrecer servicio a sus usuarios al nivel de Cornershop: buenos tiempos de respuesta, producto robusto y buena UX. | X | X | | |
| C2 | Se pierden clientes debido a que tiendas prefieren abrir su propia experiencia virtual, no solo vender por Cornershop. | X | | | Se espera que esta sea la característica fundamental del producto. Por ende, como el producto se considera suficiente, se considera logrado. |
| C3 | El sistema muestra un catálogo de los productos que ofrece la tienda. | X | X | X | |
| C4 | El sistema permite realizar compras/órdenes en la tienda. | X | X | X | |
| C5 | El sistema debe hacerse cargo del delivery de la orden al consumidor. | X | X | X | |
| C6 | El sistema posee un acceso al servicio al cliente. | X | X | X | |
| C7 | Aplicación debe poder ser accedida desde dominio externo suministrado por los clientes. | X | X | X | |
| C8 | El producto debe permitir la personalización para cada tienda; como mínimo: logotipo y colores. | X | X | X | Información del modelo de datos. |
| C9 | El sistema debe generar información sobre las órdenes generadas en la plataforma | X | | X | Marcado de órdenes. |
| C10 | El desarrollo de software debe estar a la cabeza del estándar internacional | X | X | | |
| C11 | *Cornershop es una empresa IT, por lo que los desarrollos deben aportar información al sistema general de Cornershop.* | X | X | | Se cumple, en parte, por el punto C9. |
| C12 | Software debe ser mantenible, siguiendo los estándares de Cornershop. | X | X | | Fue desarrollado bajo los estándares de la compañía |
| C13 | Software debe ser seguro. | X | X | | Además, se cumple al utilizar un protocolo aceptado de seguridad a día de hoy OAuth2 |
| C14 | Software debe asegurar tiempos de respuesta razonables. | X | X | | Consultas optimizadas a la base de datos y uso de caché. |
| C15 | El desarrollo debe seguir el estándar interno de “well-sized services”. | X | X | | |
| C16 | *Backend debe ser desarrollado en base Python/Django.* | X | X | | |

| | | | | | |
|---|---|---|---|---|---|
| C17 | Siguiendo la línea de Test Driven Development, el código debe ser cubierto por tests en al menos un 80 %. | X | X | | Es parte de los estándares de desarrollo de Cornershop. |
| C18 | Código debe seguir el estándar de código de Python PEP8 | X | X | | Es parte de los estándares de desarrollo de Cornershop. |
| C19 | La solución, al ser usada desde dominios externos, debe cumplir con el estándar del W3C - CORS. | X | | | Desarrollo de Whitelabel CORS Middleware |
| C20 | Se tiene un tiempo límite de 6 meses para hacer una prueba de concepto del producto en un escenario real. | X | | | Esto es parte intrínseca del proyecto; esto fue cubierto al seguir los de la compañía. |
| C21 | Los costos esperados del desarrollo deben ser bajos, equipo máximo de 10 personas. | X | | | |
| C22 | Conocimientos técnicos requeridos: frontend, backend, diseño e infraestructura. | X | | | Los equipos de trabajo estuvieron conformados por estos tipos. |
| C23 | Se espera que la solución opere funcionalmente en periodos de actividad regular en el continente americano. | X | X | | Es el mismo horario operativo de Cornershop. |

Tabla 3.1: Evaluación de características
Fuente: Elaboración propia

Todas las inquietudes iniciales del proyecto fueron logradas (L) ya sea por transitividad, por resultados de la encuesta (S) o, en otros casos, por una descripción lógica en la sección comentarios. Es posible decir, entonces, que el proyecto cumple con las expectativas y restricciones iniciales del mismo, aún más, siguiendo el proceso de definición de arquitectura, definido en el libro [1], dado que las inquietudes derivan en los principios y, a su vez los principios derivan en decisiones de arquitectura: *Whitelabel* cumple con las inquietudes, principios y decisiones de arquitectura definidos en su desarrollo.

Además, tal como fue definido en la sección 1.3.3, el éxito de esta prueba, implica que los objetivos específicos 1, 2, 3 y 4 también fueron conseguidos. Cabe recordar que estos objetivos guardan relación con las características esenciales del producto, las cuales están cubiertas por en las inquietudes iniciales (tabla 1) del mismo.

## 3.2. Prueba de concepto en escenario real

Luego de validar las cualidades de *Whitelabel*, la siguiente forma de evaluación es una prueba de concepto en un escenario real, es decir, lanzar la *Whitelabel App* de manera temporal en una tienda. A partir de esta prueba se obtendrán dos resultados que respaldan el producto y ofrecen mayor valor al proyecto, por un lado, probar la solución en ambiente de

producción y volviendo a la analogía de "*Internet is made of duck tape*", confirma que el producto es funcional, no solo en ambiente de prueba; por otro lado, y más importante, los *stakeholders* del proyecto, en específico los encargados de negocio de la compañía, podrán tomar una decisión - de negocio - respecto al lanzamiento del producto a nivel comercial.

Para realizar esta prueba de concepto, se utilizó un dominio de prueba comprado por la compañía ("https://whitelabel-prod.buenaonda.com") y se realizaron las configuraciones iniciales para montar la experiencia de *Whitelabel* en una de las tiendas interesadas en el producto: *AquaChile*, especializada en la venta de productos marítimos en Chile. Es importante destacar que, por simplicidad y comodidad, esta tienda ya formaba parte del eco-sistema de Cornershop y, por lo tanto, no requería configuraciones adicionales aparte de las necesarias para *Whitelabel*.

Respecto a las configuraciones, las cuales derivan de lo definido en el diseño de la solución, se tienen las siguientes:

- Configuración del dominio web para apuntar, mediante un registro DNS *CNAME*, a los servidores de *Whitelabel*.
- Registro del dominio web en los servicios de *Cloudflare*, de modo de poder manejar peticiones desde este dominio y poder suministrar certificados HTTPS válidos a los usuarios de la página.
- Inicialización de una instancia en la base de datos con la información de la tienda *Whitelabel*: dominio de origen, logotipo y colores.

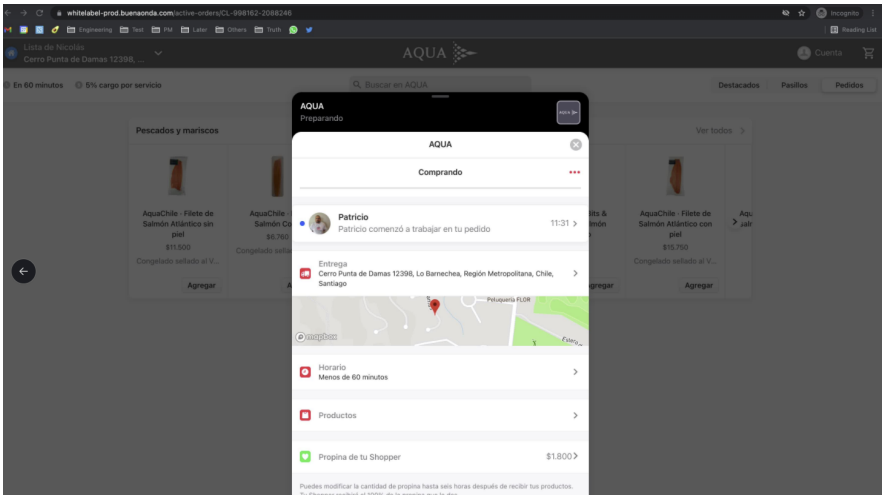


Figura 3.1: Captura de pantalla de *Whitelabel* en producción
Fuente: Elaboración propia

Luego de montar la aplicación *Whitelabel* para *Aqua*, tal como se puede apreciar en la figura 3.1, se realizaron pruebas de usabilidad sobre el sitio web, asistidas por un equipo de QA. Estas se centraron en la experiencia básica esperable de un comercio virtual, es decir, las funcionalidades definidas en el anexo 1, específicamente las relacionadas con los requerimientos: catálogo de productos, creación de orden y entrega de la orden.

Los resultados de esta prueba fueron satisfactorios, pues, por un lado, se verificó que el producto es funcional y por otro, se recaudó información sobre el flujo de esta experiencia y sobre su integración con el sistema original de *Cornershop*, el cual está encargado, en gran parte, de la funcionalidad base de la experiencia.

Esta prueba de concepto, en conjunto con las pruebas de usabilidad previas, permiten llevar el proyecto a un nivel funcional y suficientemente listo para ser lanzado (MVP[1]). Dada esta condición y, de manera análoga a lo explicado en la evaluación anterior, el éxito de esta prueba implica que el objetivo específico 5 fue logrado satisfactoriamente. Con esto, todos los objetivos específicos del proyecto se consideran logrados y aún más, es posible concluir que el objetivo general del mismo está logrado. Este objetivo representa la visión general que se espera del producto y por ende, es definido por el conjunto de los principios de este proyecto (tabla 1.1), entonces, dado que se cumplen las inquietudes y, a su vez, los principios, el objetivo general de este proyecto está logrado.

Finalmente, es posible establecer que *Whitelabel* se encuentra listo para ser lanzado al mercado, pues todos sus objetivos e inquietudes iniciales fueron cubiertas, sin embargo, el estado real del proyecto, a nivel de negocio, se encuentra pausado. En el transcurso de desarrollo de cualquier proyecto, no solo en el ámbito de software, es posible que una empresa cambie su motivación inicial y decida priorizar otros objetivos por cuestiones de negocio. Este es el caso de lo ocurrido en *Whitelabel*: un proyecto en espera, pero listo para ser lanzado.

[1]Producto viable mínimo.

# Capítulo 4

# Conclusión

## 4.1. Conclusiones generales

El proyecto *Whitelabel* nace como una herramienta que permite a tiendas, montar un comercio virtual propio asistido por una compañía que lleva unos años en el negocio: *Cornershop*. Esta experiencia permite a un usuario la posibilidad de realizar pedidos en una tienda virtual que tiene elementos estéticos que identifican el sitio con el *branding* de la tienda. De esta forma, la tienda, sin necesidad de conocimiento técnico en desarrollo de *software*, puede poseer una tienda virtual que, a la vez, le ayuda a impulsar el estatus de su marca.

En este documento, teniendo en cuenta que el proyecto está relacionado con el desarrollo de *software*, se ha abordado el diseño de la arquitectura e infraestructura sobre el cual se implementa *Whitelabel*. Este conforma, en término generales, una aplicación, inspirada y basada en la de *Cornershop*, y en un sistema que permite conectar la aplicación con un dominio web externo.

Para apoyar la definición de arquitectura de esta solución, se ha hecho uso de un proceso definido en el libro escrito por *Rozanski* y *Woods* [1]. Es a partir de este, que se ha presentado toda una estructura de inquietudes, principios y decisiones que explican las razones por las cuales se ha hecho el proyecto, lo que se esperaba del mismo y finalmente, sobre qué decisiones se ha basado la arquitectura. Este proceso aporta congruencia y fluidez entre las ideas del mismo, puesto que, cada definición de diseño propuesta en este documento, tiene su origen en alguna de estas propuestas iniciales. Aún más, dada la lógica de este proceso, cada decisión de arquitectura puede ser explicada por un conjunto de principios, y a su vez, cada principio puede ser explicado por un conjunto de inquietudes, estas últimas son, entonces, el pilar fundamental del proyecto. A partir de lo mencionado, es posible decir que el diseño de la solución de este proceso se ha realizado en un sistema lógico que nos permite encontrar el origen de cada idea.

El proyecto *Whitelabel*, en cuanto a arquitectura de *software*, ha definido todos los componentes que conforman la estructura final del producto, a partir de un proceso lógico y secuencial, tal como se menciona en el párrafo anterior. Además, esta arquitectura, consigue solventar todas las inquietudes y características esperadas del proyecto, las cuales son la base

del proceso de diseño utilizado. Es por estas razones que es posible afirmar que este proyecto ha tenido éxito al definir su arquitectura.

Por otro lado, a nivel de proyecto, ha conseguido cumplir con todos los objetivos propuestos en un inicio, ya sea mediante la evaluación de características o la prueba de uso en un entorno real. En resumen, dado que se encuentran logrados tanto objetivos generales como específicos, *Whitelabel* se considera un proyecto exitoso en cuanto ha conseguir lo que se ha propuesto y, aún más, son estos logros los que lo definen como producto listo para ingresar al mercado.

Si bien, *Whitelabel* es un producto funcional y está, en teoría, listo para ser lanzado al mercado, se encuentra en un estado de pausa por decisiones internas de la compañía. Congelar la salida de un proyecto provoca emociones no deseadas entre los encargados de desarrollar una aplicación, puesto que luego de realizar esfuerzos durante meses y conseguir armar un producto que satisficiera las motivaciones del proyecto, no continuar con una parte fundamental del mismo es decepcionante. Aun así, lo sucedido con este proyecto no es un caso particular, y de hecho, es común en una compañía que ciertas iniciativas se congelen a medida que se desarrollan, pues, tal como ocurrió en este caso, las prioridades de la empresa pueden cambiar.

## 4.2. Impacto

En los inicios de este escrito se ha planteado la posibilidad de identificar un nuevo tipo de producto, de reciente integración en los mercados, estos son los *Whitelabel E-commerces*, aún más, se ha ahondado en esta temática y se ha logrado desarrollar este bajo los términos de la compañía. Bajo esa premisa, el desarrollo de este proyecto ha abierto el término *Whitelabel E-commerce* como un nuevo tipo de producto, que, si bien ya existía para el caso de *MercadoShops* y *GetJusto* no tenía una categoría en particular; de manera más local, permite a *Cornershop* abrirse y estar preparado para este nuevo mercado.

Además, se ha conseguido desarrollar una solución, que en términos de potencial, puede satisfacer las necesidades de negocio que motivaron este proyecto, es decir, poder otorgar a las tiendas la posibilidad de montar un comercio virtual propio, con la calidad de servicio que hoy en día otorga *Cornershop*. Entonces, la compañía, siguiendo con la idea del párrafo anterior, posee una solución que le permite ampliar el horizonte de sus clientes.

## 4.3. Logros

Como estudiante, haber tenido la posibilidad de guiar el desarrollo, y por ende, la arquitectura de un proyecto, ha sido una experiencia desafiante y muy enriquecedora. Si bien, *Whitelabel* es un producto que nace como una adaptación de un sistema ya creado (*Cornershop*), este caso, es, en realidad, una de las experiencias que más pueden preparar a un Ingeniero para los proyectos que nacen de compañías grandes, pues, con propósitos de desarrollar siendo eficientes con el tiempo y capital, es sabio aprovechar las soluciones que ya

posea la compañía para satisfacer el objetivo por el cual nace el proyecto; más aún teniendo en cuenta que la primera fase del proyecto, previa a su lanzamiento, era la realización de una prueba de concepto, a partir de la cual, los *stakeholders* decidirían la manera óptima de vender el producto.

*Whitelabel* ha conseguido ser un producto robusto, que cumple con todas las características esperadas de él, las cuales fueron definidas al inicio de este proceso y se encuentran en el apartado de inquietudes 1. Respecto a lo esperado del proyecto, es un logro haber sido capaz de desarrollar una estructura compleja que forma parte de una extensión de otro producto masivo, el cual ya estaba desarrollado por la compañía. Esta estructura es capaz de proveer de todas las características esperadas de una experiencia de un comercio virtual y, como parte del mismo proceso de desarrollo, ha pasado por distintas pruebas de calidad, asistidas por equipos encargados de esto.

Uno de los resultados colaterales del proyecto es el *Whitelabel CORS Middleware*, una herramienta de código poderosa que permite flexibilizar el uso de un protocolo muy usado, a día de hoy, en el mundo de las aplicaciones web. A través de este se permite manejar un listado de orígenes admitidos para *Whitelabel*, de forma eficiente y escalable, pues no dependen, como librerías ya existentes en *Python*, de listados estáticos. En un futuro, esta solución puede ser la base para otras herramientas que deban hacer uso de este protocolo de maneras más flexibles que otras soluciones en el mercado.

Además, apoyado de un equipo de infraestructura y haciendo uso de una herramienta externa, se ha logrado diseñar una estructura que permite a una aplicación, ser accedida desde múltiples orígenes web. Esta es una parte fundamental del proyecto que, en conjunto con el *middleware* y el modelo de datos desarrollado, permiten que la aplicación, a partir del origen del cual provenga el usuario, modifique su *look feel*, es decir, que cambie su apariencia y comportamiento para que el usuario identifique la experiencia con una marca: *branding*

## 4.4. Lecciones Aprendidas

Los proyectos de ingeniería, como en muchas otras áreas, se conforman entre equipos de distintas áreas, en este caso equipos encargados de negocio; de diseño; de QA; de arquitectura de *software*; desarrollo *backend* y desarrollo *frontend*. Es parte fundamental del trabajo de Ingeniería, ser capaz de organizar y hacer efectivo el trabajo entre todas estas áreas. Además, teniendo en cuenta que los proyectos pueden ser parte de un proyecto mayor, que es una empresa, es necesario, en muchas ocasiones, coordinarse y generar soluciones con otros equipos en paralelo.

Dentro de lo aprendido en la universidad, se da mucho énfasis en lo teórico de la programación, sin embargo, retomando lo descrito en la idea anterior, parte importante de cualquier proyecto, se centra en la comunicación con los demás, es decir, en las habilidades blandas de un ingeniero con su equipo e incluso con otros. Si bien, tener conocimiento de distintas áreas de la informática es importante, pues permite generar soluciones eficientes y efectivas dependiendo de la problemática presentada, tener la capacidad de darse cuenta de que no se es experto en ciertas áreas y preguntar opiniones, ideas y conocimientos que puedan tener

otro compañero del área, permite a uno crecer como ingeniero y, más importante aun, como persona.

*Whitelabel* fue un proyecto capaz de generar conocimiento completo sobre desarrollo de *software*, pues en él, se aglomeran partes fundamentales del área: definición de proyectos (problemáticas), diseño de arquitectura, implementación de la misma, pruebas de calidad asistidas por QA, pruebas de concepto y finalmente ser espectador de un conjunto de decisiones de negocio que se originan en una compañía exitosa en el área del desarrollo tecnológico.

Seguir un proceso de desarrollo de arquitectura de *software*, para este caso, el definido en el libro *Software Systems Architecture* [1], es una decisión inteligente que eleva las posibilidades de éxito de un desarrollo de *software*, pues permite, desde un inicio, acordar entre los *stakeholders* un conjunto de cualidades esperadas del proyecto que luego sirven como guía para el diseño de la solución. El proceso sugiere que cada una de las decisiones tomadas en el proyecto estén respaldadas de manera lógica por alguna de estas cualidades, es así que se logra mantener el foco inicial del proyecto y en caso de cubrir todas estas inquietudes, se tiene una solución que, lógicamente, satisface lo esperado del producto.

## 4.5. Futuros pasos del proyecto

El proyecto, a nivel general, se encuentra con su fase MVP completa, es decir, está suficientemente listo para ser lanzado al mercado. Sin embargo, existen ciertas características del mismo que pueden ser mejoradas, y más aún, existen otras funcionalidades que pueden ser agregadas para darle más valor a la idea de un comercio virtual *Whitelabel*.

Tal como se ha mencionado en la sección Desafíos técnicos 2.2. Existen ciertas problemáticas en torno del uso del *grant type Authorization Code Flow* en el protocolo *Oauth2*. Si bien, este tipo de autorización es altamente utilizado a nivel global por distintas empresas, desde hace algunos años se ha sugerido de la idea de aumentar las barreras de seguridad del mismo al utilizar su extensión PKCE, esto, dado que las aplicaciones web carecen de la capacidad de almacenar de manera segura sus credenciales de *Client*. Es así que para mejorar la seguridad del sistema *Whitelabel* se plantea para futuras versiones, el cambio de tipo de concesión utilizada a *Authorization Code Flow with PKCE* [20]. Para ello se tendrá que actualizar, por un lado, la librería utilizada para administrar los servidores *Oauth2* en *Cornershop*, la cual desde la versión 1.3[1] soporta esta extensión; por otro lado, se debe extender el *wrapper* de peticiones utilizado en el *frontend* para soportar los cambios requeridos por PKCE, es decir, la generación y envío de un codigo al azar para que luego el *Authorization Server* pueda identificar al *Client* de forma segura.

Dentro de las nuevas características que se pueden agregar al sistema, se tiene, por un lado, la funcionalidad mencionada en el apartado Estado Actual de la Solución 2.3, el cual es el diseño e implementación de una interfaz CRUD para permitir a las tiendas modificar las características de su experiencia *Whitelabel* de manera más simple e intuitiva, pues, actualmente, este proceso se realiza directamente en la consola provista por *Django*, es decir, es

[1] https://github.com/jazzband/django-oauth-toolkit/blob/master/CHANGELOG.md

asistido por un desarrollador de *Cornershop*. Esta solución agregaría más valor en términos de experiencia de usuario, lo cual era una de las prioridades iniciales esperadas del proyecto [C1]

Por otro lado, basado en una idea similar [C1], se plantea para etapas más avanzadas de *Whitelabel* la posibilidad de personalizar la disposición de la página presentada en *Whitelabel*, es decir, permitir a la tienda elegir cómo están posicionados los elementos de la interfaz de usuario. Esto conlleva muchas decisiones de negocio y es una idea que debe ser pulida, pues hay que definir hasta que punto una tienda puede cambiar la posición de los elementos en la interfaz, de modo que el sitio no quede "inusable". Para realizar esta solución, se estima que será necesario, primero, un proceso de diseño para definir el alcance del reposicionamiento, el diseño de un modelo de datos que soporte e identifique estos cambios y finalmente la implementación de esta disposición personalizable de la experiencia *Whitelabel* a partir de la información almacenada por tienda, de manera similar a como se realiza con el *bootstrap*.

## 4.6. Cierre

*Whitelabel* es un proyecto exitoso, que a pesar de encontrarse en estado de pausa, consigue diseñar e implementar una solución que forma parte de un horizonte nuevo de soluciones, los comercios virtuales sin marca. En los párrafos de este documento, se presentó el diseño e implementación de la arquitectura que da vida *Whitelabel*, en consecuencia de esto que se tiene una solución que consigue cumplir con todas las expectativas iniciales del proyecto, estas pueden resumirse, de manera amplia, en la necesidad de un producto que de la posibilidad a tiendas de montar sus propios comercios virtuales, siguiendo los estándares de calidad de una compañía asentada en este mercado: *Cornershop*.

Previo al final de este documento, vale la pena retomar la pregunta planteada en la sección Marco Teórico 1.2 relacionada con la posibilidad de adaptar un sistema de comercio multi-tienda online a una experiencia *Whitelabel*. Para responder a esta pregunta simplemente basta revisar el proceso de diseño e implementación de *Whitelabel*, una solución que consigue la premisa antes mencionada y por ende, la respuesta a la interrogante es **sí**. Adicionalmente, se plantean dos interrogantes, que forman parte de un nuevo horizonte de investigación que sigue línea de lo planteado en esta memoria:

**¿Existe una manera óptima de adaptar un modelo de negocios multi-tienda online a una experiencia whitelabel? En caso de existir, ¿Cuál sería?**

# Bibliografía

## Libros


[1] Eoin Woods Nick Rozanski. *Software Systems Architecture: Working with Stakeholders Using Viewpoints and Perspectives*. 2nd. Addison-Wesley Professional, 2000.

[22] V. Sarcar. *Java Design Patterns*. Apress, Berkeley, CA, 2022.


## Sitios web


[2] Microsoft. *The OAuth 2.0 Authorization Framework*. 2012. URL: https://datatracker.ietf.org/doc/html/rfc6749#section-1.3 (visitado 15-07-2022).

[3] Drew. Gainor. *Why A White Label Solution Is Easier Than Building Your Own.* 2014. URL: https://www.forbes.com/sites/theyec/2014/06/03/why-a-white-label-solution-is-easier-than-building-your-own/?sh=1d57ea3fdd9e (visitado 12-04-2021).

[4] RedHat. *¿Qué es una arquitectura de aplicaciones?* 2014. URL: https://www.redhat.com/es/topics/cloud-native-apps/what-is-an-application-architecture (visitado 12-04-2021).

[5] Ed. N. Sakimura. *Proof Key for Code Exchange by OAuth Public Clients*. 2015. URL: https://datatracker.ietf.org/doc/html/rfc7636 (visitado 15-07-2022).

[6] Google. *Evolving Chrome's security indicators*. 2018. URL: https://blog.chromium.org/2018/05/evolving-chromes-security-indicators.html (visitado 15-07-2022).

[7] A. Parecki. *OAuth 2.0 for Browser-Based Apps*. 2018. URL: https://datatracker.ietf.org/doc/html/draft-parecki-oauth-browser-based-apps-00 (visitado 15-07-2022).

[8] Alison Vivanco. *Ranking: 10 startups chilenas que revolucionaron el mercado*. 2020. URL: https://laboratorio.latercera.com/laboratorio/noticia/ranking-10-startups-chilenas/1013360/ (visitado 10-04-2021).

[9] Hugo. Delgado. *Advantages and benefits of having a Website on internet.* 2021. URL: https://disenowebakus.net/en/website-benefits (visitado 12-04-2021).

[10] Amazon. *Caching Overview*. 2022. URL: https://aws.amazon.com/caching/ (visitado 15-07-2022).

[11] Amazon. *What is an API?* 2022. URL: https://aws.amazon.com/what-is/api/ (visitado 15-07-2022).

[12] Corecursive. *The Internet Is Made of Duct Tape*. 2022. URL: https://corecursive.com/internet-is-duct-tape/ (visitado 15-07-2022).

[13] Google. *Using OAuth 2.0 to Access Google APIs*. 2022. URL: https://developers.google.com/identity/protocols/oauth2 (visitado 15-07-2022).

[14] Red Hat. *¿Qué es la infraestructura de TI?* 2022. URL: https://www.redhat.com/es/topics/cloud-computing/what-is-it-infrastructure (visitado 15-07-2022).

[15] IBM. *Arquitectura de tres niveles*. 2022. URL: https://www.ibm.com/cl-es/cloud/learn/three-tier-architecture (visitado 15-07-2022).

[16] IBM. *Indexación de base de datos*. 2022. URL: https://www.ibm.com/docs/es/mam/7.6.0.8?topic=databases-database-indexing (visitado 15-07-2022).

[17] Mozilla. *Cross-Origin Resource Sharing*. 2022. URL: https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS (visitado 12-04-2021).

[18] Mozilla. *HTTP cookies*. 2022. URL: https://developer.mozilla.org/es/docs/Web/HTTP/Cookies (visitado 15-07-2022).

[19] Mozilla. *Origin*. 2022. URL: https://developer.mozilla.org/es/docs/Web/HTTP/Headers/Origin (visitado 15-07-2022).

[20] Oauth. *Authorization Code Flow with Proof Key for Code Exchange (PKCE)*. 2022. URL: https://auth0.com/docs/get-started/authentication-and-authorization-flow/authorization-code-flow-with-proof-key-for-code-exchange-pkce (visitado 15-07-2022).

[21] Oauth. *Oauth2.0*. 2022. URL: https://oauth.net/2/ (visitado 15-07-2022).

[23] WSGI. *What is WSGI?* 2022. URL: https://wsgi.readthedocs.io/en/latest/what.html (visitado 15-07-2022).

# Anexo

**Request:**

GET `/stores_whitelabel_api/v0/bootstrap/`

**Response:**


```
{
    "store": {
        "id": 177,
        "uuid": "6b4fe548-258e-4983-8610-68743848d586",
        "name": "Casa&Ideas",
        "img_url": "https://shopping-devel.s3.amazonaws.com/media/store-logo/bc2432
        "light_img_url": "https://shopping-devel.s3.amazonaws.com/media/store-logo/
        "brand_color": "ffb902"
    },
    "whitelabel_config": {
        "uuid": "54b92db4-679a-40d3-85ed-6e425be05df3",
        "host": "local.superpal.com:3000",
        "favicon_url": null,
        "google_analytics_tracking_id": "G-MKZPYGS1Y8",
        "links_of_interest": {
            "facebook": "https://facebook.com",
            "instagram": "https://instagram.com"
        },
        "light_primary_color": "#425563",
        "light_secondary_color": "#ffffff"
    }
}
```


Figura 1: Firma del servicio de *bootstrap*
Fuente: Elaboración propia

| Característica a evaluar | Satisfacción ▲ |
|---|---|
| [C3] El sistema debe ser capaz de mostrar el catálogo de productos de la tienda | 5/5 |
| [C4] El sistema debe permitir realizar compras/órdenes en la tienda específica | 5/5 |
| [C5] El sistema debe asegurar el delivery de la orden al consumidor | 5/5 |
| [C6] El sistema debe poseer un acceso a servicio al clientes | 5/5 |
| [C7] Aplicación debe poder ser accedida desde un dominio web suministrado por el cliente | 5/5 |
| [C8] El producto debe permitir personalización para cada tienda: como mínimo siguiendo la estética de la marca, i.e logotipo y colores. | 5/5 |
| [C9] Whitelabel Order tagging - El sistema debe generar información sobre las órdenes generadas en la plataforma. | 5/5 |

Figura 2: Encuesta de satisfacción por característica
Fuente: Elaboración propia

| Inquietudes iniciales | |
|---|---|
| C1 | Cornershop es una empresa dedicada al mercado de grocery online, permitiendo la compra en múltiples tiendas. Bajo este concepto se diferencia del resto de competidores dado su alto compromiso al servicio de sus usuarios (tiendas y consumidores): buenos tiempos de respuesta, un producto robusto y una gran experiencia del usuario. |
| C2 | Cornershop está perdiendo potenciales clientes (tiendas) que, por temas de branding, prefieren generar su propia experiencia de comercio virtual. |
| C3 | El sistema debe mostrar un catálogo de los productos que ofrece la tienda [C1, C2] |
| C4 | El sistema debe permitir realizar compras/órdenes en la tienda específica [C1, C2] |
| C5 | El sistema debe asegurar el delivery de la orden al consumidor [C1, C2] |
| C6 | El sistema debe poseer un acceso a servicio al cliente [C1, C2] |
| C7 | Aplicación debe poder ser accedido desde un dominio suministrado por los clientes [C2] |
| C8 | El producto debe permitir personalización para cada tienda: como mínimo siguiendo la estética de la marca, i.e logotipo y colores. BRANDING [C2] |
| C9 | El sistema debe generar información sobre las órdenes generadas en la plataforma. [C4, C11] |
| C10 | El desarrollo de software en Cornershop debe estar a la cabeza del standard internacional. |
| C11 | Siendo una empresa dedicada no solo al comercio online, sino que también al desarrollo tecnológico, Cornershop se enfoca mucho en la recolección y uso de datos, por lo tanto, cualquier desarrollo debe aportar información al sistema general de Cornershop. |
| C12 | Software debe ser mantenible, para ello se recomienda seguir las líneas de desarrollo que existen actualmente en la compañía. |
| C13 | Software desarrollado debe ser seguro, para ello se recomienda seguir las líneas de desarrollo actual en la empresa. Además, componentes nuevos deben estar guiados por el estándar internacional en cuanto a seguridad. |
| C14 | Se deben mantener tiempos de respuesta razonables. |
| C15 | *Desarrollo debe seguir la arquitectura denominada "well-sized services" como está definido en el documento de Enginering Guidelines de Cornershop.* |
| C16 | Con respecto a backend, todo el código en la aplicación base está construida en Python/Django. Esta es una de las restricciones principales respecto al desarrollo de código en el sistema backend de Cornershop. |
| C17 | El código desarrollado debe seguir el tipo de desarrollo Test Driven Development, es decir, debe estar altamente enfocado en las pruebas de código. Para este punto y siguiendo los lineamientos de la empresa el código debe cumplir un porcentaje de coverage de tests de al menos un 80 %. |
| C18 | Se debe seguir el estandar PEP8 para escribir código en Python [C10, C16] |
| C19 | Se deben seguir los lineamientos web definidos por el W3C, para este caso en específico, el protoclo definido CORS. [C7, C10] |
| C20 | Se tiene un tiempo límite de 6 meses para probar en un escenario real un MVP del producto. Estas son restricciones iniciales de negocio. |
| C21 | Los costos esperados del desarrollo deben ser bajos, incluso se asignará el proyecto a un equipo pequeño. |
| C22 | Skills requeridos: frontend, backend, diseño e infraestructura. |
| C23 | Se espera que la solución opere funcionalmente en periodos de actividad regular en el continente americano, de igual forma que la aplicación base de Cornershop. |

Tabla 1: Tabla de inquietudes iniciales del proyecto
Fuente: Elaboración propia

| Páginas web relacionadas | |
|---|---|
| ¿Qué es el branding? | https://www.elisava.net/es/noticias/que-es-branding |
| ¿Qué navegadores implementan CORS? | https://caniuse.com/cors |
| Apache Server | https://httpd.apache.org/ |
| Cloudflare | https://www.cloudflare.com/es-es/ |
| Django | https://www.djangoproject.com/ |
| Get Justo | https://www.getjusto.com/ |
| Google | https://google.cl |
| IETF | https://www.ietf.org/ |
| Javascript | https://www.javascript.com/ |
| MercadoShops | https://www.mercadoshops.cl/ |
| Oauth2 | https://oauth.net/2/ |
| React JS | https://reactjs.org/ |
| Rock The Food | https://www.rockthefood.com |
| Sennheiser Store Chile | https://www.sennheiserstore.cl/ |
| Sushi OK | https://www.sushiok.cl/ |
| Github del Django Oauth Toolkit | https://github.com/jazzband/django-oauth-toolkit |

Tabla 2: Sitios web mencionados en el documento
Fuente: Elaboración propia