Escuela Politécnica Superior de Jaén

Grado en Ingeniería Informática

UNIVERSIDAD DE JAÉN

Escuela Politécnica Superior de Jaén

Trabajo Fin de Grado

DESARROLLO DE UNA APLICACIÓN MULTIPLATAFORMA PARA LA GESTIÓN Y SEGUIMIENTO DE STOCK DE UNA EMPRESA

Alumno: Jonnathan García Urrutia Rivera

Tutor: Prof. D. José Ramón Balsas Almagro

Dpto: Informática

Septiembre, 2020

Universidad de Jaén
Escuela Politécnica Superior de Jaén
Departamento de Informática

Don José Ramón Balsas Almagro, tutor del Trabajo Fin de Grado titulado: Aplicación móvil multiplataforma para la gestión y seguimiento de stock en una empresa, que presenta Jonnathan Junior Garcia Urrutia Rivera, autoriza su presentación para defensa y evaluación en la Escuela Politécnica Superior de Jaén.

Jaén, Septiembre de 2020

El alumno: Los tutores:

Jonnathan García Urrutia Rivera José Ramón Balsas Almagro

1. Ilustraciones

2. Tablas

Agradecimientos

Este año ha sido un año distinto para todos, muchos han tenido que reinventarse para poder seguir trabajando en esta pandemia.

Al finalizar este trabajo de fin de grado recordaré este año no solo como el de la pandemia sino como el año que terminé esta bonita etapa la cual me ha ayudado mucho en mi crecimiento personal e intelectual.

La educación me lo ha dado todo y quiero agradecer a mis padres por el gran esfuerzo y sacrificio que hacen día a día, por el apoyo incondicional, por todo, muchas gracias, ni en tres vidas podría pagar todo lo que habéis hecho.

También quiero agradecer a mi novia por apoyarme y estar conmigo en los momentos duros y celebrar los momentos buenos.

Una de las cosas más importantes que me ha aportado la universidad ha sido el poder conocer a grandes personas, muchas de las cuales tengo la suerte de llamar amigo, gracias a todos ellos por hacer de esta etapa más divertida.

Gracias a mi tutor por ayudarme siempre, por su increíble paciencia, por su comprensión y por las lecciones las cuales me llevo y agradezco enormemente.

Por último quiero agradecer a todos los que habéis estado en este camino, desde amigos de mi barrio en Perú, hasta mis mascotas, todos habéis aportado para que hoy pueda escribir estas líneas.

Muchísimas gracias a todos.

Introducción al problema

La gestión de stock es la capacidad de tener controlada la cantidad física de artículos de los cuales disponemos en un momento determinado. Para conseguir tener una buena gestión es importante realizar un inventario de todos los artículos que disponemos en los distintos almacenes.

La gestión de stock es una parte imprescindible para conseguir tener una empresa de éxito. Gestionarlo correctamente o no, nos llevará a resultados favorables como eficiencia, eficacia y en caso contrario puede llevarnos a problemas en otros ámbitos de la empresa como por ejemplo la facturación, como podemos ver en [1].

La gestión de stock está muy ligada a la valoración contable es por ello que han surgido distintos medios para la gestión de stock tales como:

- FIFO (First In First Out) el primer producto en entrar es el primer producto en salir.
- LIFO (Last In First Out) el último en entrar será el primero en salir
- HIFO (Highest In First Out) el producto que se tenga mayor existencia es el primero en salir
- FEFO (First Expired First Out) usado principalmente en la industria alimentaria, el producto más cercano a su caducidad es el primero en salir.

Para tener una visión más global explicaremos qué es el stock y cuales son las ventajas de tener una buena gestión del mismo.

¿Qué es el stock?

El stock es un conjunto de mercancías o productos que se tienen almacenados a la espera de ser distribuidos a los clientes o a un almacén remoto de una misma empresa.

Ventajas de una correcta gestión de stock [2]

1. Ahorro de dinero

Con un buen control de stock disminuimos el gasto de almacenaje, así como el tiempo y el personal destinado a su organización, también nos permite tener una mejor optimización del espacio destinado para los artículos.

2. Eficiencia en el tiempo

Gestionar correctamente el stock (cantidad, ubicación, precios, etc…) permitirá no dedicar tantas horas al proceso de almacén y gestión de pedidos.

Si tenemos controlada la ubicación de cada referencia, no será necesario perder tiempo buscando en qué lugar se dejó el producto “A”.

Otra ventaja es cuando se realizan los inventarios anuales de todos los artículos, si durante todo el año se ha realizado un óptimo control, el proceso se podrá realizar de una forma más rápida.

Los seres humanos somos más productivos y eficientes cuando trabajamos en lugares donde existe un orden y este hecho es más fuerte aún en un almacén.

3. Mejoras en la facturación

Tener una buena gestión nos permitirá disponer de una rápida respuesta ante peticiones de clientes o algún almacén remoto.

Es muy importante conocer cuál debe ser el Stock mínimo por cada artículo de la empresa, así evitaremos tener roturas de stock detectando cuales son los productos que están saliendo con más regularidad, cuáles son los picos fuertes de cada producto y el stock que tiene cada uno en cada momento.

Con toda esta información se podrán generar informes los cuales nos ayudarán a anticiparnos a los siguientes pedidos y tener la capacidad de poder abastecer de forma eficaz y eficiente a nuestros clientes o almacenes.

4. Reducción de tareas administrativas

Aunque una correcta gestión de stock conlleva tiempo de trabajo administrativo, a la larga resulta mucho menos que tener un stock desorganizado.

2.1 Motivación

Este trabajo de fin de grado, de ahora en adelante TFG, consiste en crear una aplicación multiplataforma móvil para la gestión y seguimiento de stock de una empresa.

Para optimizar el proceso de desarrollo multiplataforma se realizará un entorno de desarrollo híbrido basado en HTML5, JavaScript y CSS.

La motivación principal es hacer que las empresas puedan gestionar el stock de una forma rápida y accesible, realizando salidas, entradas y revisando la cantidad de stock de la cual se dispone en un momento determinado en un almacén determinado.

Todo esto se podrá realizar desde cualquier lugar que se disponga de un móvil y conexión a internet.

La segunda motivación la encontré en la actual empresa en la que trabajo, ya que una de las labores que tuve que desempeñar, fue automatizar todo el proceso de gestión del stock. Esto me llevó a darme cuenta de la importancia que tenía esta aplicación no solo a nivel de eficiencia en cuanto a los trabajadores encargados del almacén, si no a nivel administrativo, ya que como sabemos para que una empresa crezca tenemos dos opciones, vender más o gastar menos y con esta aplicación se puede lograr el segundo punto.

2.2 Objetivos

El objetivo principal de este TFG es crear una aplicación multiplataforma que facilite a las empresas la gestión de su stock, haciendo que esta gestión se pueda realizar no solo desde un ordenador si no que también se pueda hacer desde un móvil (ANDROID o IOS) de una forma rápida, eficaz y de una forma simple e intuitiva.

Para conseguir realizar este proyecto se han propuesto los siguientes objetivos

- Implementar una API REST para el backend
- Estudio de entornos de desarrollo multiplataforma basados en tecnologías web
- Estudio de la arquitectura de la aplicación existente y propuesta de modificaciones arquitectónicas para la integración de clientes móviles
- Diseño y desarrollo de los cambios propuestos en la parte del servidor
- Diseño y desarrollo de un cliente móvil multiplataforma con soporte de notificaciones desde el servidor
- Implementación de la aplicación móvil multiplataforma

2.3 Metodología

A lo largo de la carrera hemos visto muchas metodologías para el desarrollo del software. Algunas de ellas son:

- Modelo en cascada
- Modelo de proceso incremental
- Prototipo
- Incremental
- Espiral
- Metodología ágil SCRUM

Después de estudiarlas la que mejor se adapta a mi TFG es el modelo de proceso incremental ya que en esta metodología se combinan los elementos del flujo lineal y paralelo, aplicando secuencias de líneas en forma escalonada conforme avanza el calendario, como podemos ver en [3].

Cuando se utiliza esta metodología es normal que el primer programa que se entregue sea un producto fundamental, que no sea el producto final pero que el cliente pueda interactuar para poder evaluar de forma detallada todas las características de esa entrega.

Los siguientes pasos serán modificar el producto fundamental con las nuevas características y más funcionalidad desarrollando un plan de incremento para que se cubran todas las necesidades del cliente. Este proceso se repite hasta que se consiga tener el producto final que el cliente espera.

Una de las características de esta metodología es que en cada incremente, incluyendo el producto fundamental son productos que son operativos, productos que no tienen todas las funcionalidades del productos final pero que son muy útiles para que el cliente pueda participar en cada incremento y testearlos independientemente.

Algunos de los beneficios de esta metodología son:

- Se pueden gestionar las expectativas del cliente
- El cliente puede comenzar el proyecto con expectativas de alto nivel pero que no sean completas, por lo que en cada incremento tendrá la posibilidad de ir completándolas.
- Se pueden obtener resultados que sean usables desde tempranos incrementos.
- Se pueden perder como máximo los recurso dedicados en un incremento.
- Permite gestionar la complejidad del proyecto.

Como se puede observar en la ilustración 1 cada incremento consta de los mismos pasos, los cuales serían los productos operativos.

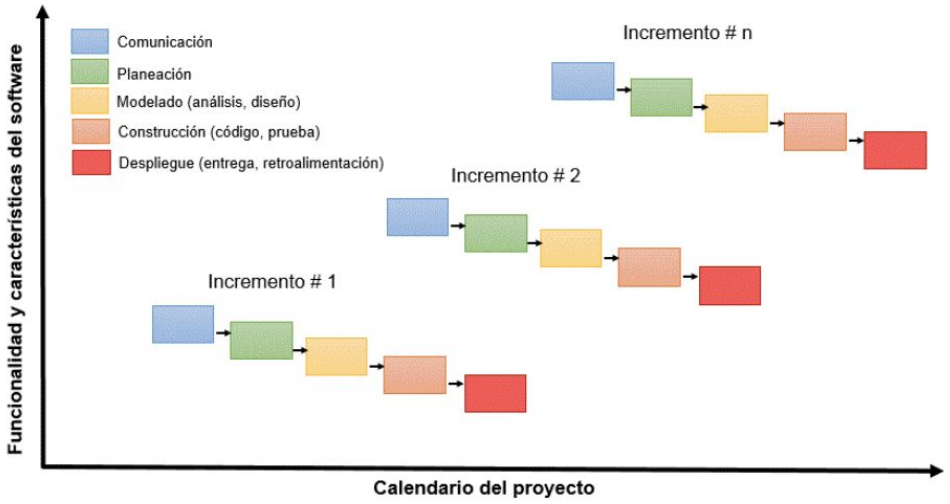


Ilustración 1. Modelo Incremental

2.4 Estructura del documento

La documentación de este proyecto se ha organizado de la siguiente manera:

En el apartado uno de introducción hemos hablado sobre la motivación para el proyecto, cuáles son los objetivos y la metodología utilizada.

En el apartado de análisis se hablará sobre todos los requisitos del proyecto, además se explicarán todas las tecnologías que se usarán y el por que de su elección, comentando cada una de ellas con una breve descripción.

En el apartado de diseño se hablará sobre los distintos diagramas del proyecto.

- Entidad-Relación
- Diagrama de clases
- Diagrama de secuencias
- Diseño de interfaces

En el apartado de implementación se hablará sobre la herramientas que se utilizaron para desarrollar el proyecto, también se comentarán los problemas que se han ido presentando a lo largo de este apartado, además del código de algunas de las partes más importantes, todo esto de una forma más detallada.

En el apartado de pruebas, comprobaremos que tanto los casos de uso como los requisitos funcionales y no funcionales se realizan correctamente.

En los últimos apartados hablaremos sobre las conclusiones del proyecto, posibles mejoras, manual de instalación y manual de usuario.

3. Análisis

En este apartado tenemos que prestar especial atención ya que se trata de uno de los más importantes puesto que resulta imprescindible hacer un buen análisis ya que será la base sobre la cual empezaremos nuestro proyecto.

Hacer un buen análisis de software hace que tengamos una completa comprensión del problema y podamos identificar cuales son las restricciones de la misma, como podemos verlo en [4].

El análisis de requisitos puede dividirse en cinco áreas:

- Reconocimiento del problema
- Evaluación y síntesis
- Modelado
- Especificación
- Revisión

3.1 Análisis preliminar

Este tfg surge a raíz de una aplicación ya existente que desarrolle en mi actual trabajo, en ella se gestiona todos los artículos del almacén central y demás almacenes distribuidos por el país. Esta aplicación aunque cumple con su función y ayuda a automatizar algunos procesos no era muy accesible ya que se desarrolló solo para web y aunque se hizo utilizando un diseño responsive, al mostrar tablas con mucha información no resultaba fácil de usar.

El diseño no era muy amigable y si se añade que la mayoría de usuarios entraban a la aplicación desde un dispositivo móvil la experiencia no era la adecuada. Es por eso que decidí coger esa carencia y realizar una aplicación móvil, pero claro, aquí solucionaba una parte del problema lo cual me llevó a realizar la siguiente pregunta, ¿en qué sistema operativo desarrollo la aplicación móvil, Android o iOS?.

Desarrollando una aplicación móvil multiplataforma utilizando una misma base de código (en este caso tecnología web) podría cubrir gran parte del problema actual y conseguir que la experiencia de usuario sea considerablemente mejor ya que esta aplicación contará con una interfaz amigable y centrada en las características principales que aportan valor al usuario.

3.1.1 Análisis de soluciones similares

Para entenderlo mejor el proyecto haremos una recopilación de algunas de las aplicaciones que realizan una función como las de este TFG.

- **Nubea Stock** [1]

Esta es una aplicación móvil para la gestión del almacén desarrollado en Android.

Se trata de una aplicación que trabaja con módulos que se encuentran en Nubea ERP, un sistema modular desarrollado en la plataforma Open Source Odoo (Open ERP).

Su principal función consiste en lograr una mejora de todas las operaciones internas, consiguiendo optimizar los procesos como entradas y salidas de artículos, llevando así un control de todos los artículos de los que dispone cada almacén.

El punto débil de esta aplicación es que solo se encuentra para dispositivos móviles Android, lo cual reduce su mercado considerablemente.

Ilustración 2. Nubea Stocks

[1] https://diagram.es/nubea-stock/

- **Kyte** [2]

Este es un software más potente que nos permite la administración de inventarios, gestionar la cantidad de stocks, movimientos, realizar ventas.

Para poder usar este software tenemos que estar suscritos a un plan anual o mensual, aunque también tenemos la posibilidad de usarlo con el plan free que cuenta con funciones limitadas.

Las ventajas con respecto a la aplicación anterior es que no solo gestiona el stock, si no que nos permite generar facturas, fidelizar clientes, y crear páginas web gratis.

La desventaja quizás podría verse en que es muy completo para una pequeña empresa la cual no utilizará todas las características de la aplicación.

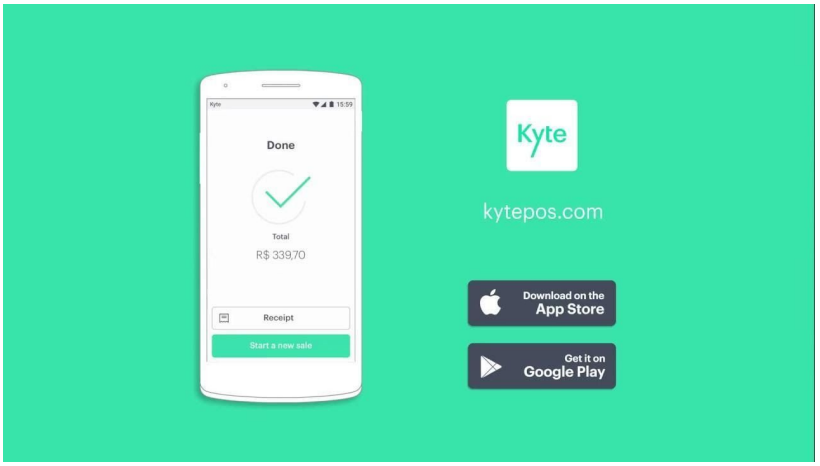


Ilustración 3. Kyte

- **gstock** [3]

Esta es una aplicación en la nube para gestionar las compras de materia prima mediante el control de pedidos, recetas e inventarios, analizando la rentabilidad de las ventas.

[2] https://www.appkyte.com/
[3] https://www.g-stock.es/

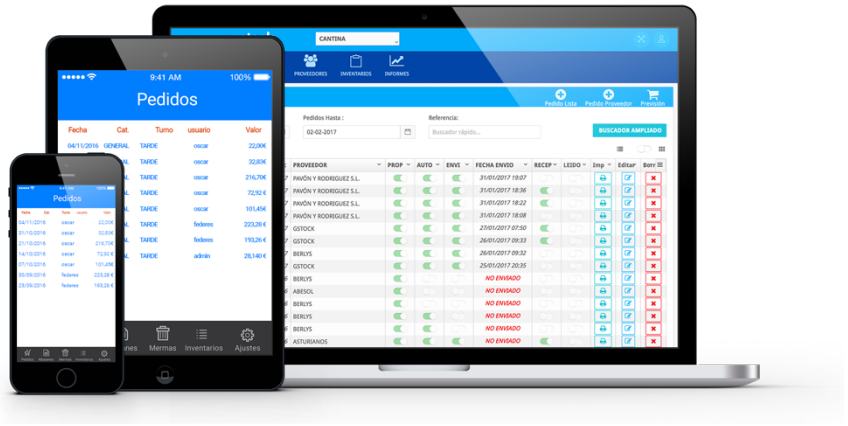


Ilustración 4. gstock

Después de analizar las opciones que actualmente existen la más completa creo que es kyte ya que ofrece una buena plataforma tanto web como móvil además de contar con funciones adicionales a la gestión de stock.

Las soluciones para abordar la gestión del stock son pocas y como el caso de gstock o nubea Stock cuentan con una interfaz anticuada haciendo que la experiencia de usuario no sea la correcta, eso sin comentar que nubea stock y gstock solo están disponibles para Android limitando así la cantidad de usuarios a los que puede llegar.

3.1.2 Análisis de tecnologías

En este punto se hablará sobre las tecnologías existentes para el desarrollo de aplicaciones móviles tanto nativas como híbridas.

Como ya sabemos, el mundo de la tecnología no para de crecer y esto también le afecta al uso de los móviles, según el Consumer Barometer de google [5] y tal como se muestra en la imagen se ha pasado de un 44% de personas que utilizan un smartphone en 2012 a un 87% en el 2017.

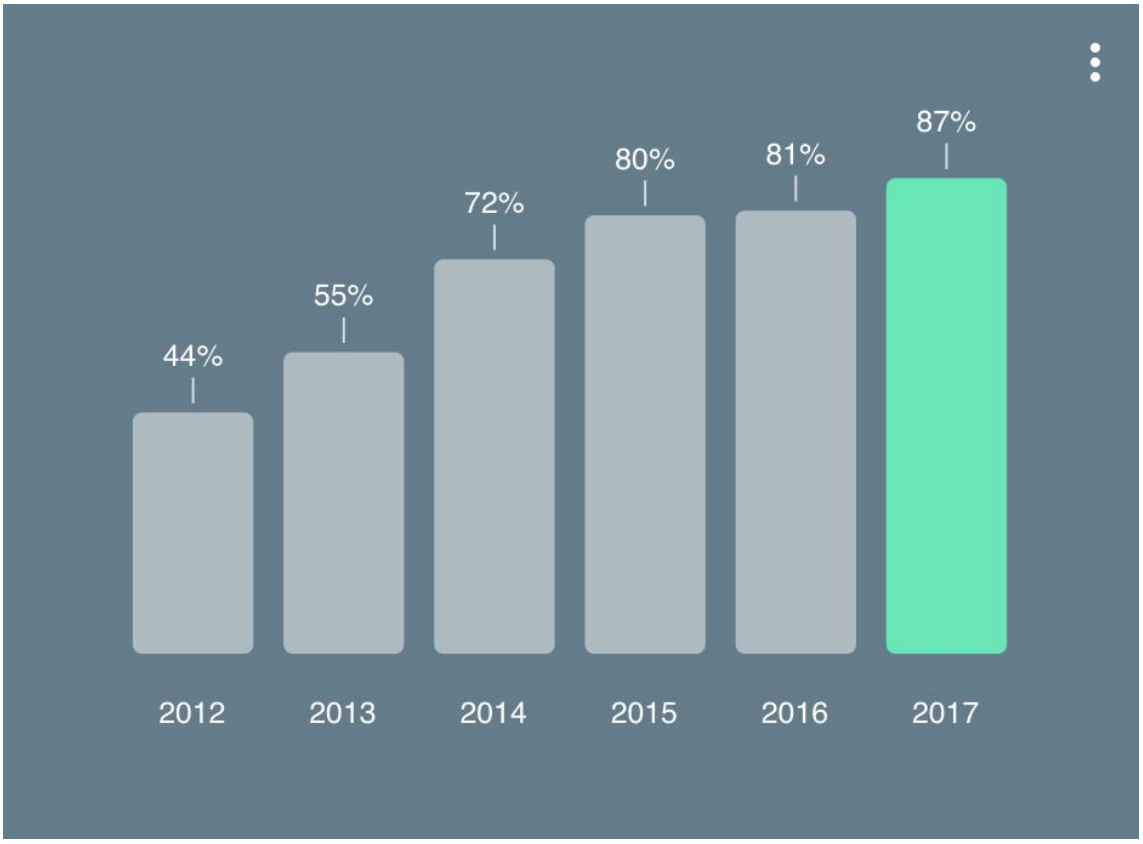


Ilustración 5. Gráfico del uso del smartphone

Según las estadísticas elaboradas por ditrendia [6] a partir de datos de EGM nos dice que el móvil es el dispositivo más utilizado en España para acceder a internet, usado por el 97% de los españoles, tal y como se muestra en el siguiente gráfico.

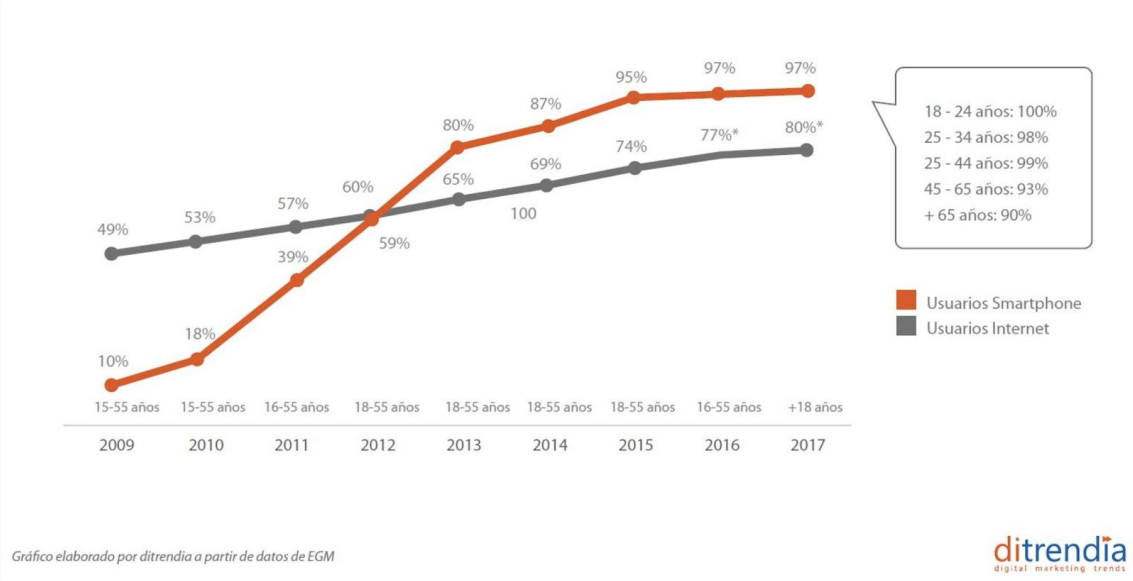


Ilustración 6. Gráfico de dispositivo más usado en España

3.1.2.1 Aplicaciones nativas

Las aplicaciones nativas se denominan así porque el desarrollo de aplicaciones se hace en el lenguaje nativo del propio dispositivo.

Dependiendo de la plataforma en la que queramos desarrollar nuestra aplicación nos encontraremos con distintas tecnologías.

Sistema operativo	Fabricante	Lenguaje	Entorno
Android	Google	Java	Android Studio
iOS	Apple	Objective C, Swift	XCode

Tabla 1. Tecnologías para desarrollar APPs nativas

La principal ventaja del desarrollo nativo lo encontramos en la eficiencia de nuestras aplicaciones ya que aprovechamos todas las ventajas que nos brinda el lenguaje, al estar desarrollada sobre la capa nativa del terminal el rendimiento será el más óptimo posible.

El inconveniente lo encontramos en el tiempo de desarrollo y mantenimiento ya que tenemos que desarrollar dos aplicaciones totalmente diferentes para cada plataforma, lo cual no sería un problema si contamos con los recursos suficientes pero se convierte en una desventaja si no contamos con ellos. En algunos casos esto nos lleva a tener que tomar una decisión y elegir qué plataforma vamos a lanzar primero lo que automáticamente nos limita en el alcance del producto.

Si este fuera el único inconveniente quizás podríamos hacer el esfuerzo y desarrollar las aplicaciones para las distintas plataformas. Pero esto no termina aquí, el software a diferencia de otros campos es un producto vivo el cual tiene que recibir constante mantenimiento y mejoras, por lo que tenemos que tener un equipo para cada plataforma

Al desarrollar una aplicación móvil con tecnologías híbridas este problema no lo tenemos ya que tanto para el desarrollo como para el mantenimiento y mejoras trabajamos sobre la misma base de código.

3.1.2.2 Aplicaciones híbridas

Actualmente el desarrollo de aplicaciones móviles híbridas se ha convertido en una opción muy a tener en cuenta debido a su rapidez en el desarrollo de aplicaciones de distintos sistemas operativos, más aún si la empresa no cuenta con los recursos necesarios para desarrollar dos aplicaciones en lenguajes distintos para sistemas operativos distintos.

Las aplicaciones híbridas utilizan una capa de abstracción sobre la capa nativa, lo cual nos permite encapsular el código nativo de forma que solo realicemos un único código fuente.

La principal ventaja al realizar aplicaciones híbridas es la rapidez y el ahorro de recursos al tener solo que mantener una base de código.

La desventaja la podemos encontrar en la fluidez en que la que comporta una aplicación híbrida al no ejecutarse sobre la capa nativa.

Estos sistemas se encargan de encapsular la aplicación con un webkit[4] de la plataforma nativa, la cual nos permite poder publicarla en las respectivas tiendas (Google Play Store de Android, App Store de iOS), además de poder conectarnos a las API's nativas ofrecidas por cada entorno, ofreciéndonos así la posibilidad de enviar notificaciones push, conectarnos al GPS, cámara, sensores, etc.

En la ilustración 7 se puede resumir las diferencias entre las tecnologías.

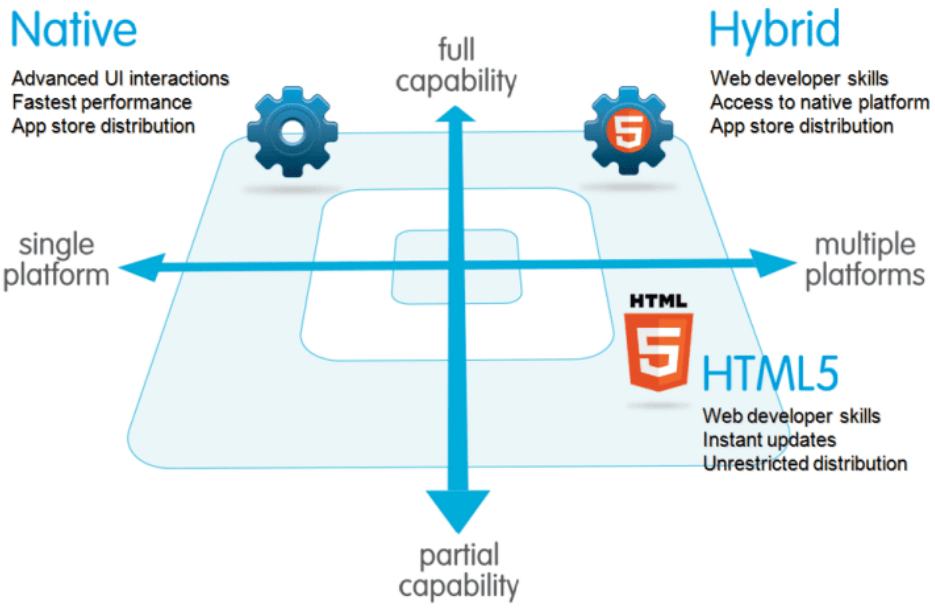


Ilustración 7. Diferencias entre aplicaciones nativa vs híbridas

Para desarrollar estas aplicaciones podemos usar las siguientes tecnologías[5].

- Ionic[6]
- Apache Cordova[7]
- React Native[8]

[4] motor de navegación web de código libre utilizado para construir aplicaciones como navegadores, clientes de correo electrónico, lectores RSS, etc)
[5] Frameworks para desarrollo de aplicaciones móviles híbridas
[6] Ionic - Cross-Platform Mobile App Development
[7] Apache Cordova
[8] React Native · A framework for building native apps using React

- Flutter[9]
- Framework 7[10]
- NativeScript[11]
- Jquery Mobile[12]

Para realizar este TFG se ha utilizado Ionic. La elección de este framework se debe a que cuando se desarrolla una aplicación con IONIC lo que verdaderamente se está haciendo es apostar por tecnologías web, tecnologías consolidadas tales como TypeScript, Angular, Cordova, además de HTML, CSS y JavaScript. La documentación es otro punto muy a tener en cuenta y tiene un buen ecosistema y una comunidad activa que lo respalde.

En el siguiente apartado se hablará más sobre esta tecnología.

3.1.2.3 Desarrollo back-end

El back-end es la parte del desarrollo web que se encarga de toda la lógica de negocio que tendrá nuestra aplicación entre otra cosas, el back-end es transparente al usuario pero muy importante encargándose de acciones como conexión a la base de datos, intercambio de información entre las aplicaciones y el servidor, etc.. Para este intercambio de información se utilizará la arquitectura REST de la cual hablaremos con más detalle en los próximos apartados.

Existen muchas tecnologías con las que se puede desarrollar el back-end, p.e. Nodejs, Java, Python, PHP, .NET, etc., todas de ellas factibles para realizar este TFG pero al final opte por utilizar PHP ayudado de un framework llamada Zend Framework, me he decantado por Zend debido a que es un framework que conozco, tiene una trayectoria probada, soporte y estoy muy familiarizado con él.

3.2 Propuesta de solución

El proyecto que se pretende desarrollar es una aplicación cliente - servidor en Internet con el objetivo de controlar la gestión de stock en una empresa y facilitar el acceso de los usuarios a dicha información. Para ello propongo crear una aplicación móvil multiplataforma utilizando las siguientes tecnologías.

Para desarrollar la aplicación móvil multiplataforma se utilizará un framework de desarrollo híbrido como es Ionic y adaptar el back-end existente para incorporar una capa REST la cual utilizaremos para servir los recursos al cliente, el backend se realizará utilizando Zend Framework, la persistencia de datos la tendremos con MySql y por último se utilizará Firebase para el envío de notificaciones PUSH.

[9] Flutter - Crea hermosas aplicaciones nativas en tiempo récord
[10] Framework7 - Full Featured Framework For Building iOS, Android & Desktop Apps
[11] NativeScript: Native mobile apps with Angular, Vue.js, TypeScript, JavaScript
[12] jQuery Mobile

Para llevar a cabo este proyecto se hará uso de las siguientes tecnologías.

- **Ionic v4** [7]

Este SDK se usará para el desarrollo de la interfaz. Ionic Framework es un kit de herramientas de interfaz de usuarios de código abierto para crear aplicación híbridas y de escritorios con tecnología web (HTLM, CSS y JavaScript).

Este framework se centra en la experiencia del usuario haciendo que las interacciones de la interfaz, controles, gestos y animaciones sean de la forma más natural posible.

Ilustración 8. Ionic

Actualmente Ionic Framework tiene integración con Angular que es uno de los framework más populares por los desarrolladores frontend.

- **Angular** [13]

Angular es un framework para aplicaciones web desarrollado en TypeScript y mantenido por Google.

En un framework que utiliza el Modelo Vista Controlador (MVC) haciendo que el desarrollo y las pruebas sean más fáciles.

[13] Angular.io

Ilustración 9. Angular

- **Zend Framework**[14]

Zend Framework (ZF) es un framework de código abierto para desarrollar aplicaciones web en PHP el cual cuenta con una implementación orientada a objetos.

La estructura de componentes de ZF está construida con una baja dependencia de otros componentes lo cual permite a los desarrolladores utilizar componentes por separado.

Con este framework se ha implementado el back-end utilizando una arquitectura REST.

Ilustración 10. Zend Framework

[14] Zend Framework: Home

- **Firebase**[15]

Firebase es una plataforma para el desarrollo de aplicaciones Web y móviles desarrollada por James tamplin y Andrew Lee en 2012 y adquirida por Google en 2014.

Firebase nos proporciona una serie de servicios que nos ayudan a desarrollar aplicaciones de calidad, algunas de estos servicios son :

- Firebase Cloud Messaging
- Firebase Auth
- Realtime Database
- Firebase Storage
- Firebase Firestore

Para este TFG se ha utilizado el servicio Firebase Cloud Messaging [16] para el envío de notificaciones PUSH.

Firebase Cloud Messaging (FCM) proporciona una conexión confiable entre nuestro servidor y los dispositivos de nuestros clientes, lo cual permite enviar y recibir mensajes y notificaciones en las distintas aplicaciones, Android, iOS y Web sin ningún costo.

3.3 Requisitos del sistema

En este apartado se hablará sobre los requisitos que la aplicación debe cumplir, tanto a nivel de requisitos funcionales como no funcionales.

3.3.1 Requisitos funcionales

Los requisitos funcionales que encontraremos en este TFG son los siguientes :

- Posibilidad de acceder a la aplicación con las credenciales de acceso (usuario y contraseña)
- Posibilidad de ver listar todos los almacenes que tenemos asignados
- Posibilidad de listar los artículos con la cantidad de stock de cada uno de ellos por almacén
- Generar salida de artículos de un almacén a otro
- Confirmar entrar de artículos

[15] Firebase
[16] Firebase Cloud Messaging (FCM)

- Ver los detalles de cada salida y entrada
- Filtrar las salidas y entradas por rango de fecha
- Generar notificaciones PUSH[17] para un mejor feedback

3.3.2 Requisitos NO funcionales

Los requisitos no funcionales especifican criterios del sistema que están relacionados con el grado de cumplimiento de los requisitos funcionales.

Los requisitos no funcionales de este TFG se separarán en distintas áreas, estas son:

- **EFICIENCIA**
  - Toda funcionalidad del sistema debe responder al usuario
  - Los datos modificados en la base de datos deben de ser actualizados para todos los usuarios.

- **SEGURIDAD LÓGICA Y DE DATOS**
  - El sistema debe de ser desarrollado aplicando patrones que incrementen la seguridad de los datos. p.e.
    - Validar todas las entradas de datos
  - Tener un respaldo incremental de la base de datos y un control de versiones para el código
  - Todas las comunicaciones tienen que viajar por un canal seguro

- **USABILIDAD**
  - El tiempo de aprendizaje por parte de usuario para utilizar la aplicación tiene que ser el mínimo
  - El sistema tiene que contar con manuales en los que los usuarios se puedan apoyar ante cualquier duda

[17] https://ionicframework.com/docs/native/push

- El sistema tiene que dar feedback al usuario (mediantes mensajes de alerta que sean informativos) antes los distintos eventos, más aún cuando se produce algún error

- Al tratarse de una aplicación híbrida y multiplataforma, la interfaz tiene que adaptarse adecuadamente al tamaño del dispositivo donde se ejecute, a esta característica de adaptación se le conoce como responsive. Hoy en día accedemos a los sitios web desde distintos dispositivos por lo que es cada vez más necesario, incluso diría obligatorio que nuestras aplicaciones se puedan adaptar a los distintos tamaños de los dispositivos. Esto se consigue gracias a las propiedades de CSS3 usando las media-queries, las cuales se caracterizan por organizar los layouts e imágenes de forma fluida según las dimensiones de las pantallas. En definitiva cada vez se le da más importancia a que la aplicación se pueda consumir desde distintos dispositivos y más aún desde un móvil ya que pasamos mucho de nuestro tiempo consumiendo información desde el, tanto es así que una de las buenas prácticas cuando se está desarrollando una aplicación es hacerlo usando el modo de desarrollo mobile first, el cual nos asegura que los usuario tendrán la misma experiencia al navegar por la aplicación que desde un ordenador, móvil o tablet.

3.4 Planificación inicial de tareas

Para empezar se detalla cada una de las actividades para lo cual nos apoyaremos en la ilustración 11, mostrando el tiempo dedicado en días y horas de cada una de ellas.

Como podemos ver las tareas se han dividido en tres partes, análisis inicial, desarrollo y documentación. En la primera parte analizaremos la interfaz, el desarrollo de la API REST para el cliente móvil, la base de datos y el estudio de las tecnologías para su posterior elección.

En la segunda parte prepararemos el entorno del servidor, la API REST, interfaz con ionic y notificaciones PUSH con firebase. Como podemos ver en el desarrollo cada una de estas parte se dividen en tres grupos, análisis, diseño e

implementación, con ello conseguimos una mejor organización en el listado de las tareas.

	Nombre de la tarea	Fecha de inicio	Fecha final	Duración (Día)	Duración (Horas)
		2019/07/15 19:00	2019/12/02 22:00	101d	
1	**Análisis**	2019/07/15 19:00	2019/08/05 22:00	16d	48
1.1	Análisis de la interfaz	2019/07/15 19:00	2019/07/15 22:00	1d	3
1.2	API REST	2019/07/16 19:00	2019/07/25 22:00	8d	24
1.3	Base de datos	2019/07/26 19:00	2019/08/02 22:00	6d	18
1.4	Elección de tecnologias	2019/08/05 19:00	2019/08/05 22:00	1d	3
2	**Desarrollo**	2019/08/06 19:00	2019/09/20 22:00	34d	102
2.1	**Entorno servidor**	2019/08/06 19:00	2019/08/12 22:00	5d	15
2.1.1	Análisis	2019/08/06 19:00	2019/08/06 22:00	1d	3
2.1.2	Diseño	2019/08/07 19:00	2019/08/07 22:00	1d	3
2.1.3	Implementacion	2019/08/08 19:00	2019/08/12 22:00	3d	9
2.2	**API REST**	2019/08/13 19:00	2019/08/26 22:00	10d	30
2.2.1	Análisis	2019/08/13 19:00	2019/08/15 22:00	3d	9
2.2.2	Diseño	2019/08/16 19:00	2019/08/19 22:00	2d	6
2.2.3	Implementación	2019/08/20 19:00	2019/08/26 22:00	5d	15
2.3	**Interfaz gráfica**	2019/08/27 19:00	2019/09/16 22:00	15d	45
2.3.1	Análisis	2019/08/27 19:00	2019/08/29 22:00	3d	9
2.3.2	Diseño	2019/08/30 19:00	2019/09/05 22:00	5d	15
2.3.3	Implementación	2019/09/06 19:00	2019/09/16 22:00	7d	21
2.4	**Notificaciones PUSH (Fireba...**	2019/09/17 19:00	2019/09/20 22:00	4d	12
2.4.1	Análisis	2019/09/17 19:00	2019/09/17 22:00	1d	3
2.4.2	Diseño	2019/09/18 19:00	2019/09/18 22:00	1d	3
2.4.3	Implementación	2019/09/19 19:00	2019/09/20 22:00	2d	6
3	**Documentación**	2019/09/23 19:00	2019/12/02 22:00	51d	153
3.1	Redactar documentación	2019/09/23 19:00	2019/12/02 22:00	51d	153

Ilustración 11. Planificación de las tareas

Para exponer el tiempo previsto que se dedicará para las diferentes tareas utilizaremos el siguiente diagrama de Gantt. Para una mejor visualización el diagrama se dividirá en tres partes.

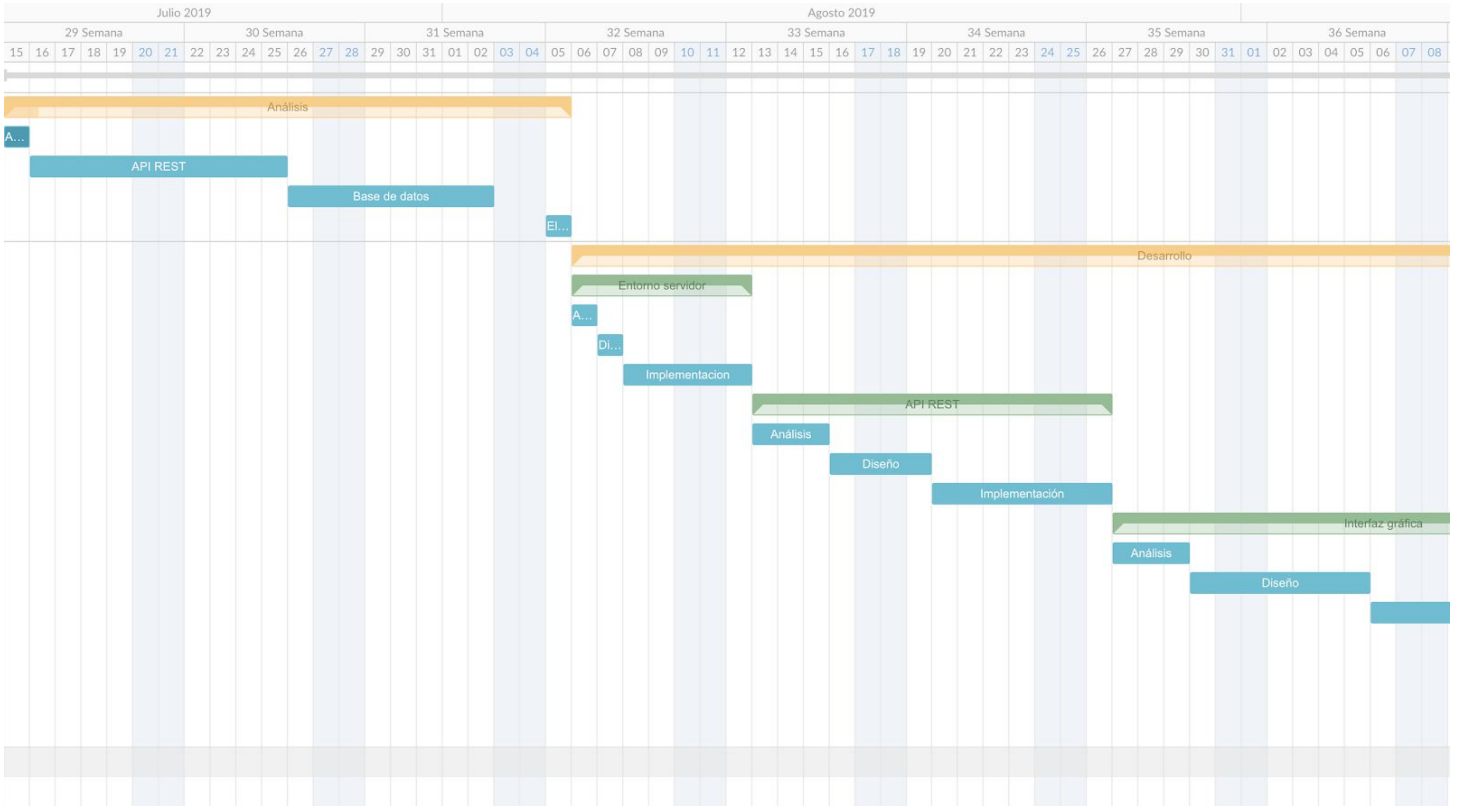


Ilustración 12. Diagrama de gantt

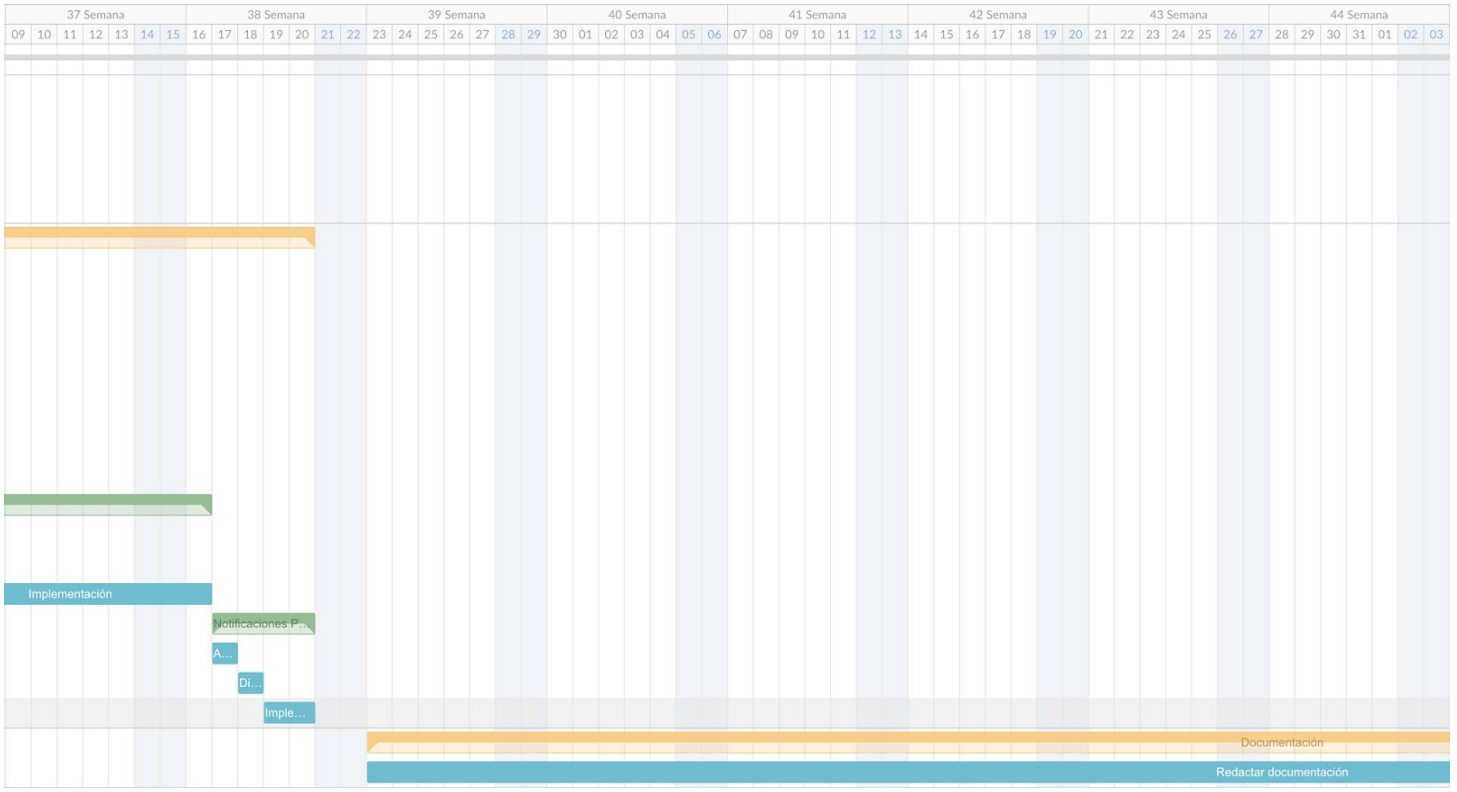


Ilustración 12. Diagrama de gantt

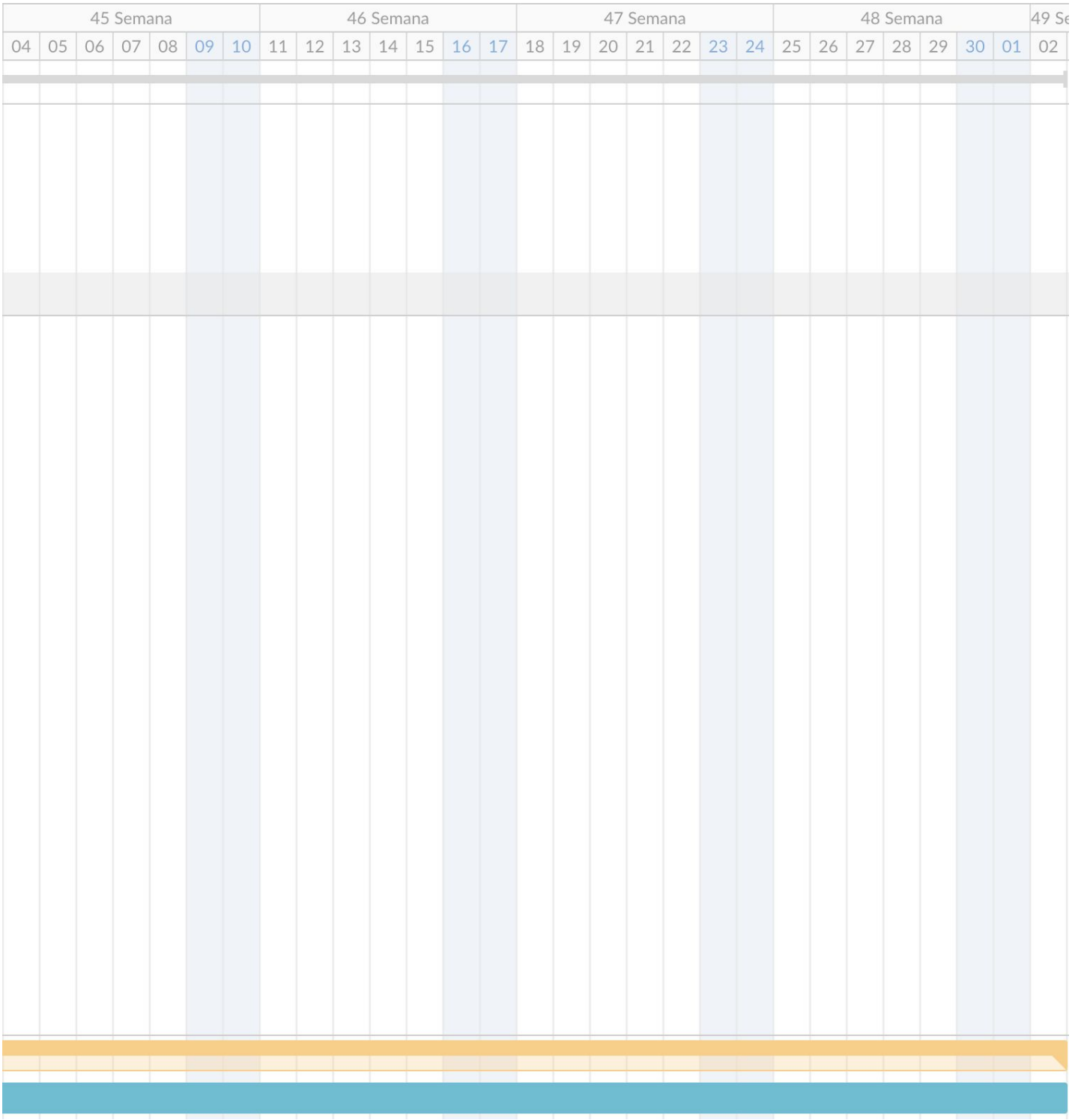


Ilustración 12. Diagrama de gantt

Ahora estimaremos el total de horas que nos tomará realizar este proyecto.

Este proyecto comenzó en julio de 2019 y se terminará aproximadamente en diciembre del mismo año, por lo que el desarrollo durará 5 meses en los cuales se trabajará 3 horas diarias aproximadamente.

Tareas	Duración (Horas)
Análisis y planificación	15
Análisis de la interfaz	12
Preparar entorno servidor	34
Análisis API REST	39
Análisis de base de datos	30
Desarrollo de endpoint	60
Interfaz gráfica	49
Gestión de notificaciones PUSH	24
Documentación	40

Tabla 2. Estimación de horas para realizar el proyecto

Cabe mencionar que estas fechas son de estimación y podrán cambiar, por lo que en el siguiente subapartado se hará una comparativa comentando las diferencias que hubo entre la planificación original y el resultado final.

3.4.1 Diferencias entre planificación original y resultado final

La principal diferencia entre la planificación original y el resultado final es debido a tiempos no previstos en la realización de la documentación. Documentar un proyecto de inicio a fin fue más complicado de lo que me esperaba, pero principalmente la razón por la cual no se cumplió la planificación inicial fue porque durante el proceso de desarrollo de este TFG hubo muchas interrupciones debidas a la mala organización que realicé de mi tiempo compaginando el trabajo con los estudios.

3.5 Estudio de viabilidad

En este apartado se realizará una estimación de los costes necesarios para realizar este proyecto. Para afinar estos costes lo máximo posible se

dividirán en tres apartados, costes software, costes hardware y costes del personal o mano de obra, la suma de todos estos nos dará el coste total.

3.5.1 Costes software

Los costes asociados al software en este proyecto son los siguientes:

Software	**Coste**
Ionic	0€
Zend Framework	0€
Base de datos	0€
Firebase	0€
Suite Online Visual Paradigm	0€
Planificación	0€
Visual Studio Code	0€
Atom	0€
GitHub	0€
TOTAL	**0€**

Tabla 3. Costes software proyecto

Como podemos ver el coste del software es nulo. Esto se debe a que se ha utilizado herramientas de software libre gracias a las cuales el importe de este apartado es **0€.**

3.5.2 Costes hardware

Los costes hardware de este proyecto los encontramos en el ordenador iMac de 27 pulgadas, procesador de seis núcleos a 3.1 Ghz, 16 GB de memoria y 1 TB de almacenamiento, ya que este ordenador no se adquiere exclusivamente para este TFG tenemos que calcular la parte proporcional.

Dado que la media de vida útil del ordenador es de 6 años y el coste del equipo es 2299€ nos da un coste de 1,05 € por día.

También se ha utilizado un móvil iphone XS de 256 GB para realizar las distintas pruebas, el móvil al igual que el ordenador no fueron comprados exclusivamente para el proyecto por lo que tendremos que calcular la parte

proporcional. Teniendo en cuenta que el precio fue de 830€ y la vida útil suele ser de 3 años nos da un precio de 0,76€ por día.

Para poder desplegar la aplicación hace falta tener un servidor y aunque esto lo podría hacer desde mi ordenador personal y posteriormente abrir puertos del router para tener acceso a este desde internet. Para evitar las complicaciones que esta configuración pueda tener decidí contratar un servicio que me proporcionará un VPS en el cual desplegar la aplicación, este servicio se ha contratado con el proveedor OVH por 10,85€ tres meses.

Por último tenemos que tener en cuenta el coste proporcional de la conexión a Internet el cual tiene un coste de 50€ al mes, 1,66€ al día.

Para calcular el coste de cada uno de estos puntos tenemos que tener en cuenta que todo el proyecto se estima que se realizará en 101 días.

Hardware	**Coste**
iMac	1,05€ x101 (días totales) = 106,05€
Móvil	0,76€ x 101 (días totales) = 76,76€
Conexión a Internet	1,66€ x 101 (días totales) = 167,66€
OVH - Proveedor de alojamiento web	10,85€ (3 meses)
TOTAL	**361,32€**

Tabla 4. Costes hardware

3.5.3 Costes personal

Para calcular el coste total del personal se dividirá entre el sueldo del analista y el sueldo del desarrollador.

El sueldo de un analista según el convenio de ingenieria publicado en el BOE-A-2019-14977 [8] es de 26065,07€ dividido entre 12 meses nos da 2172,09€ y por hora sería 13,5€, en este proyecto el analista trabajará aproximadamente 58 horas (las cuales se estimaron en la tabla de tareas y podemos ver en la columna Duración (Horas) de la misma) lo cual da un coste de 783€.

Por otro lado el sueldo de un desarrollador es de 20077,13€ dividido entre 12 meses nos da 1673,09€ y por hora nos da 9,65€, en este proyecto el desarrollador trabajará aproximadamente durante 245 horas (215 de desarrollo

y 30 de documentación, tal y como podemos ver en la tabla de tareas) haciendo un total de 2364,25€.

Sumando el coste del analista más el desarrollador nos da un total de 3147,25€.

El coste final de este proyecto sería la suma de cada apartado de costes, quedando de la siguiente manera.

Coste software	0€
Coste hardware	361,32€
COSTE PERSONAL	
Coste analista (58 horas)	13,5€ * 58 (horas) = 783€
COSTE DESARROLLO	
Coste desarrollador (245 horas)	9,65€ * 245 (horas) = 2364,25€
TOTAL	**3508,57€**

Tabla 5. Resumen costes totales del proyecto

3.6 Modelo de dominio

En este apartado se hablará sobre los aspectos más importantes de las entidades y relaciones que componen este proyecto, para ello nos apoyaremos sobre el diagrama que se muestra en la ilustración 13.

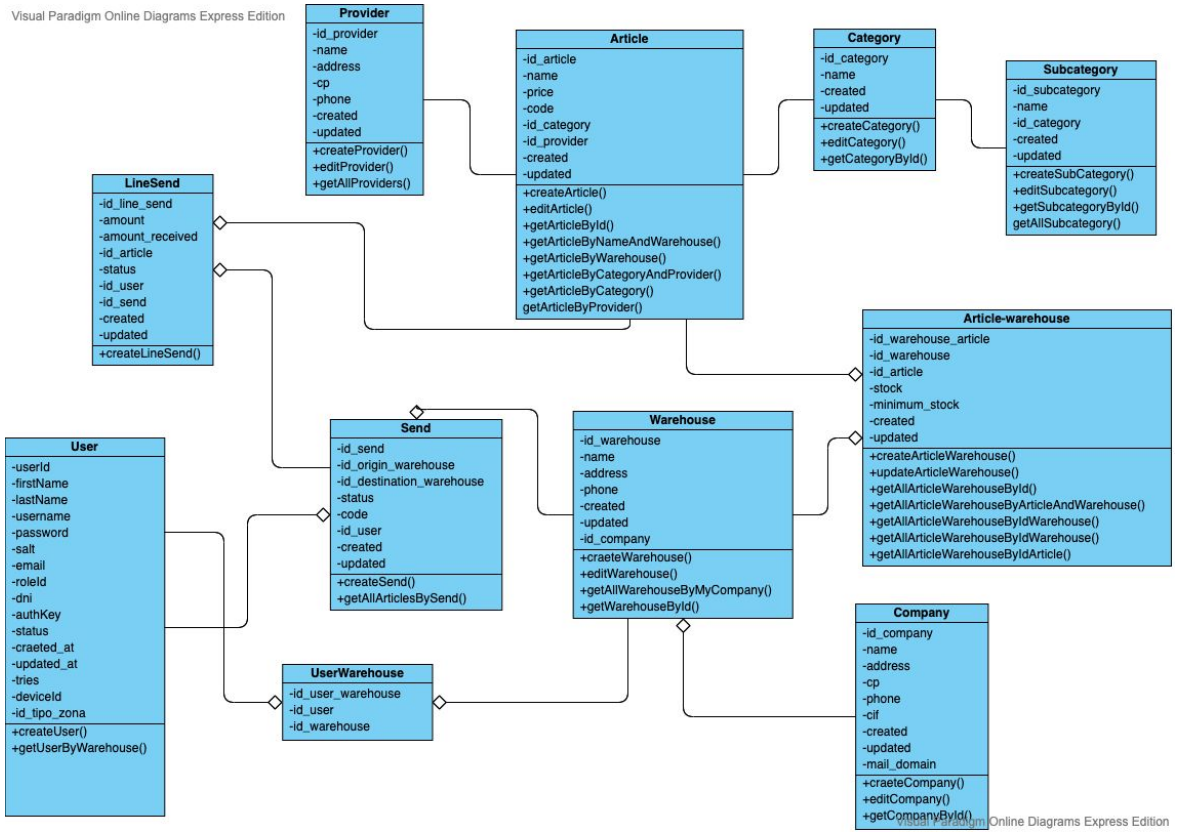


Ilustración 13. Diagrama UML

Las clases principales del modelo de dominio son Article, Article-warehouse y Warehouse, a continuación se describe los métodos de cada una de estas clases.

Article

- createArticle : Encargado de crear un artículo.
- editArticle : Encargado de editar un artículo.
- getArticleByNameAndWarehouse : Encargado de obtener los artículos filtrando por nombre y almacén.
- getArticleByWarehouse : Encargado de obtener todos los artículos filtrando por almacén.
- getArticleByCategoryAndProvider : Encargado de obtener todos los artículos filtrando por categoría y proveedor.

- getArticleByCategory : Al igual que el método anterior pero devuelve solo los artículos filtrando por categoría.
- getArticleByProvider : Encargado de obtener los artículos filtrando por proveedor.

Article-warehouse

- createArticleWarehouse : Encargado de asociar un artículo a un almacén.
- updateArticleWarehouse : Edita un artículo que se encuentra en un almacén.
- getAllArticleWarehouseById : Obtiene la instancia de articleWarehouse filtrando por el identificador.
- getAllArticleWarehouseByArticleAndWarehouse : Obtiene los artículos que pertenecen a un almacén filtrando por artículo y almacén.
- getAllArticleWarehouseByIdWarehouse : Obtiene todos los artículos del almacén filtrando por el identificador de un almacén.
- getAllArticleWarehouseByIdArticle : Obtiene todos los artículos del almacén filtrando por artículo.

Warehouse

- createWarehouse : Crea un almacén.
- editWarehouse : Edita un almacén.
- getAllWarehouseByMyCompany : Obtiene todos los almacenes filtrando por empresa.
- getWarehouseById : Obtiene el almacén filtrando por almacén.

3.7 Casos de uso

A continuación se muestra el diagrama de casos de uso:

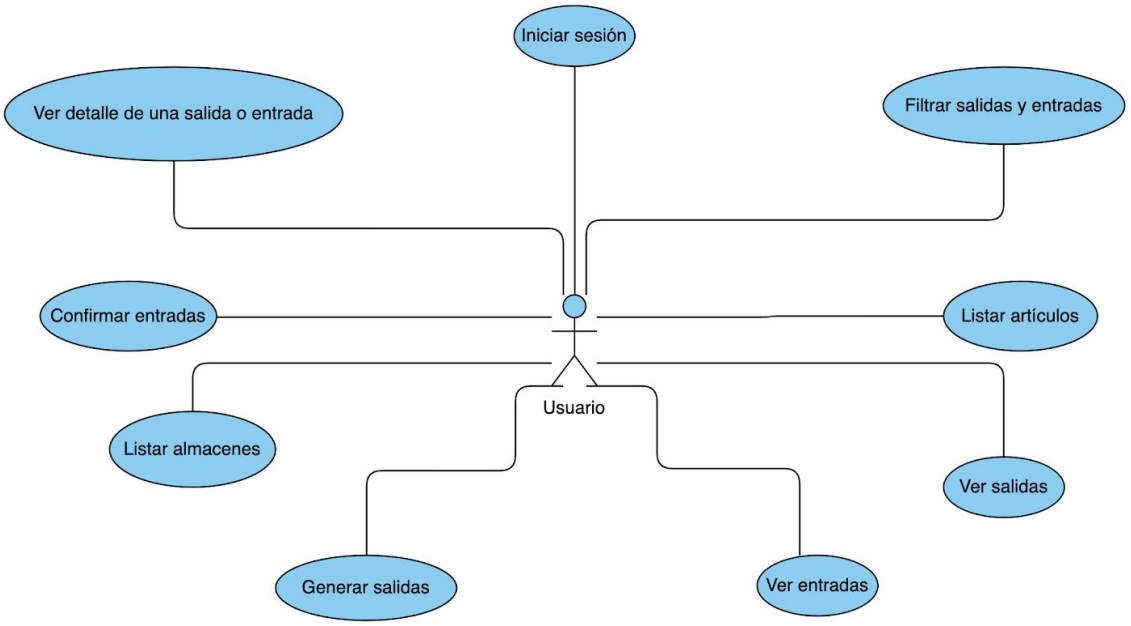


Ilustración 14. Diagrama de casos de uso

La aplicación permite realizar las siguientes acciones:

- Iniciar sesión
- Filtrar salidas y entradas
- Listar artículos
- Ver salidas
- Ver entradas
- Generar salidas
- Listar almacenes
- Confirmar entradas
- Ver detalle de una salida o entrada

4. Diseño

En este apartado se hablará sobre el diseño de la interfaz, el plan de pruebas y los distintos diagramas que se han utilizado a lo largo del TFG. Los diagramas de los cuales se hablará son los siguientes :

- Diagrama entidad - relación
- Diagrama de clases
- Diagrama de secuencias

4.1 Diagrama entidad - relación

El modelo entidad relación es una herramienta que facilita el diseño de las relaciones en una base de datos. Este proyecto cuenta con distintas entidades y relaciones como vemos en la ilustración 15, las cuales se explicarán a fondo para una mayor comprensión del por qué de cada entidad.

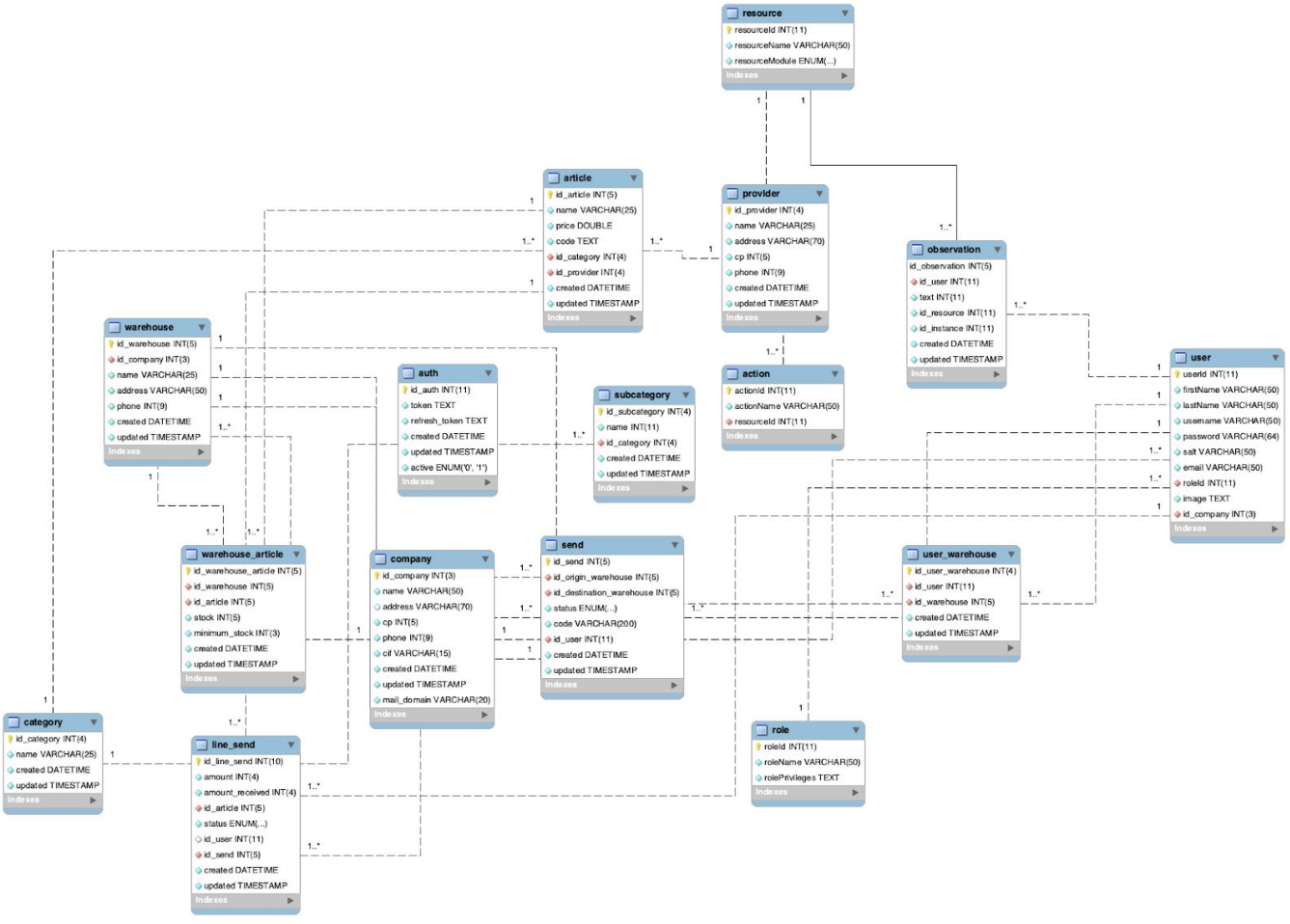


Ilustración 15. Diagrama ER

Empezaremos por enumerar las distintas entidades.

1. action
2. article
3. auth
4. category
5. company
6. line_send
7. observation
8. provider
9. resource
10. role
11. send
12. subcategory
13. user
14. user_warehouse
15. warehouse
16. warehouse_article

Para tener una mejor perspectiva acerca del diagrama se comentarán brevemente cada una de las tablas y las relaciones entre ellas.

- **article**
  - tabla para guardar todos los artículos.

- **warehouse**
  - tabla para almacenar los almacenes.

- **warehouse_article**
  - tabla que relaciona los artículos que pertenecen a un almacén.

- **company**
  - table que guarda las empresas.

- **user**
  - tabla que almacena los usuarios. Esta tabla es muy importante de cara a la aplicación móvil ya que en ella guardaremos el token que nos proporciona Firebase y que posteriormente utilizaremos para enviar las notificaciones push.

- **send**
  - tabla que almacena todos los envíos que se realizan, cada envío puede tener más de un artículo por lo que también se necesitará almacenar esta información.

- **line_send**
  - tabla para almacenar cada línea del envío, cada línea cuenta con el artículo en cuestión, la cantidad, estado (‘enviado’, ‘recibido’, ‘recibido parcialmente’).

- **user_warehouse**
  - tabla que relaciona los usuarios con los almacenes a los que tiene acceso. Con esta tabla podemos saber que almacenes mostrar a cada uno de los usuarios que hayan iniciado sesión en la aplicación móvil.

- **category**
  - tabla para guardar las categorías de un artículo. Esta tabla no interacciona con el API REST, pero será útil para las mejoras.

- **subcategory**
  - tabla que guarda las subcategorías. Al igual que la tabla anterior, esta no interacciona con el API REST.

- **provider**
  - tabla que guarda los proveedores de los distintos artículos. Tampoco interacciona con el API REST, pero será útil para futuras mejoras.

Las tablas que más interacción tienen con la aplicación son user, *user_warehouse*, *warehouse_article*, *send* y *line_send*. Existen una serie de restricciones las cuales se explicarán tabla por tabla.

La tabla *user* es donde se almacenan todos los usuarios del sistema, esta tabla se relaciona con la tabla company y *role*, por lo que un usuario solo puede pertenecer a una empresa y tener un rol. Sin embargo puede pertenecer a más de un almacén y es por eso que existe la tabla *user_warehouse*, la cual almacena la relación entre la tabla user y la tabla warehouse. Esta tabla es muy importante ya que en ella almacenamos el token que nos proporciona Firebase para poder enviar las notificaciones PUSH. Este token es único por cliente y tenemos que actualizarlo cada vez que el cliente inicie sesión en la app móvil, ya que la generación de este no depende de nosotros si no de Firebase y si ven oportuno que cambie podrá cambiar y nosotros tendremos que actulizarlo en esta tabla.

Como cada almacén tiene distintos artículos se creó la tabla *warehouse_article* la cual relaciona las tablas *warehouse* y *article*. En esta tabla almacenamos el stock del cual dispone cada almacén, el stock mínimo que debe tener (este atributo es útil para realizar avisos a los distintos encargados de departamentos cuando se llegue al stock mínimo de un artículo) además de la fecha de creación y la fecha de actualización de cada instancia.

Aunque con estas entidades tenemos organizado el stock de cada uno de nuestros almacenes podemos darle acceso a cada usuario su almacén correspondiente, no es suficiente si queremos llevar un buen control de los artículos, es por eso se que crearon las tablas *send* y *line_send*, en ellas se almacenarán todos los envíos que se realicen desde un almacén origen hasta un almacén destino.

Ya que cada envío puede tener más de un artículo en la tabla *line_send* se almacenará la relación entre el envío y los artículos, con esa relación podemos saber si todos los artículos de un envío han llegado correctamente y poder marcar el estado del envío como resuelto o si algún artículo no ha llegado a su destino por cualquier motivo, indicarlo y marcar el estado del envío como recibido parcialmente.

Al tener esta estructura nos permite escalar el proyecto a cualquier empresa que quiera usarlo, solo se tendría que crear la empresa, los distintos almacenes, usuarios y asociar a cada usuario al almacén que corresponda.

4.2 Diagrama de clases

En este apartado se muestran los diagramas de clases que se ha utilizado para el TFG. Ya que este TFG cuenta con dos aplicaciones, una para el cliente y otra para el api tendremos dos diagramas.

4.2.1 Cliente

En este apartado se explicarán las clases que componen la aplicación del cliente, explicando los métodos más importantes de cada clase.

Empecemos por explicar la clase **Home**, esta es la puerta de entrada de nuestra aplicación y desde la cual guardaremos el token que nos proporciona firebase para enviar las notificaciones PUSH.

La clase **Article** gestiona todos los artículos a los que tenemos acceso agrupados por almacenes, para ello utilizamos el método getAllWarehouse() el cual nos devolverá todos nuestros almacenes. El método onChangeSelected() nos permite recuperar todos los artículos del almacén seleccionado. Los métodos getItems(), initializeItems() y reinitialize() se usan para autocompletar la búsqueda de artículos según se vaya escribiendo.

La siguiente clase **Send** muestra todas las salidas que se han realizado desde uno de nuestro almacenes, para ello utilizamos el método getAllSend(), en esta clase también podemos filtrar las salidas en un intervalo de fechas para ellos se utiliza el método search() y por último podemos ver los detalles de cada salida (cuántos artículos se han enviado, quien realizó la salida, cuál es el almacén de origen y destino, etc.).

Otra de las funcionalidades de la aplicación es realizar salida de artículos, para ello usamos la clase Send-add la cual cuenta con métodos como:

- getAllWarehouses() : Recupera todos nuestros almacenes.
- getAllLikeArticlesByNameAndWarehouse(): Muestra los artículos que se corresponda con los escrito por el usuario y que pertenezca al almacén seleccionado.
- sendDataParams(): Método para crear una salida.
- addArticle(): Añade los artículos que serán enviados.

Otra de las clases importantes es Receive la cual gestiona las entradas de artículos a nuestros almacenes, es una clase muy parecida a la clase Send contando con métodos como getAllReceive() el cual se encarga de obtener todos los artículos que haya llegado y search() que nos permite buscar en un rango de fechas.

Por último tenemos la clase Receive-details desde la cual podremos confirmar que los artículos han llegado correctamente con el método saveEntry().

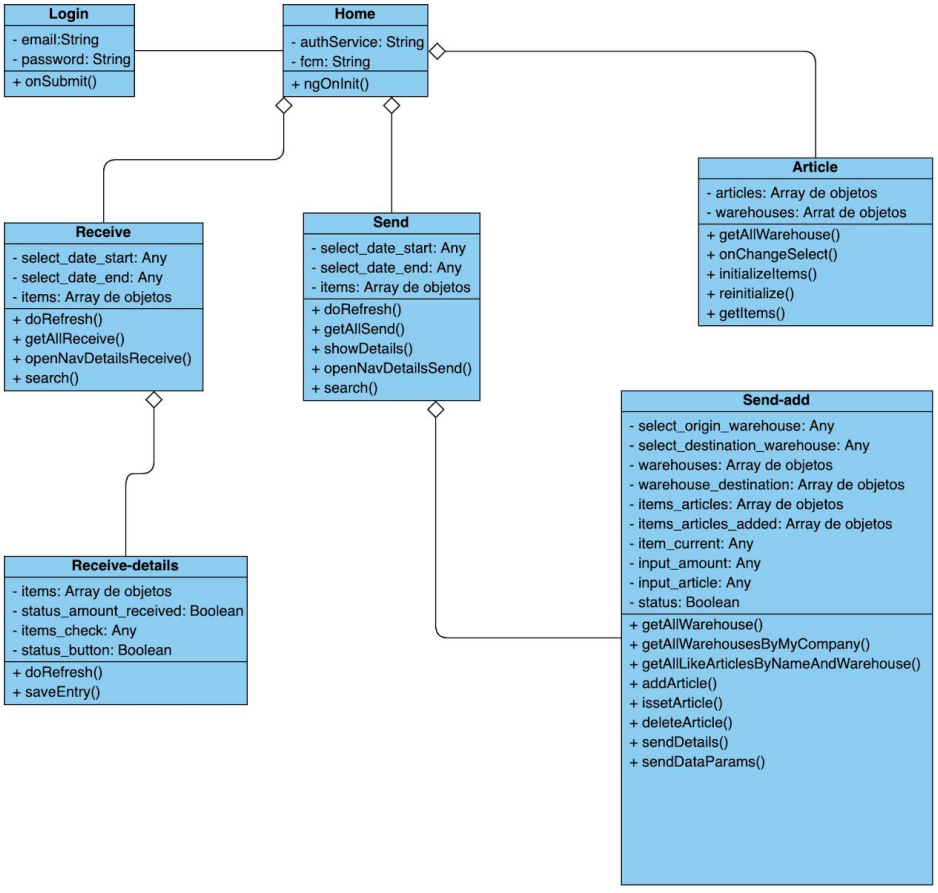


Ilustración 16. Diagrama de clases cliente

4.2.2 Servidor

Como se ha comentado la aplicación servidor será un api rest, la cual cuenta con el siguiente diagrama. Al igual que en el apartado anterior se explicará los métodos más importantes de las clases.

Empezaremos por la clase **Login**, la cual se encarga de gestionar el token de los usuarios siempre que éstos introduzcan sus credenciales correctamente, esto se consigue con el método login(). La clase login se comunica con la clase User ya que es desde la cual valida las credenciales de acceso.

La clase **Article** nos permite gestionar los artículos de nuestro sistema ayudándose de métodos como getArticleByNameAndWarehouse() y getArticlesByWarehouse(). Esta clase interacciona con la clase **Warehouse** y **ArticleWarehouse** permitiendo así la gestión de los artículos de los que dispone cada almacén en un momento determinado, eso lo conseguimos con métodos como getAllArticleWarehouseByIdWarehouse().

Cada usuario pertenece a uno o más almacenes esta gestión la realizamos con la clase **UserWarehouse**.

Por último pero no menos importante tenemos la clase **Send**, con la cual gestionamos todas las salidas y entradas de artículos.

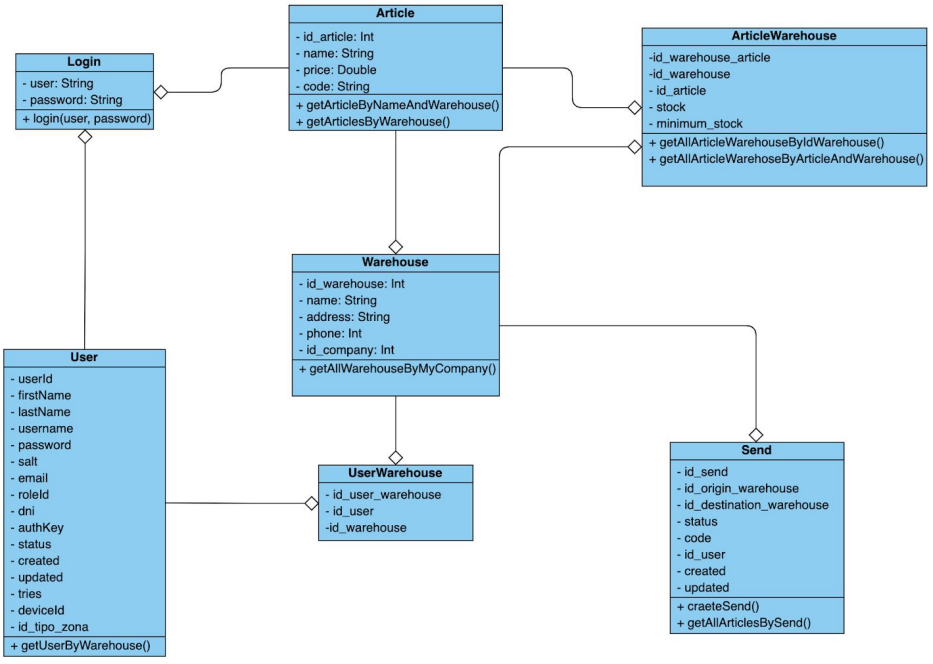


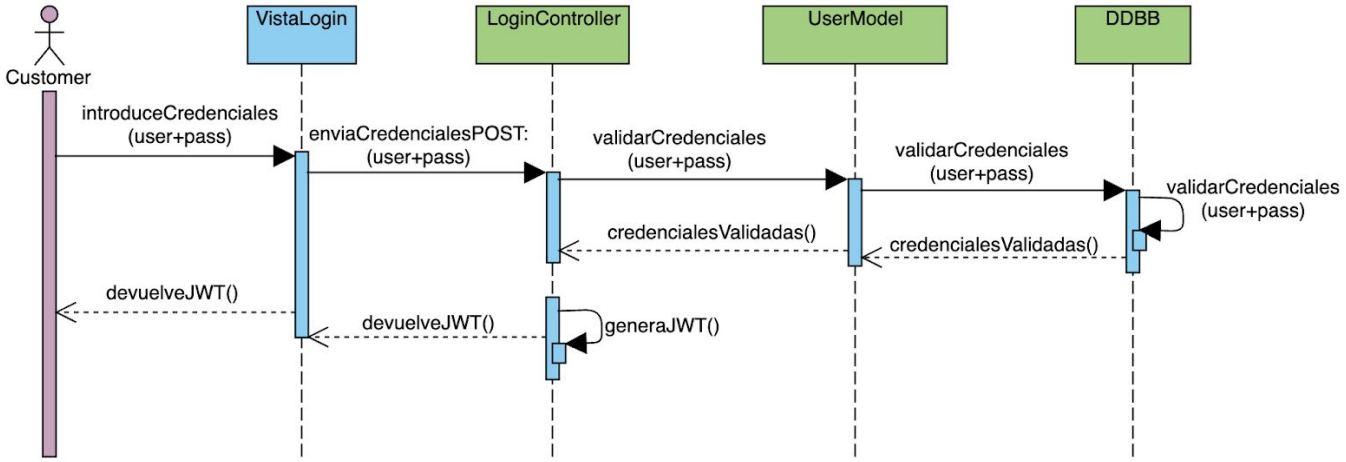
Ilustración 17. Diagrama de clases servidor

4.3 Diagrama de secuencia

En estos diagramas se muestra la interacción entre el usuario y el sistema a través del tiempo. Ya que la aplicación cuenta con un parte Frontend y otro Backend se dibujarán las líneas de tiempo de distinto color, azul para el frontend y verde para el Backend.

4.3.1 Login

Ilustración 18. Diagrama de secuencia LOGIN



En este diagrama se muestra cómo el usuario introduce sus credenciales de acceso y estas son validadas por la API REST generando un token de tipo JSON Web Tokens o JWT[18] el cual es un estándar abierto basado en JSON y utilizado para la creación de tokens de acceso que permiten identificar y dar privilegios a los usuarios. JWT se puede usar prácticamente para todos los lenguajes, por ejemplo Node.js, Python, Java, JavaScript, PHP y existen librerías para crear y gestionar estos tokens.

[18] https://jwt.io/

4.3.2 Listar artículos

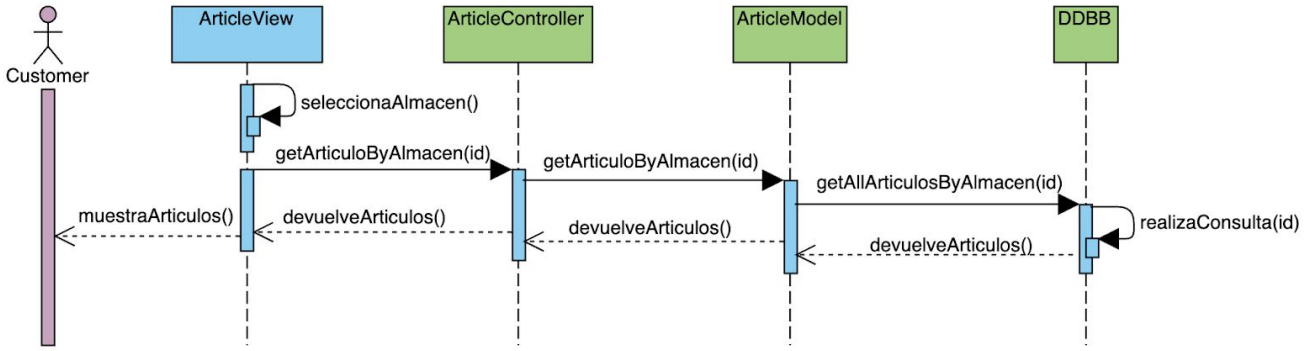


Ilustración 19. Diagrama de secuencia Listar artículos

En esta secuencia se muestra cómo el usuario lista los artículos de un almacén en concreto, para ello lo primero que hace es seleccionar desde la vista el almacén, la vista realiza una petición GET al servidor con el almacén seleccionado, esta petición llega al controlador **ArticleController** el cual se comunica con el modelo para obtener los artículos y éste realiza la consulta a base de datos para buscar los artículos del almacén seleccionado.

4.3.3 Buscar salida

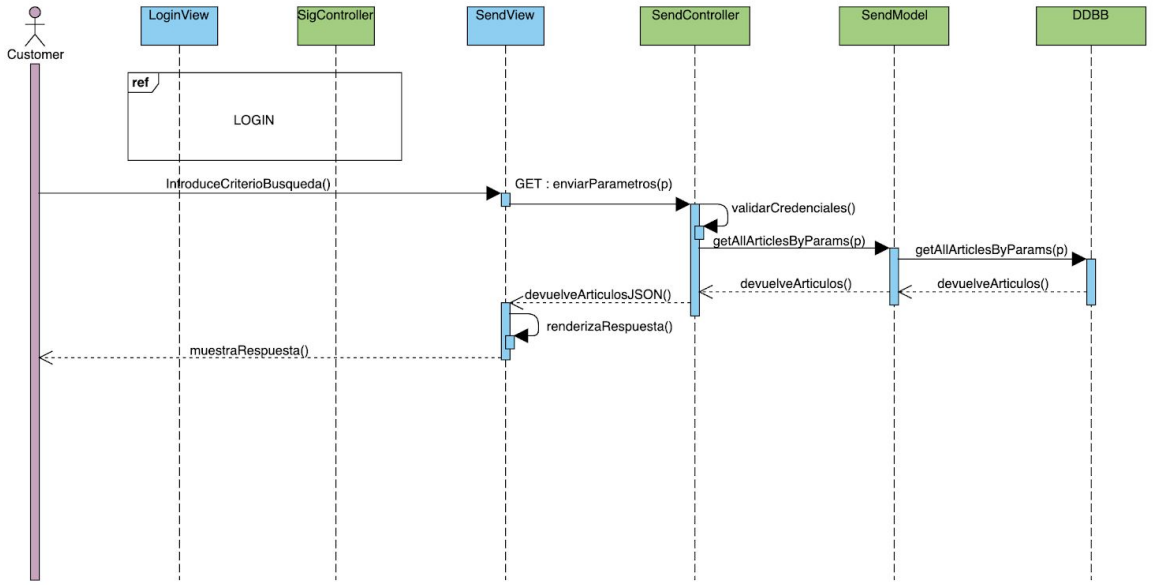


Ilustración 20. Diagrama de secuencia Buscar salida

En esta secuencia se muestra cómo el usuario realiza una búsqueda entre todas las salidas realizadas por el mismo, para ello lo primero que hace es iniciar sesión y acto seguido introduce en la vista los criterios de búsqueda, estos parámetros se pasan al servidor a través de una petición http GET desde la vista.

Este petición la recibe el controlador y lo primero que hace es comprobar la validez del token, si el token es correcto este se comunica con el modelo para recuperar los datos que cumplan con el criterio de búsqueda y devolverlos a la vista en formato JSON para posteriormente mostrarlo al usuario.

4.3.4 Generar salida

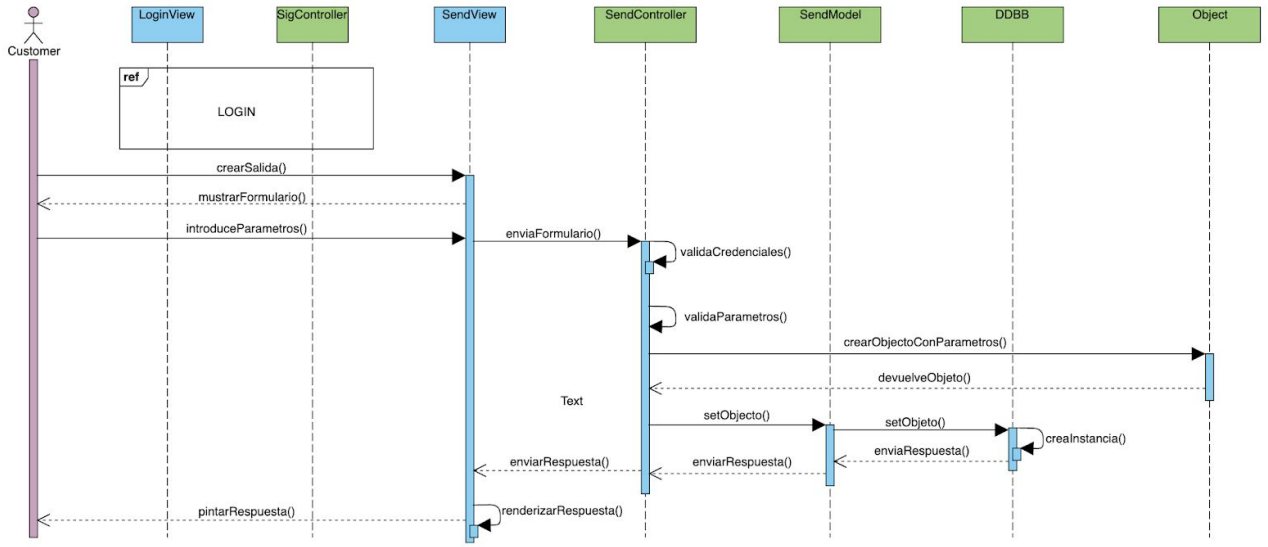


Ilustración 21. Diagrama de secuencia Generar salida

En esta secuencia se muestra cómo el usuario genera salidas de artículos a un almacén. El inicio de la secuencia es igual que el diagrama anterior, lo primero que hace el usuario es loguearse en la aplicación.

Lo siguiente es dirigirse al apartado salida y pulsar en crear salida, esto abrirá un formulario con todos los parámetros necesarios para poder realizar dicha salida, parámetros tales como almacén de origen, almacén destino, artículos, observación, etc.

Cuando el cliente haya complementado todo el formulario, pulsará en enviar y la vista realizará una petición POST al servidor, esta petición la recibirá el controlador el cual lo primero que hará será validar el token, luego validará los parámetros de entrada para evitar así introducir parámetros que no cumplan con las condiciones del sistema.

Cuando se hayan realizado todas las comprobaciones el controlador instancia un objeto el cual tendrá todos los parámetros necesarios para crear la salida. Por último se creará la salida enviando el objeto.

La vista siempre recibirá una respuesta en formato JSON, tanto si todo ha ido bien como si no.

4.3.5 Confirmar entrada

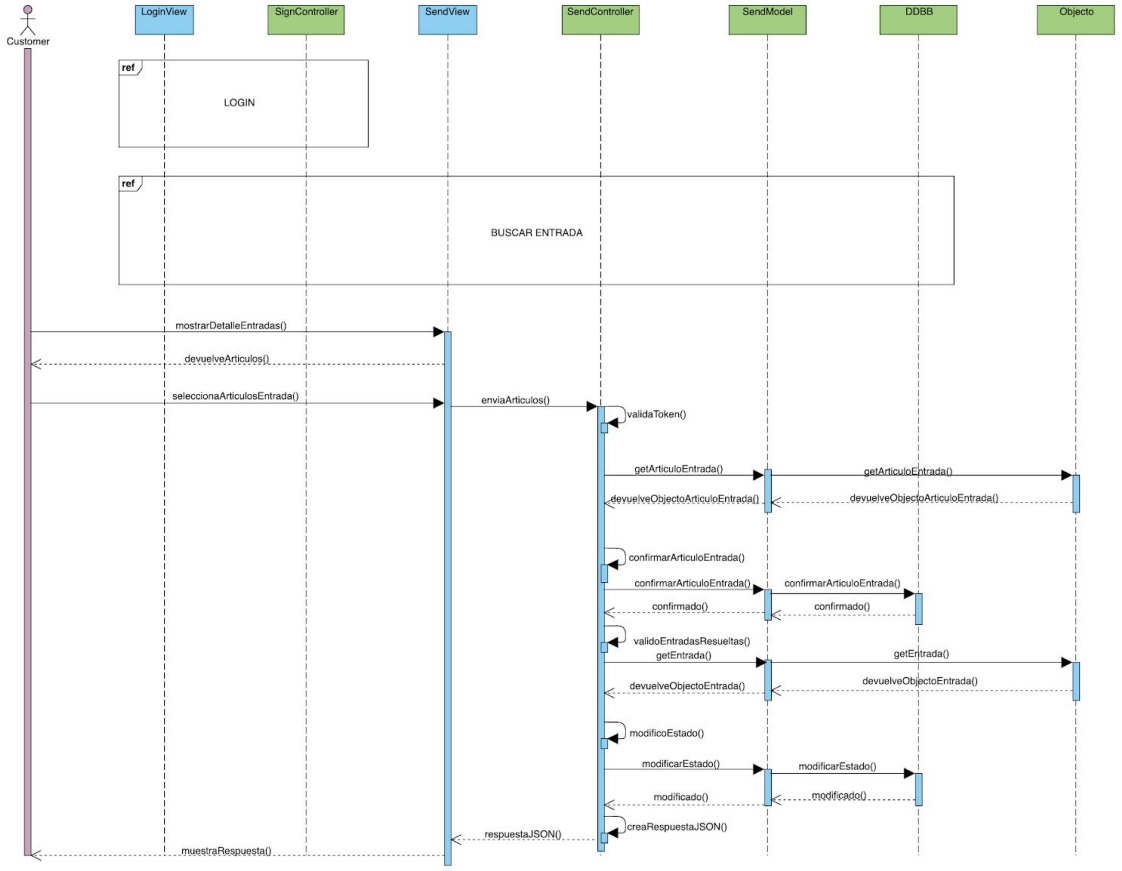


Ilustración 22. Diagrama de secuencia Confirmar entradas

Este diagrama es el que más interacciones tienen entre los objetos, lo primero y al igual que los demás diagramas (excepto el de LOGIN) es que el usuario tiene que iniciar sesión. Luego tiene que buscar la entrada, la cual tendrá los artículos a confirmar, para ello nos apoyamos en el diagrama 17.

Cuando el usuario haya encontrado la entrada seleccionará los artículos que desea confirmar y los enviará al servidor mediante una petición PUT al servidor.

Lo primero al llegar la petición al servidor es validar el token, luego se validan todos los datos introducidos por el cliente y se confirma la entrada de los artículos. En caso que todos los artículos de la entrada hayan sido confirmados la entrada cambia de estado a “resuelto”, por último se envía la respuesta a la vista en formato JSON para que ésta pueda dar feedback al cliente sobre la petición.

4.3.6 Buscar entradas

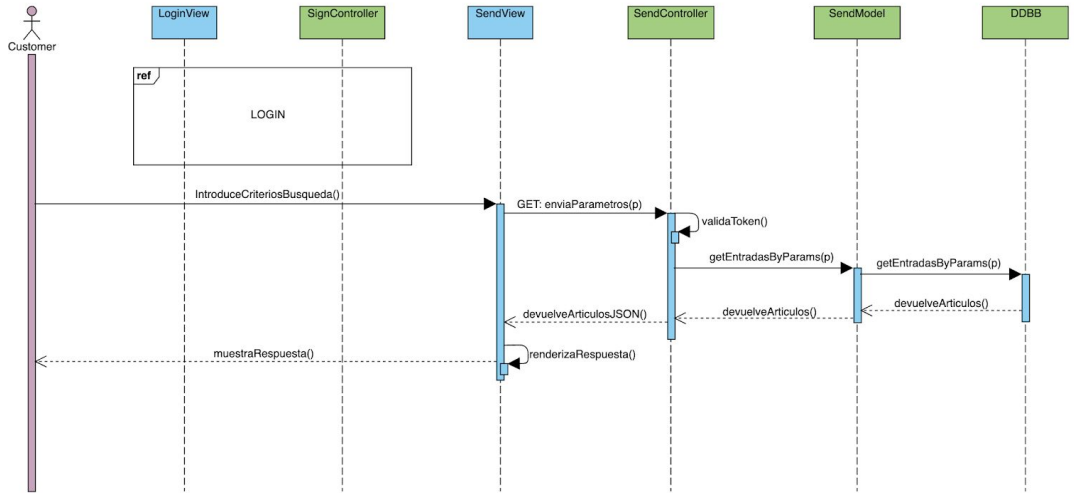


Ilustración 23. Diagrama de secuencia Buscar entradas

Este diagrama de secuencia es muy parecido al diagrama de secuencia “Buscar salida”, con la diferencia que en lugar de buscar los artículos que ha sacado de alguno de sus almacenes, busca todos los artículos que hayan entrado a alguno de sus almacenes.

4.4 Diseño de la interfaz

En este apartado se hablará sobre el diseño de la interfaz y dado que el TFG consta de dos partes, aplicación multiplataforma móvil con Ionic y API REST se dividirá en dos partes.

4.4.1 Interfaz de usuario gráfica

Para realizar la interfaz de usuario y después de ver algunos de los software más populares para el diseño de interfaces me decanté por Figma[19], un software de diseño UI que cuenta con muchas ventajas como p.e.

- Interfaz amigable
- Todo se almacena en la nube
- Permite trabajar en proyectos colaborativos
- Puedes acceder desde la web

Quizás el punto por el cual decidí usar Figma es lo fácil que es empezar a trabajar ya que no hace falta descargar ningún archivo ni instalar nada en nuestro ordenador, basta con registrarse en su web y empezar a usarlo.

Aunque también cuenta con instaladores para todos los sistemas operativos, a mi personalmente me gusta mucho la idea de tener todos los componentes accesibles desde la web, ya que me permite ser más productivo y poder diseñar las interfaces por ejemplo desde un ipad sin problemas.

Otra de las ventajas de figma es que permite crear interacciones entre las vistas creadas y compartirlas a un tercero para que pueda darnos feedback.

Para acceder al prototipo de la interfaz de este proyecto pulse el siguiente enlace (véase enlace[20]).

Para una mejor comprensión de las vistas, se mostrará cada una de ellas por separado y se explicará brevemente para qué sirven.

[19] https://www.figma.com/
[20] https://bit.ly/30bs9u8

4.4.1.1 Iniciar sesión

En esta vista el usuario tendrá que indicar las credenciales de acceso para poder acceder a la aplicación.

Ilustración 24. Vista iniciar sesión

4.4.1.2 Home

Esta es la vista principal en la cual el usuario podrá buscar ver todos los artículos seleccionando un almacén (entre los almacenes solo aparecerán aquellos que tenga asignado).

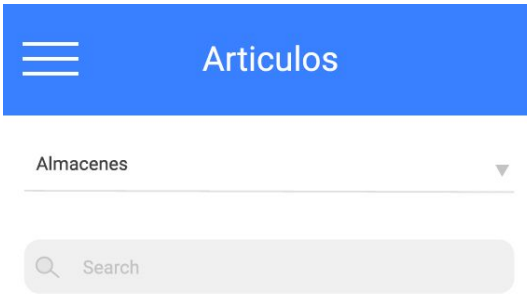


Ilustración 25. Vista principal

4.4.1.3 Buscar artículos

Cuando el cliente seleccione un almacén se cargarán automáticamente todos los artículos correspondientes al mismo.

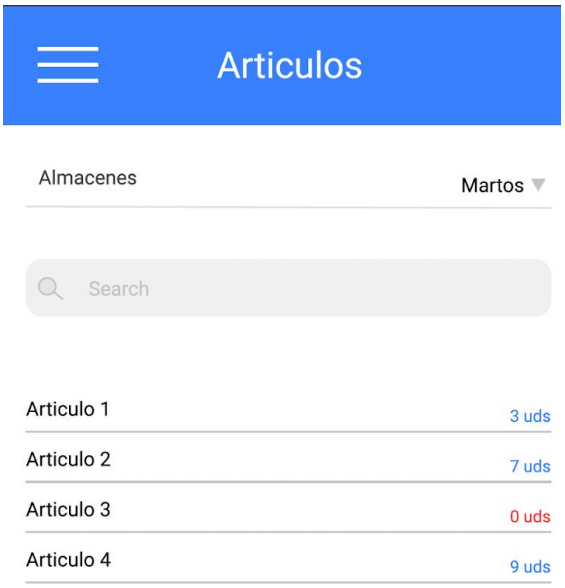


Ilustración 26. Vista buscar artículos

4.4.1.4 Menú

Esta vista corresponde al menú que tendrá la aplicación, como se puede ver se accede a la vista SALIDAS, ENTRADAS y ARTÍCULOS, además de la posibilidad de cerrar sesión.

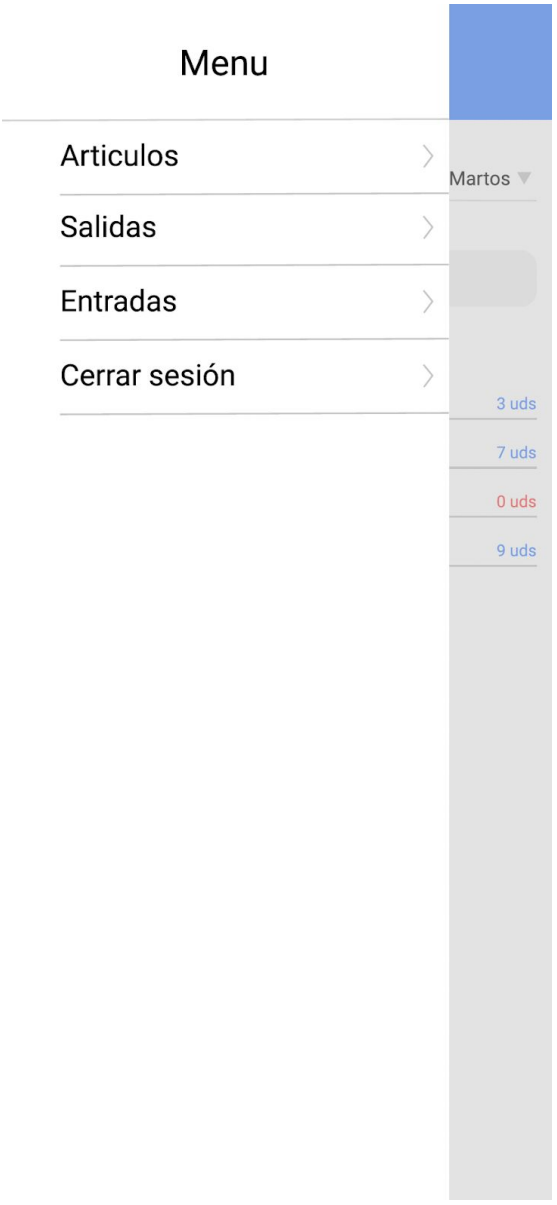


Ilustración 27. Vista menú

4.4.1.5 Salidas

En esta vista se podrá crear una salida, ver todas las salidas que hayamos realizado o filtrar por un rango de fechas.

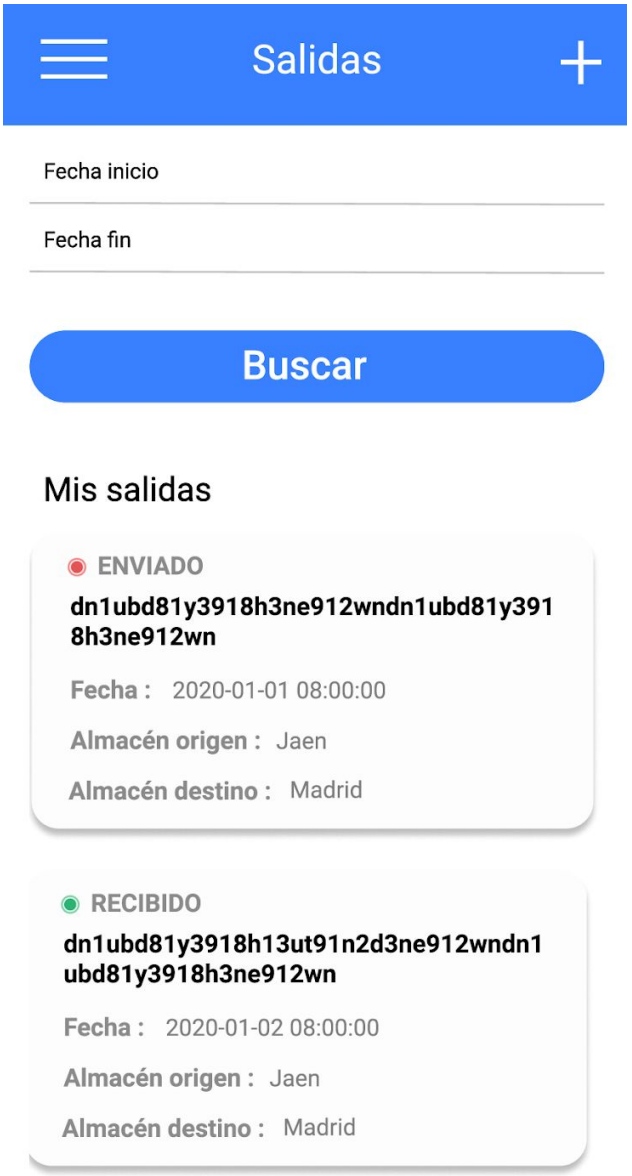


Ilustración 28. Vista salidas

4.4.1.6 Crear salida

Esta es la vista desde la cual el usuario podrá crear una salida, como se puede ver se ha dividido en dos pasos. Primero, seleccionar el origen y destino; segundo, indicar el/los artículo(s) con la cantidad correspondiente y por último pulsar en el botón enviar.

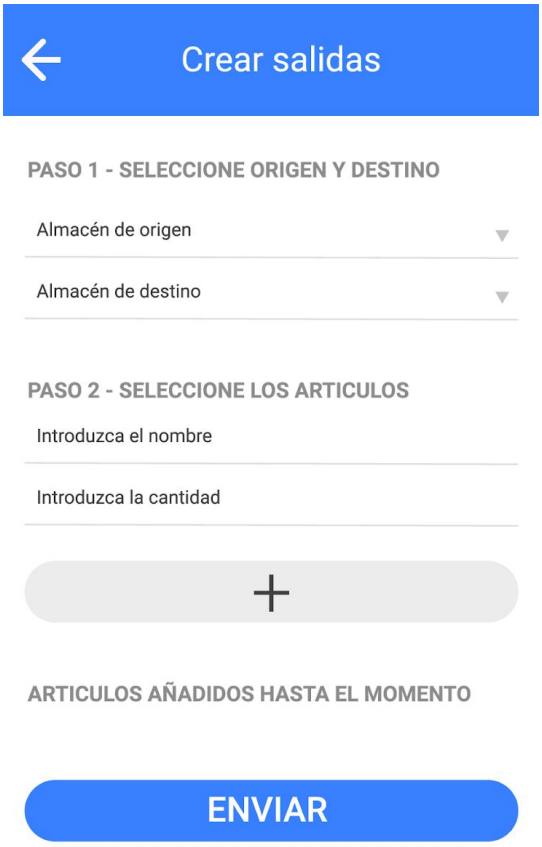


Ilustración 29. Vista para crear una salida

4.4.1.7 Entradas

En esta vista el usuario podrá ver todas las entradas de todos los almacenes que tiene asignado.

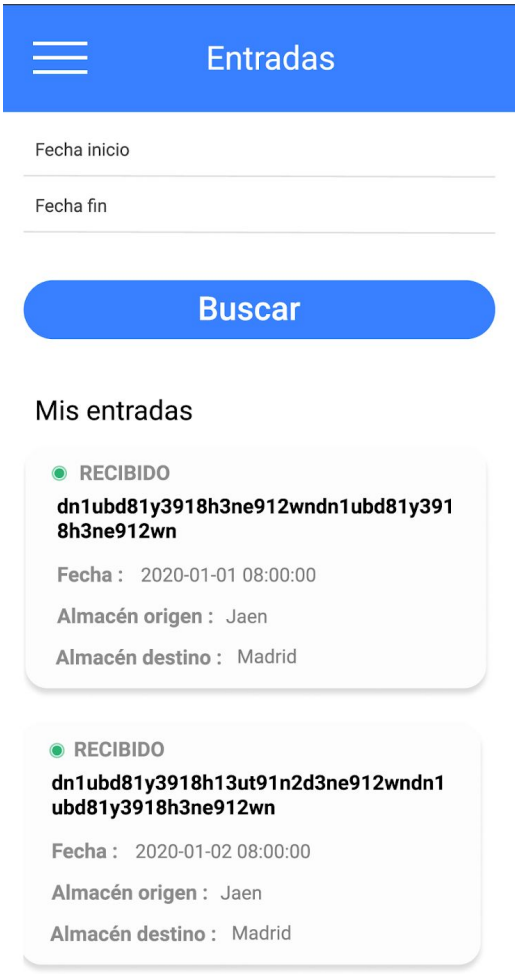


Ilustración 30. Vista entradas

4.4.1.8 Detalle entrada

Esta vista es desde la cual el usuario podrá confirmar que los artículos han llegado a su destino correctamente.

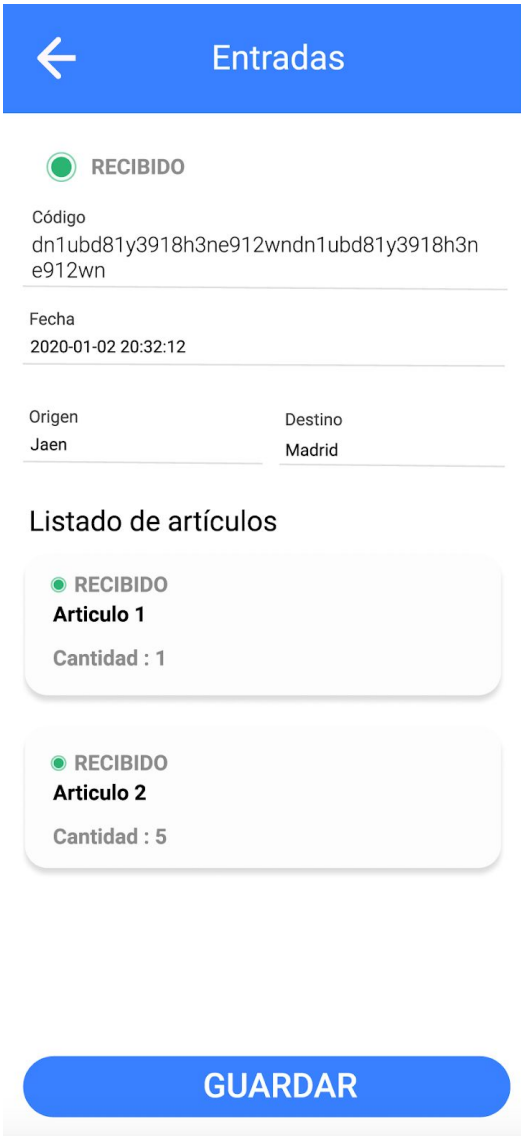


Ilustración 31. Vista detalles de una entrada

4.4.2 Diseño de api rest

En este apartado se describen todos los endpoint con los que cuenta nuestra API.

Como hemos comentado utilizamos una API REST para obtener datos a través de llamadas HTTP. Los datos pueden intercambiarse en muchos formatos (JSON, XML, etc.). Los verbos HTTP más usados son cuatro: POST (crear), GET (leer y consultar), PUT (editar), PATCH (editar), DELETE (eliminar). Además, al utilizar REST tenemos muchas ventajas como es contar con un estándar lógico y eficiente.

Para poder utilizar esta API desde diferentes clientes, es fundamental disponer de una buena documentación de la misma, ya que en muchas cosas las API REST son creadas para que puedan interactuar con otros sistemas y si no contamos con una documentación precisa el api deja de ser accesible y pierde todo su sentido.

Para realizar la documentación se ha utilizado la herramienta swagger[21], la cual permite generar documentación de una forma fácil y accesible, permitiéndonos ver y probar todos los endpoint de nuestra API desde un navegador tal y como se muestra en la ilustración 32. También podemos acceder a ellos desde el siguiente enlace (véase enlace[22]).

Esta documentación al igual que la aplicación en ionic se han instalado en un servidor de pruebas para conseguir que sea más accesible, aunque al desplegarse en producción se podrá consultar de la misma forma con la diferencia que estará apuntando a otra dirección IP (si procede).

[21] https://swagger.io/
[22] http://51.83.45.39:8030/swagger-ui-dist/

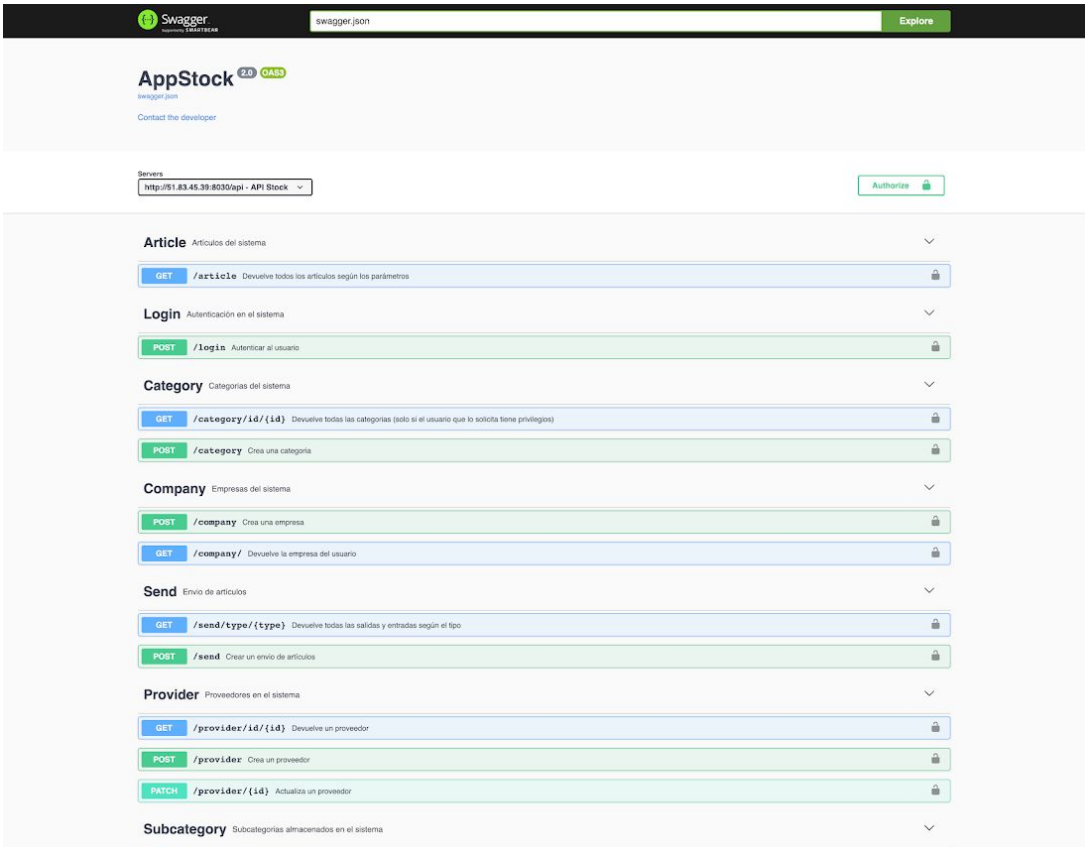


Ilustración 32. Documentación en swagger

4.5 Plan de pruebas

Para realizar las pruebas los separaremos en dos apartados, uno será la parte del cliente y otra la del servidor.

En la parte del cliente probaremos que todos los casos de uso funcionan correctamente además de probar también que se cumplen los requisitos funcionales y no funcionales.

En la parte del servidor comprobaremos que todos los endpoint funcionan correctamente, para ello nos ayudaremos de la documentación de la API REST descrita en el apartado anterior, ya que swagger cuenta con la opción de probar cada uno de los endpoint, incluida la autenticación que en nuestro caso es mediante JWT.

5. Implementación

En este apartado hablaremos sobre la arquitectura del proyecto además de los detalles sobre la implementación del mismo.

5.1 Arquitectura

Para explicar la arquitectura de este TFG empezaré por explicar las dos partes en las que se divide, el Front-end y el Back-end.

5.1.1 ¿Qué es el Front-end?

A grandes rasgos el frontend es todo lo que podemos ver de un sitio y la interfaz con la cual interactúa el usuario.

Las tres tecnologías principales son HTML, CSS y JavaScript, las cuales nos permiten dar una estructura, ponerle estilos e interacciones.

Además de crear aplicaciones web, con estas tecnologías podemos crear apps híbridas utilizando algunos frameworks como son ionic, como es el caso de este TFG.

5.1.2 ¿Qué es el Back-end?

El backend es todo lo que pasa tras bambalinas, conexión a los servidores y BBDD.

El backend es una parte muy importante ya que hace que todo funcione correctamente. Tener un buen backend ayuda en la eficiencia de las aplicaciones y como contrapartida un pequeño error puede desembocar en un

gran problema, es por ello que tenemos que tener especial atención cuando programamos esta parte.

Tal y como se muestra en la ilustración 33 la parte del front se ha realizado con tecnologías web e ionic un framework para empaquetar y generar aplicación nativas en Ios y Android.

Para la parte del backend se ha utilizado una VPS con el proveedor OVH[23]. En esta VPS tenemos el servidor web y la base de datos.

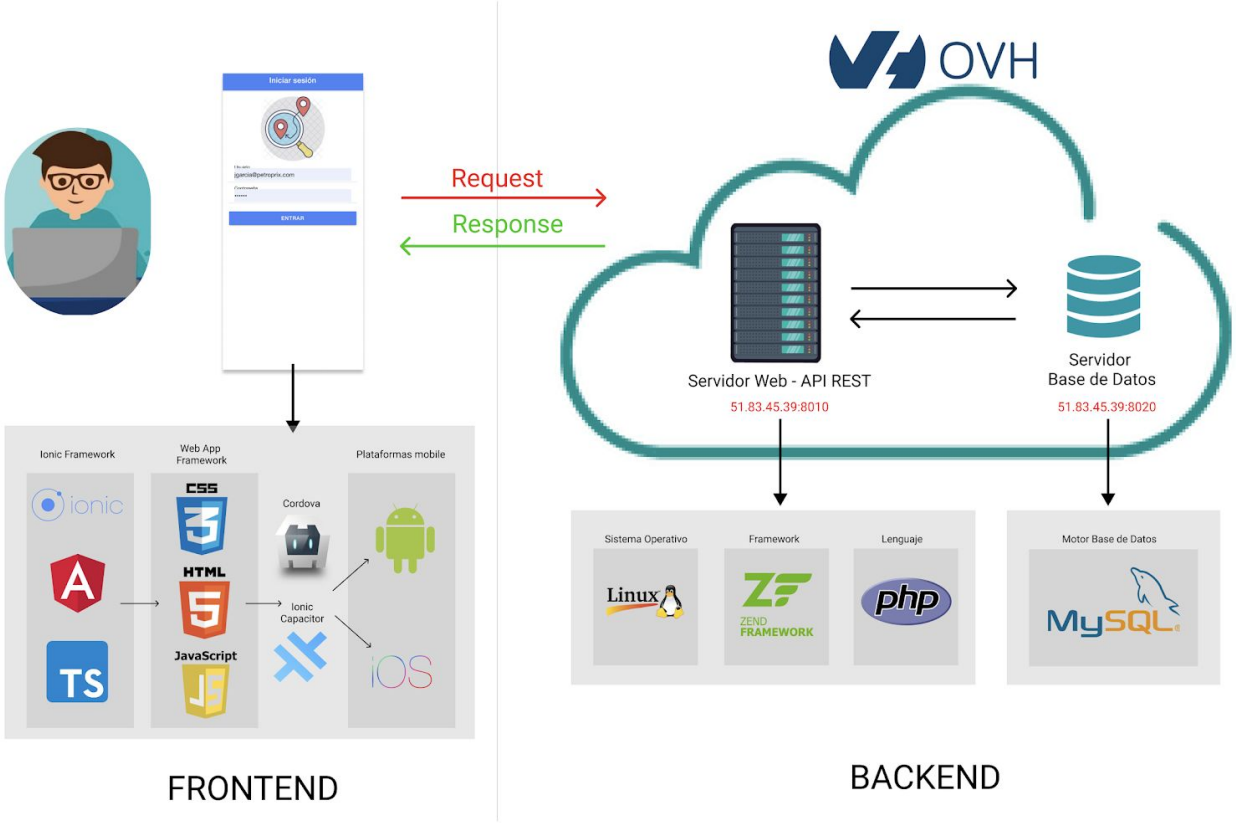


Ilustración 33. Arquitectura

5.2 Detalles de la implementación

En este apartado se hablará sobre los aspectos técnicos de la implementación, explicando cuáles han sido los framework utilizados, cómo desplegar el proyecto, aspectos importantes de funcionamiento tanto en el frontend como en el backend, detalles de comunicación entre el frontend y el backend, y este a su vez con la base de datos, entre otros detalles de la implementación.

[23] https://www.ovh.es/

5.2.1 Entorno de desarrollo

Los entornos de desarrollo utilizados para realizar la implementación han sido dos.

Por un lado VSCode[24], un editor de código desarrollado por Microsoft y soportado para plataformas Windows, Linux y macOs. VSCode a día de hoy es posiblemente el editor de código más popular entre los desarrolladores ya que cuenta con numerosas extensiones las cuales hacen que los desarrolladores se centren completamente en crear.

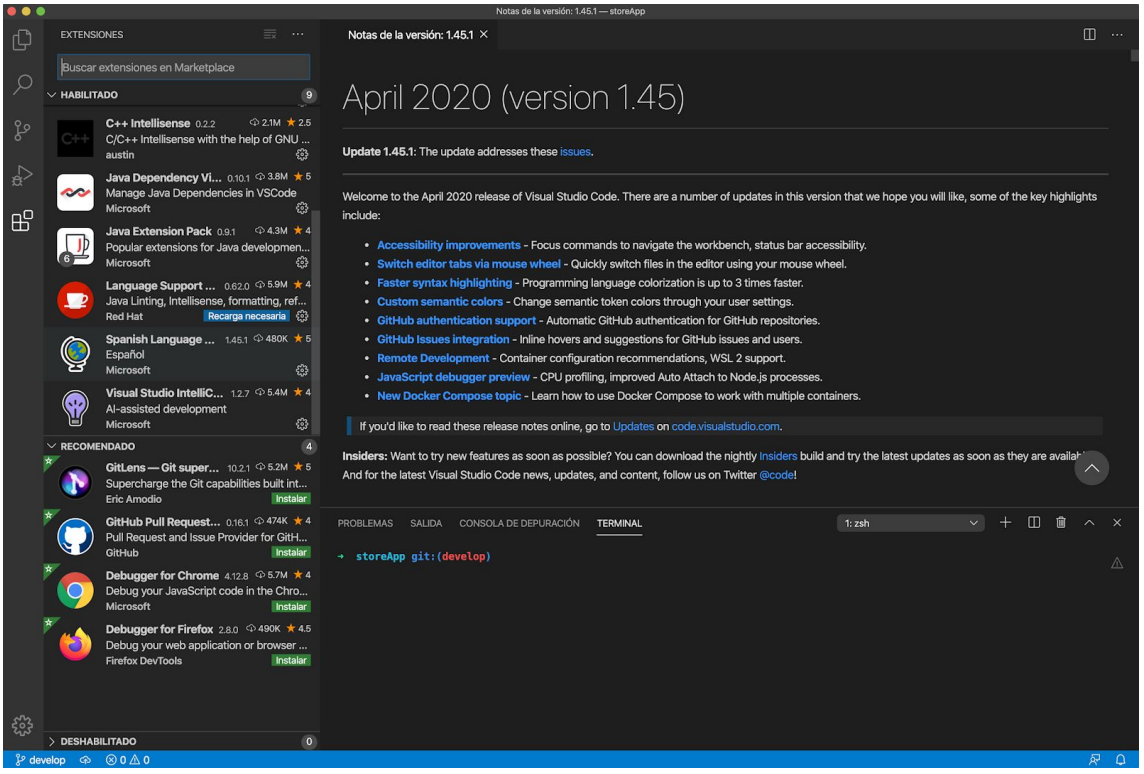


Ilustración 34. Visual Studio Code

Con VSCode se ha desarrollado prácticamente todo el proyecto, tanto el frontend como el backend y ya que cuenta con un terminal integrado también se ha usado para integrarlo con el sistema de control de versiones Git.

El segundo entorno de desarrollo que se ha usado es XCode[25] un entorno de desarrollo integrado únicamente para macOS. Este entorno fue creado por Apple y permite crear aplicación para todo el ecosistema de Apple, macOS, iOS, watchOS y tvOS.

[24] https://code.visualstudio.com/
[25] https://apps.apple.com/es/app/xcode/id497799835?mt=12

Ilustración 35. XCode

5.2.2 Tecnologías utilizadas en el frontend

En este punto hablaremos de todas las tecnologías que conforman el frontend, las cuales son:

5.2.2.1 Ionic

Ionic es un SDK abierto para desarrollar aplicaciones híbridas utilizando tecnologías web y Cordova. La primera versión de Angular se lanzó en el 2013 con Angular y Cordova, pero en la última versión se constituyó como un conjunto de componente web lo cual permite a los desarrolladores utilizar otras tecnologías como Vue.js o React.

¿Cómo arrancar un proyecto con ionic?

El primer paso para crear un proyecto con ionic es instalar Nodejs. Cuando tengamos instalado Nodejs, el segundo paso será la instalación de Ionic Cli para ello escribiremos en nuestra consola el siguiente comando.

```
> npm install -g ionic
```

Cuando termine la instalación podemos ver que se ha instalado correctamente ejecutante el siguiente comando.

```
> ionic --version
```

El tercer paso es crear nuestra primera aplicación, para ello ejecutaremos el siguiente comando en la terminal.

```
> ionic start myApp sidemenu
```

Cabe mencionar que ionic nos proporciona tres plantillas, tal y como se muestra en la ilustración 36.

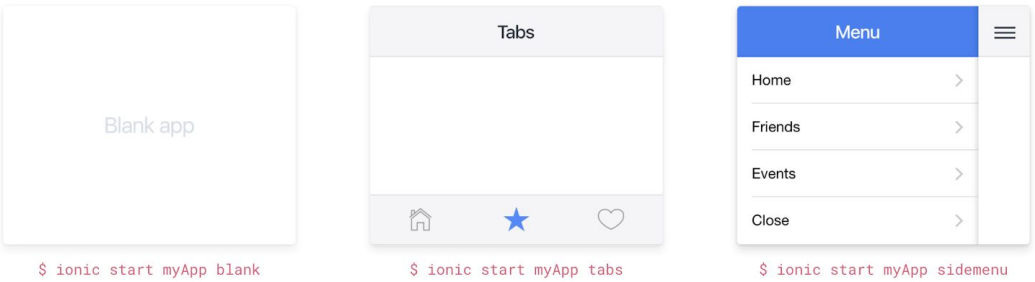


Ilustración 36. Ionic templates

Por último cuando nuestro proyecto de ionic se haya creado, desde la terminal nos situaremos en la carpeta principal del proyecto y escribiremos el siguiente comando.

```
> ionic serve
```

Este comando mostrará nuestro proyecto en ***localhost:8000***. Tanto el puerto como la dirección donde queramos que se ejecute se pueden cambiar, por defecto ionic ejecute nuestra aplicación en el puerto 8000, tal y como se muestra en la ilustración 37.

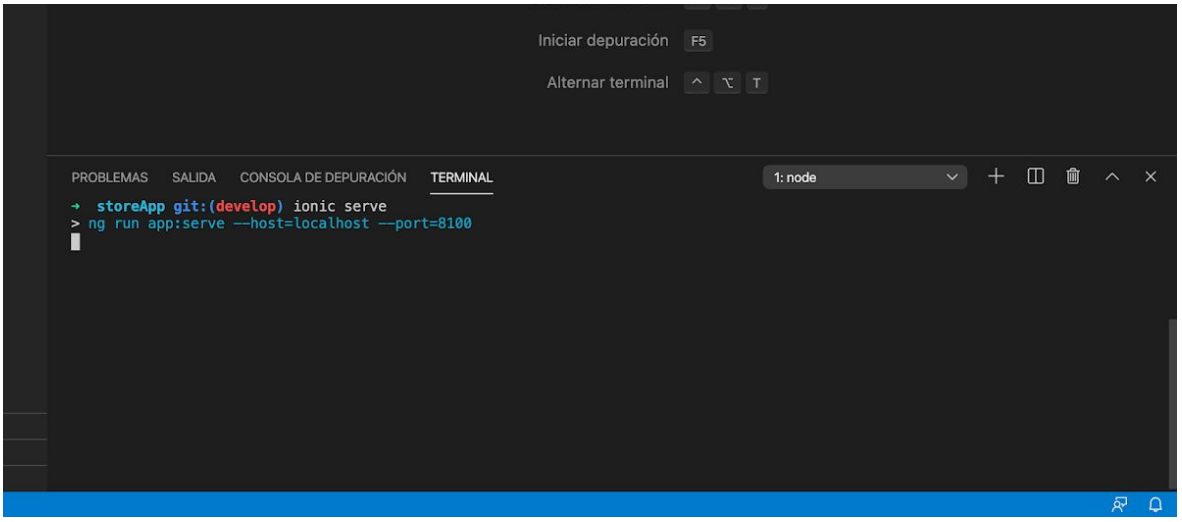


Ilustración 37. Ionic serve

¿Cómo está estructurado Ionic?

Para entender mejor Ionic y facilitar posteriores modificaciones se explicará cómo está dividido el esqueleto de un proyecto con Ionic.

Como podemos ver en la ilustración 38, los elementos más importantes que encontramos en esta estructura a groso modo son **platforms**, **src**, **package.json y tsconfig.son,** se explicará para qué sirven cada uno de ellos.

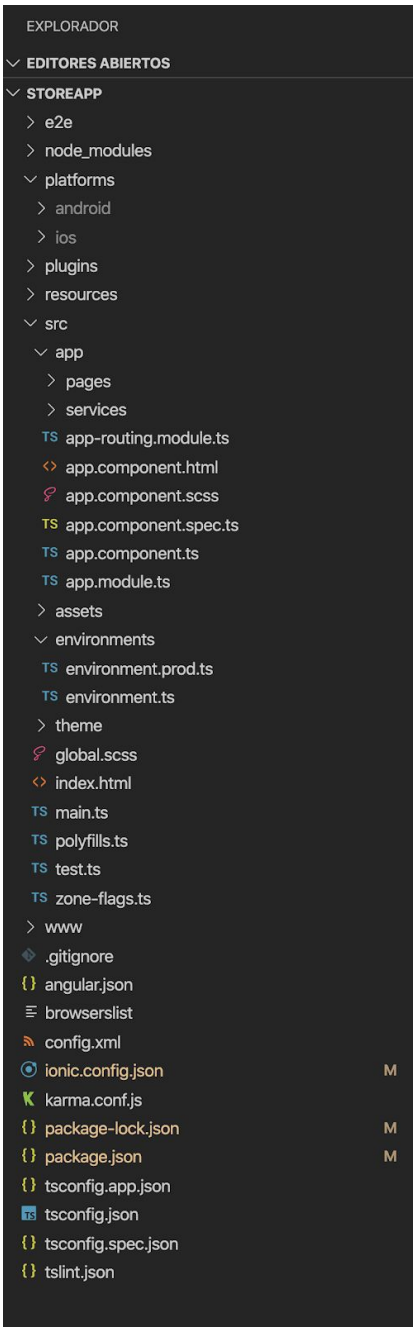


Ilustración 38. Estructura de un proyecto - Ionic

- **platforms** : Contiene el código que posteriormente utilizaremos para las aplicaciones nativas de cada sistema operativo. En el caso de Android se utilizará el IDE Android Studio y en el caso de iOS se utilizará XCode.

- **src** : Contiene todo el código de la aplicación.

    - **app** : En este carpeta guardaremos casi todo el código excepto los estilos globales.

        - **pages** : En esta carpeta guardamos información relativo a las vistas, tanto de diseño como la lógica de negocio.

        - **services** : En esta carpeta guardaremos los servicios necesarios para que nuestra aplicación funcione correctamente, algunos de estos servicios son para la autenticación del usuario, manejo del token JWT y llamadas a la API.

    - **assets** : Imágenes, vídeos y demás recursos de la aplicación.

    - **environments** : Ficheros para almacenar variables de entorno.

    - **theme** : Contiene los css o scss globales.

    - **main.ts** : El punto de entrada a nuestra aplicación, se ayuda del módulo raíz (AppModule**)** para ejecutarse en el navegador o donde corresponda.

- **package.json** : Archivo de configuración de un proyecto de Node, en el cual se definen características importante como:

    - Nombre del proyecto
    - Versión
    - Dependencias
    - Scripts
    - Repositorios
    - Autores
    - Licencias

- **tsconfig.json** : Archivo donde especificamos los ficheros raíz y los elementos necesarios para que el compilador funcione correctamente.

¿Cómo crear páginas?

Para crear páginas utilizaremos el CLI (Command-line interface) de ionic ejecutando el siguiente comando.

```
> ionic generate page <nombre_de_la_vista>
```

También podemos utilizar la versión reducida de la siguiente forma.

```
> ionic g page <nombre_de_la_vista>
```

Ionic CLI[26] es una herramienta muy potente que nos ayudar a desarrollar de una forma más rápida. Entre sus comandos más relevantes encontramos.

- build
- generate
- info
- serve
- cordova build
- cordova platform

Entre otros.

Cuando terminamos de ejecutar el comando anterior veremos que automáticamente nos creará un fichero con tres archivos. El fichero tendrá el nombre con el que lo hayamos creado tal y como se muestra en la ilustración 39.

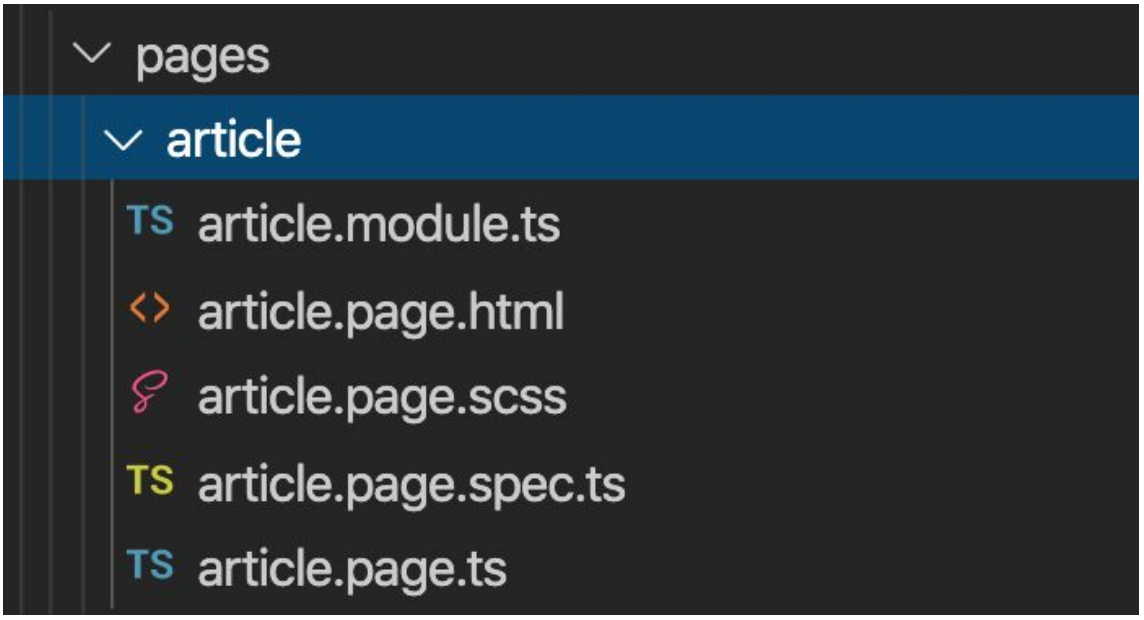


Ilustración 39. Archivos que conforma una vista

[26] https://ionicframework.com/docs/v3/cli/commands.html

- **article.module.ts**: En este archivo se importan los módulos necesarios para que nuestra página funcione.

- **article.page.html:** Contiene la plantilla HTML.

- **article.page.scss:** Contiene el estilo de la página, utilizando el preprocesador sass

- **article.page.spec.ts:** Este archivo se usa para realizar pruebas unitarias.

- **article.page.ts:** En este archivo typescript guardaremos toda la lógica y comportamientos de nuestra página, por ejemplo realizar llamadas a un API REST, añadir un comportamiento cuando se pulse un botón, etc..

¿Cómo se comunica con el backend?

Para realizar las peticiones al backend se usará el módulo HttpClient, para ello lo primero que tenemos que hacer es incluirlo en nuestro archivo **app.module.ts**, este paso es importante si queremos utilizar esta dependencia en nuestra APP.

Cuando hayamos importado el módulo usaremos el CLI de ionic para generar nuestro nuevo servicio, para ello ejecutamos el siguiente comando.

```
> ionic g services <nombre_del_servicio>
```

En este archivo crearemos todas las peticiones a nuestra api, tal y como se muestra en la ilustración 40.

```
getAllArticles(id_alm) {
  this.httpOptions = {
    headers : {
      'Authorization' : "Bearer " + this.token
    }
  };

  return this.http.get(`${this.url}/article/id_warehouse/${id_alm}`, this.httpOptions).pipe(
    catchError(e => {
      let status = e.status;
      if (status === 401) {
        this.showAlert('You are not authorized for this!');
      }
      throw new Error(e);
    })
  );
}
```

Ilustración 40. Request HTTP - IONIC

Con esto tendríamos una parte para realizar las peticiones, pero ahora necesitamos que alguna página consuma este servicio, para ello usaremos de ejemplo la página **Article**.

Empezaremos por analizar el html de esta página, ilustración 41. Esta página la encontramos en la siguiente ruta, **src** > **pages** > **article** > **article.page.ts**

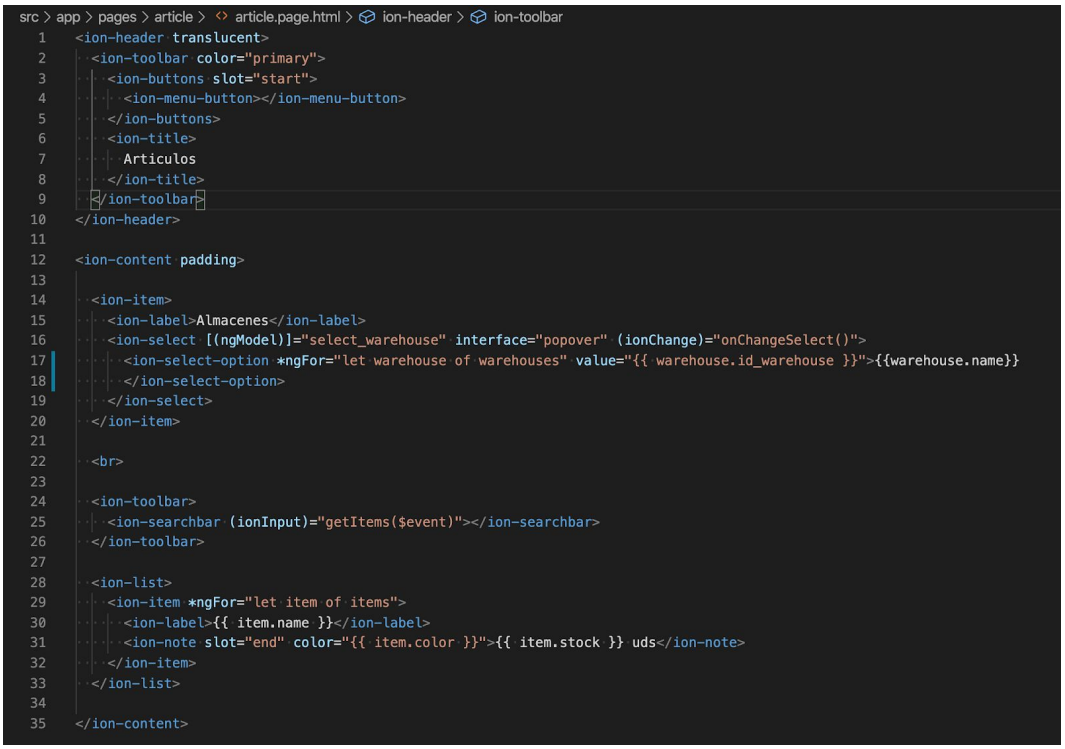


Ilustración 41. Página HTML - Articulos

Tal y como vemos el html es básico y solo contiene un **header**, un **select**, un **search** y un **list** (en el cual aparecerán todos los artículos).

Nos centraremos en el código del SELECT ya que este es el que genera el evento que usaremos como referencia para consumir nuestro servicio.

El evento se genera cuando se selecciona un elemento del SELECT (valga la redundancia) es ***ionChange*** tal y como se muestra en la línea 16 de la ilustración 41. Este evento lo capturaremos mediante la función ***onChangeSelect***, la cual se encuentra en nuestro archivo **article.page.ts** .

Un punto importante a tener en cuenta es el atributo ***[(ngModel)]*** ya que gracias a este podemos interaccionar entre el html y el ts.

Analisemos el archivo article.page.ts, para ellos nos ayudaremos con la ilustración 42.

```
49  onChangeSelect() {
50    this.globalService.presentSpinner();
51    this.reinitialize();
52
53    this.authService.getAllArticles(this.select_warehouse).subscribe(
54      (result) => {
55        console.log(result);
56        if (result["cod"] == "200") {
57          this.articles = result["data"];
58
59          this.articles.forEach((element) => {
60            element["stock"] === "0"
61              ? (element["color"] = "danger")
62              : (element["color"] = "primary");
63
64            this.items_articles.push(element);
65          });
66        } else {
67          this.globalService.presentToast(result["data"]);
68        }
69
70        this.initializeItems();
71        this.globalService.stopSpinner();
72      },
73      (error) => {
74        this.globalService.presentToastWithOptions(
75          this.globalService.MSG_ERROR_UNEXPECTED_ERROR,
76          this.globalService.TYPE_TOAST_ERROR
77        );
78      }
79    );
80  }
```

Ilustración 42. Página HTML - Articulos

Esta función es muy sencilla y lo que hace es consumir el servicio creado anteriormente.

Como se puede ver en la línea 53, le pasamos como parámetro el elemento seleccionado desde la vista, para ello utilizamos el **ngModel** y acto seguido utilizamos un **subscribe** en lugar de un **then**, esto es porque en lugar de devolver una **promesa** lo que devuelve es un **observable.**

Un **observable** se queda a la espera de recibir datos y nosotros quedamos a la espera cuando nos "**suscribimos**". La diferencia a grandes rasgos en relación con las **promesas** es que un **observable** lo que hace es "observar" por si existe algún cambio y se ejecuta cada vez que ese cambio se produce, al contrario que las **promesas** que solo se ejecutan una vez.

Cabe mencionar que un **observable** no se ejecutará hasta que nos hayamos suscrito.

5.2.2.2 TypeScript

TypeScript es un lenguaje de programación libre y de código abierto desarrollado y mantenido por Microsoft. Es una extensión subconjunto de JavaScript por lo que puede ser usado para desarrollar aplicaciones JavaScript en el lado del cliente o del servidor, consiste básicamente en la posibilidad de añadir tipos estáticos y objetos basados en clases para poder comprobar errores en tiempo de compilación a JavaScript en vez de tener que solventarlos

en tiempo de ejecución . En Ionic se utiliza debido a que es utilizado por el framework Angular.

Ilustración 43. Typescript

5.2.2.3 Cordova

El principal objetivo de ionic es generar aplicaciones híbridas que se puedan utilizar en distintos sistemas operativos, pero la pregunta es ¿cómo se consigue empaquetar el código y generar aplicaciones android e ios[27]?, ahí es donde entra Apache Cordova.

Apache Cordova es de código libre y permite generar aplicaciones nativas para plataformas móviles utilizando las principales tecnologías web como son HTML, CSS y JAVASCRIPT.

La forma en la que podemos ver las aplicaciones corriendo en distintos sistemas operativos es porque cordova procesa el código mediante un webview. Un WebView es un componente como un navegador web en el cual podemos ejecutar código html, css y javascript, además de acceder a los periféricos del dispositivo a través de distintos plugins.

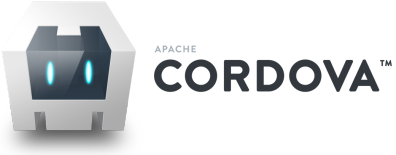


Ilustración 44. Apache cordova

5.2.3 Tecnologías utilizadas en el backend

En el backend nos encontramos con distintas tecnologías, la principal con la que se ha trabajado es PHP, ayudándonos del framework Zend en el que estaba desarrollada la aplicación de partida y utilizando la arquitectura REST. No obstante, Zend dispone de utilidades para la implementación de la

[27] https://ionicframework.com/resources/articles/what-is-apache-cordova

capa REST que han simplificado su desarrollo y facilitarán su mantenimiento futuro.

Como bien sabemos el backend es el encargado de llevar toda nuestra lógica de negocio, la comunicación con la base de datos para tener los datos de forma persistente, etc.

Para tener un esquema claro de todo lo que pasa tras bambalinas enumeramos cada una de las tecnologías.

5.2.3.1 Zend Framework

Zend framework[28] es la tecnología utilizada para realizar la API REST. Zend es un framework de código abierto para desarrollar aplicaciones web, utiliza código 100% orientado a objetos y tiene una estructura de componentes con una baja dependencia entre ellos, lo cual nos permite realizar modificaciones de forma rápida y utilizar los distintos componentes por separados.

Conexión a la base de datos

MySql es el sistema de gestión de base de datos relacional que se utilizó para realizar este TFG. En este punto se explicará cómo se integró Mysql con el proyecto a nivel de código utilizando Zend Framework.

Para realizar una conexión a base de datos desde Zend tenemos que seguir unas pautas y tener claro cuales son los componentes que intervienen. Por una parte nos encontramos con el modelo, del cual explicaremos las funciones más importantes como son el constructor, recuperar datos, crear y editar.

Empecemos con el constructor, tal y como vemos en la ilustración 45 instanciamos una clase DbTable, esta clase contiene el nombre exacto de la tabla en la base de datos y contamos con tantas clases DbTables como referencias queremos hacer a la base de datos.

```php
<?php

class API_Model_Article extends My_Model_API
{
  public function __construct()
  {
    $this->dbTable = new My_DbTable_Article();
  }

```

Ilustración 45. Constructor Modelo Zend

Este modelo por sí solo no funcionará ya que como hemos dicho necesita del DbTable. Este conector lo encontramos en la clase **library/My/DbTable/Article.php.**

[28] https://framework.zend.com/about

```php
<?php

class My_DbTable_Article extends Zend_Db_Table_Abstract
{
    protected  $_name = "article";
}
```

Ilustración 46. Db table - Zend

Como podemos ver en la ilustración 46, **$name** tiene el mismo nombre de la tabla en la base de datos, que en este caso es "**article**".

Para recuperar datos podemos hacerlo como se muestra en la ilustración 47. Aquí podemos personalizar nuestra consulta como queramos desde joins, operadores, alias, cláusulas, condiciones y todo lo que se nos ocurra que podamos hacer en una consulta normal con mysql.

```php
public function getAllArticle()
{
  $select = $this->dbTable->select()->setIntegrityCheck( false );
  $select->from("article");

  return $select->query()->fetchAll();
}

public function getArticleById($id_article)
{
  $select = $this->dbTable->select()->setIntegrityCheck( false );
  $select->from("article");
  $select->where("article =?", $id_article);

  return $select->query()->fetchAll();
}
```

Ilustración 47. Consultar a la base de datos

Otras de las funciones que no pueden faltar en un modelo son las de crear y editar, como vemos en la ilustración 48.

```php
public function addArticle(My_Object_Article $article)
{
  return $this->dbTable->insert($article->toArray());
}

public function editArticle($id_article , My_Object_Article $article)
{
  return $this->dbTable->update($article->toArray(), "id_article = $id_article");
}
```

Ilustración 48. Crear y editar en la base de datos

En ambos casos se le pasa una instancia de la clase My_Object_Article, la cual contiene todos los getter y setter de cada uno de los atributos de la tabla, p.e. name, price, etc. Esta clase la encontramos en **library/My/Object/Article.php**

Validar token JWT

Una de las partes más importantes del back-end es la validación del token de acceso, este token nos permitirá saber qué usuario está haciendo la petición y el alcance que este tiene de los recursos que solicita.

Este token será validado antes de llegar al endpoint por lo que se declara en la función init. Si nos fijamos en la ilustración 49 vemos que el controlador extiende de la clase **Zend_Rest_Controller** gracias a la cual podremos usar los verbos http tales como POST, GET, PUT, DELETE, etc.

```
class API_ArticleController extends Zend_Rest_Controller
{
    public function init()
    {
      header('Content-Type: application/json');
      $this->getHelper('Layout')->disableLayout();
      $this->getHelper('ViewRenderer')->setNoRender();

      My_Validate_API::validate($this);
    }
```

Ilustración 49. Validar token desde Zend

La clase **My_Validate_API** contiene todas las funciones necesarias para validar el token pasando el contexto de la petición, esta clase es de mucha utilidad ya que a partir del JWT nos da una instancia con los datos más relevantes del usuario p.e. username, id, role, company, etc. Otro de los beneficios de esta clase es que todos los cambios relacionados con la validación de credenciales la podemos realizar tocando solo en esta clase, modularizado nuestro código para tener así un código más sólido.

Configurar variables de entorno

Para configurar variables de entorno tenemos que irnos al archivo de configuración **application.ini** el cual se encuentra en la ruta **application/api/config/application.ini**. En este archivo podemos definir cuáles serán las variables de testing,producción, acceso a base de datos, etc. Tal y como vemos en la ilustración 50.

```
; database
resources.db.adapter = "pdo_mysql"
resources.db.params.host = "localhost"
resources.db.params.username = "root"
resources.db.params.password = "tupassword"
resources.db.params.dbname = "stock"
resources.db.isDefaultTableAdapter = true
```

Ilustración 50. Configuración de variables de entorno - Zend

1. Apache

Apache es el servidor web utilizado para correr el backend. Apache es un servidor web HTTP de código abierto para distintas plataformas en las que se encuentran Unix, Windows, MacOS, etc.

El servidor de Apache es desarrollado y mantenido por la comunidad de usuario bajo la supervisión de la Apache Software Foundation.

Algunos de sus puntos a favor es que apache es muy configurable, por contra es criticado por no tener una interfaz para ayudar a dicha configuración. Este problema lo solucionamos con las herramientas de configuración Webmin y Virtualmin.

2. Linux

A estas alturas y tanto si trabajamos en el mundo de la tecnología como si no lo más seguro es que hayamos oído de Linux.

No podemos hablar de linux sin hablar de GNU por lo que nos remontaremos a los años 80 cuando Richard Stallman estaba creando un movimiento de software libre y a su vez un sistema operativo completamente libre al que llamó GNU. Pasaron los años y Richard Stallman aún no tenía el kernel, el cual si hacemos un símil el kernel sería el motor de un coche.

Para la década de los noventa Linus torvalds estaba creando un proyecto propio al cual llamó Linux (que viene de su nombre Linus con la X de UNIX).

A Pesar de que en la jerga cotidiana se utiliza Linux como un sistema operativo la verdad es que Linux es un Kernel el cual se puede utilizar en distintos entornos.

3. JWT

Json Web Token es un estándar basado en JSON para la creación de tokens de acceso que nos permitan identificar y limitar el alcance de los usuarios.

Un token JWT se ve de la siguiente forma.

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.SflKxwRJSMeKKF2QT4fwpMeJf36POk6yJV_adQssw5c

Este token se conforma de tres partes :

- **Header**: Encabezado donde se indica el algoritmo y el tipo de token, p.e. HS256 y JWT

- **Payload** : Este sería el cuerpo del token el cual tendría la información relativa al usuario. Como punto importante este payload no debería tener información sensible, ya que como se verá a continuación esta información está codificada, mas no encriptada.

- **Signatura**: Esta sería la firma que nos permita saber que el token no ha sido manipulado por nadie.

Si utilizamos la herramienta que nos proporciona jwt.io[29] vemos que si tenemos un token y lo copiamos en el debugger este podrá descodificar el contenido del header y el payload, incluso nos permitirá verificar la signature en caso que la tengamos. tal y como se muestra en la ilustración 51.

[29] https://jwt.io/

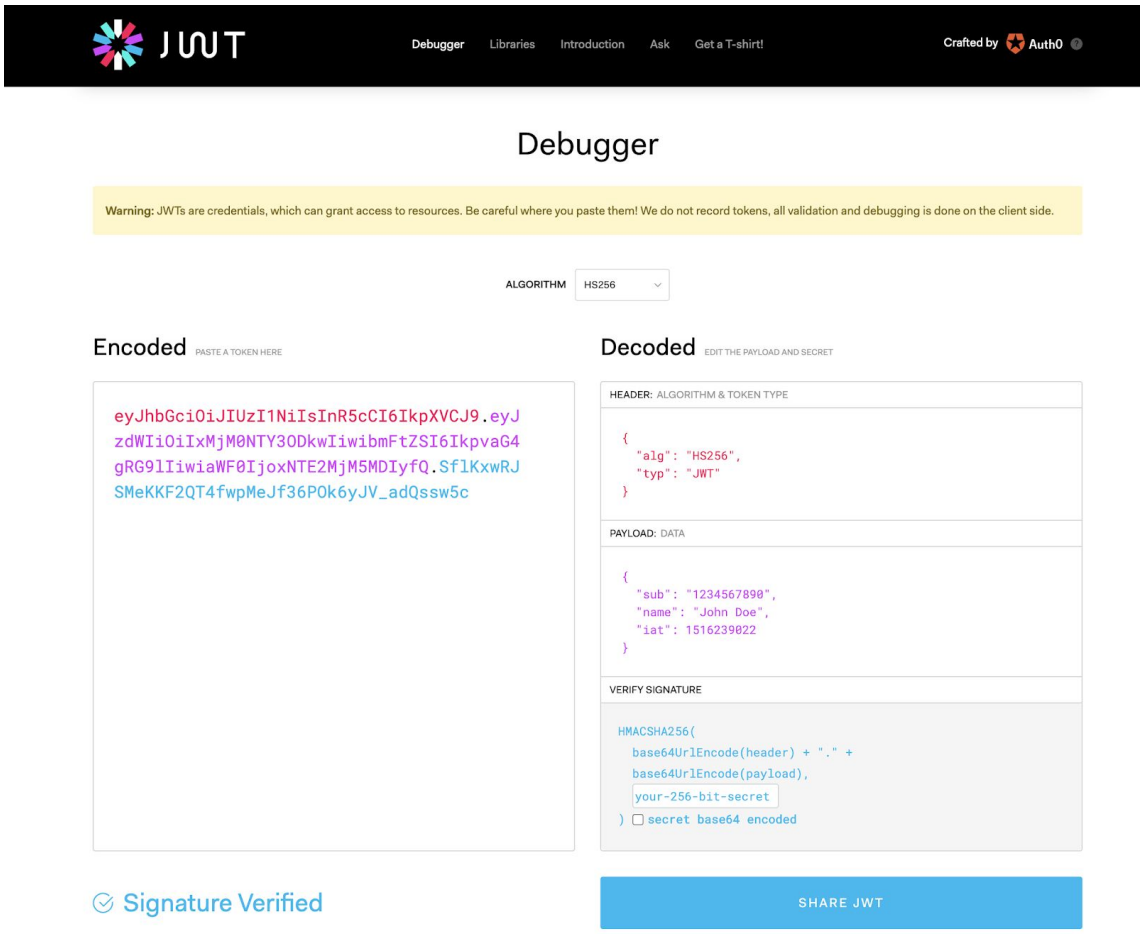


Ilustración 51. Debugger JWT

Es aquí donde surge la pregunta ¿si podemos ver el contenido del token, cúal es el beneficio que nos da jwt?.

A grandes rasgos el beneficio que nos da jwt es la certeza que el token no ha sido modificado por nadie que haya podido interceptar, añadiendo privilegios o algún dato malicioso.

La forma de utilizarlo es muy simple ya que jwt.io nos proporciona múltiples librerías para múltiples lenguajes, lo cual hace que la integración con nuestros proyectos se realicen de forma rápida.

4. Firebase

Firebase es una plataforma para el desarrollo de aplicaciones web y móviles desarrollada por Google el 2014.[30]

Firebase cuenta con una serie de servicios muy interesantes como son Firebase analytics, Firebase Cloud Messaging (FCM)[31], Firebase Auth, Realtime Database, entre otros. Para este TFG se ha utilizado el servicio FCM.

Para integrarlo en el TFG se han tenido que seguir una serie pasos, lo primero que tenemos que hacer es ir a la consola de Firebase y crear un proyecto, tal y como se muestra en la ilustración 52.

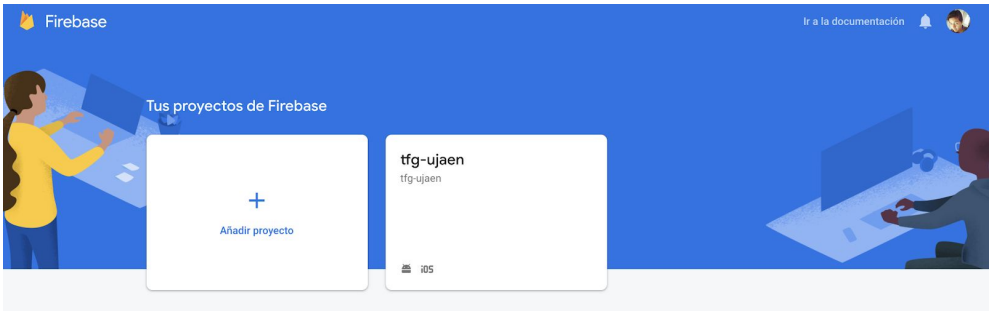


Ilustración 52. Consola Firebase

Cuando estemos dentro del proyecto que acabamos de crear pulsaremos en “Configuración del proyecto” y acto seguido pulsaremos en añadir aplicación, la cual nos preguntará si queremos añadir una aplicación Android, iOS o web. Ya que el TFG es una aplicación móvil multiplataforma crearemos una para Android e iOS, tal y como se muestra en la ilustración 53.

[30] https://firebase.google.com/?hl=es
[31] https://firebase.google.com/docs/cloud-messaging

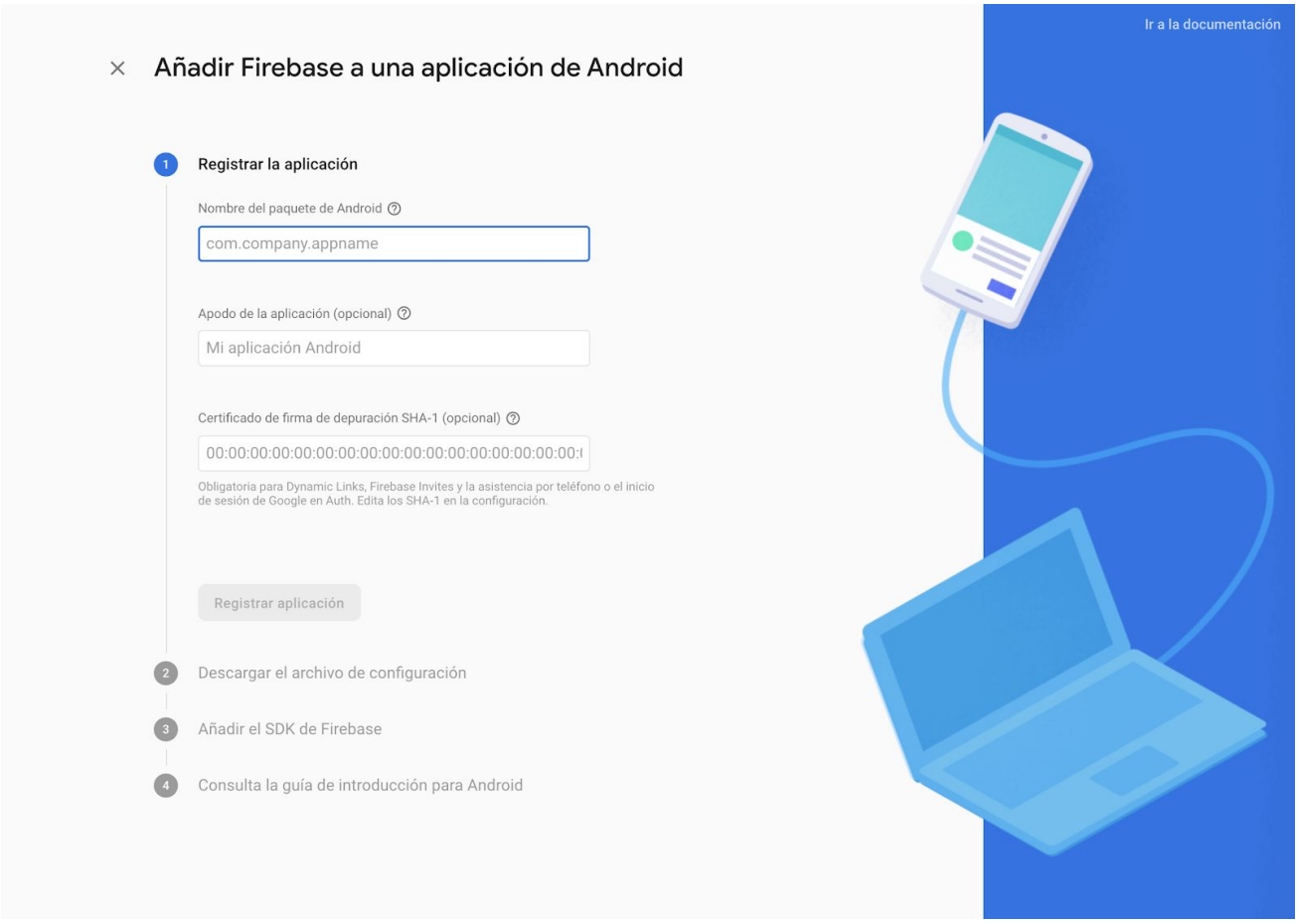


Ilustración 53. Añadir aplicación a Firebase

Aquí tenemos que indicar el nombre del paquete que tendrá nuestra aplicación, este nombre es importante ya que cuando empaquetamos nuestra aplicación tiene que tener el mismo nombre, más adelante se mostrará cómo.

Cuando hayamos añadido Firebase a nuestra aplicación nos dará la opción de descargar un archivo llamado “google.services.json” en el caso de Android y “GoogleService-Info.plist” en el caso de iOS, descargamos estos archivos y los copiaremos en la raíz de nuestra aplicación en Ionic.

Con esto hemos creado el proyecto en Firebase pero ahora tenemos que integrarlo en el proyecto, para ello nos vamos a la documentación de ionic, buscamos el plugin FCM[32] e instalamos los plugins tal y como nos indica la documentación.

Cuando hayamos instalado el plugin tenemos que importarlo en el proyecto, para ello nos vamos al archivo **app.module.ts** e importamos el plugin y lo incluimos en el array **providers** como podemos ver en las ilustraciones 54 y 55.

[32] https://ionicframework.com/docs/native/fcm

```
20  //Firebase
21  import { FCM } from "@ionic-native/fcm/ngx";
22
```

Ilustración 54. Importar plugin FCM

```
49    providers: [
50      FCM,
51      StatusBar,
52      SplashScreen,
53      { provide: RouteReuseStrategy, useClass: IonicRou
54      AuthService,
55      BarcodeScanner,
56      Base64ToGallery,
57    ],
```

Ilustración 55. Añadirlo al provider

El siguiente paso es recuperar el token y enviarlo al backend para que sea almacenado y podamos utilizarlo posteriormente para enviar las notificaciones push.

Recuperaremos el token en la página principal, HOME, tal y como vemos en la ilustración 56. Para poder utilizar el plugin tenemos que declararlo en el constructor y acto seguido recuperar el token desde la función ngOnInit, esta es una función que se ejecuta después del constructor y tiene que ver con el ciclo de vida del componente.

```
34
35  ngOnInit() {
36    this.fcm.getToken().then((token) => {
37      this.authService.registerToken(token);
38    });
39  }
40
```

Ilustración 56. Registrar token FCM en backend

Con estos pasos ya tenemos prácticamente lista la integración de Firebase con nuestra aplicación, solo faltaría cambiar el nombre del paquete y ponerle el mismo que pusimos en la consola de Firebase al crear el proyecto.

Para cambiar el nombre nos iremos al archivo **config.xml** ubicado en la raíz del proyecto. En este archivo escribiremos en el atributo **ID** de la etiqueta **widget** el nombre del paquete, que en este caso es **com.tfg.ujaen** como podemos ver en la ilustración 57.

Ilustración 57. Nombre del paquete

6. Pruebas

En este apartado se comentarán los resultados de las pruebas de la aplicación de los distintos casos de uso.

6.1 Iniciar sesión

Para iniciar sesión tenemos que introducir nuestras credenciales de acceso, estas credenciales se enviarán al backend y se comprobarán, en caso que sean correctas el backend generará y enviará el jwt al frontend para futuras solicitudes de recursos.

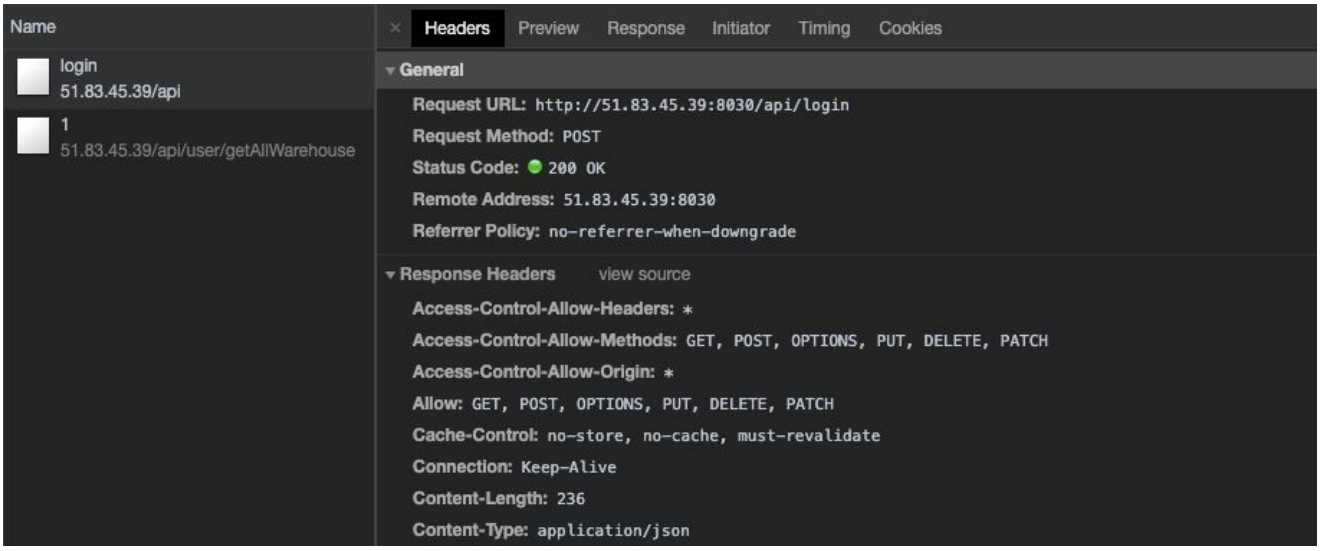


Ilustración 58. Petición iniciar sesión

Como podemos ver en la ilustración 58 hacemos una petición POST al backend para iniciar sesión la cual nos devuelve un código 200 ya que las credenciales ha sido validadas y el respectivo token de autenticación (tal y como se muestra en la ilustración 59) que usaremos para realizar las siguientes pruebas.

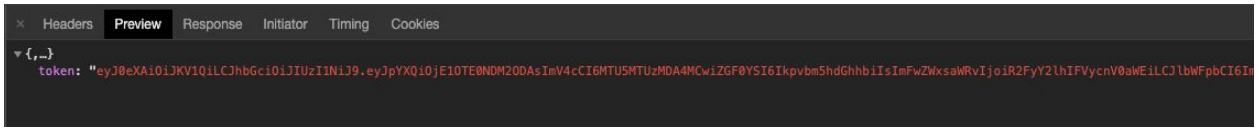


Ilustración 59. Respuesta token de autenticación

6.2 Recuperar almacenes del usuario

En este prueba comprobaremos que el usuario cuando inicia sesión recupera automáticamente todos los almacenes que tiene asignado. Estos almacenes aparecerán en un desplegable.

Si analizamos la ilustración 60, vemos que justo después de iniciar sesión se hace otra petición automáticamente, esta petición es para recuperar los almacenes que como vemos la petición es correcta y nos han llegado todos los almacenes del backend, que en este caso son dos.

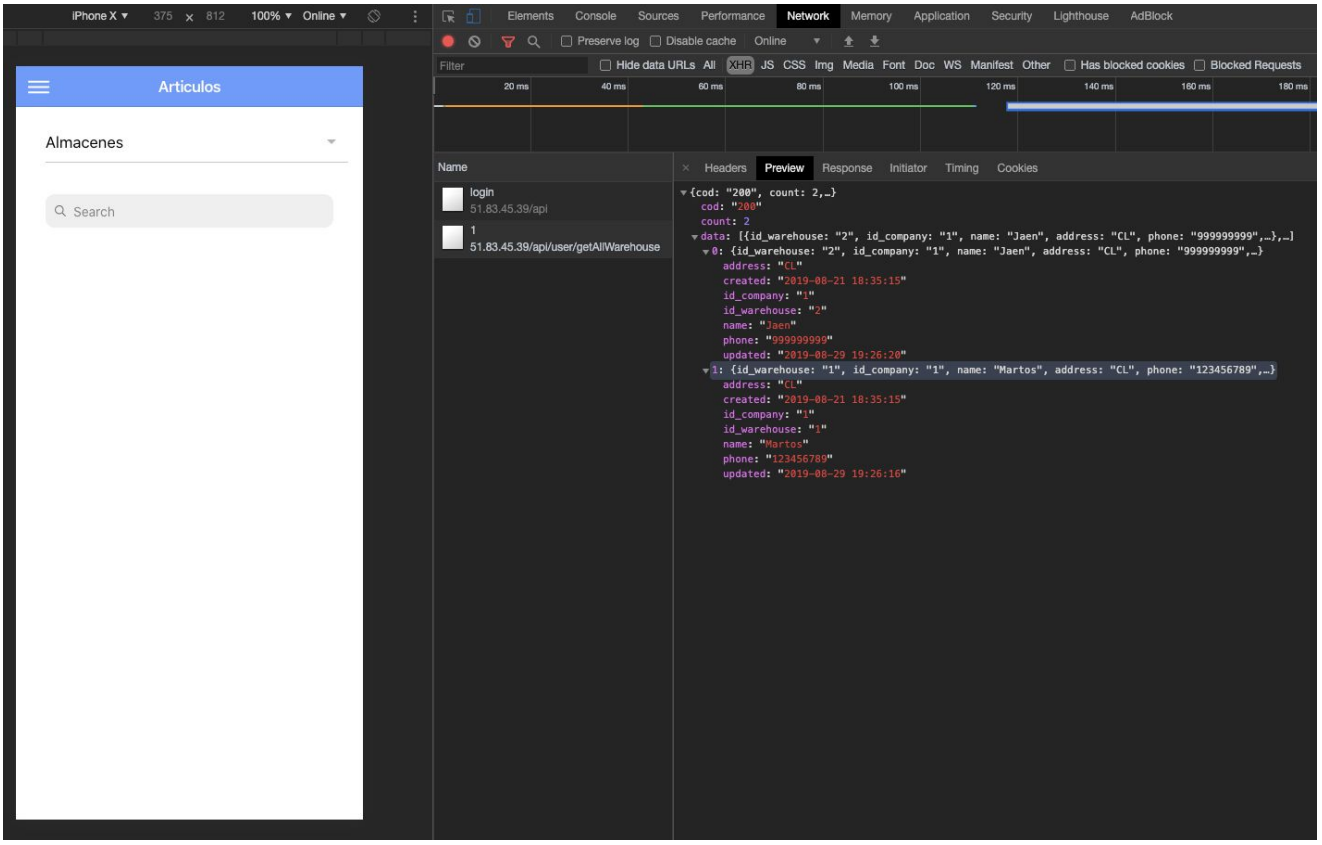


Ilustración 60. Recuperar almacenes

6.3 Recuperar artículos y stock de un almacén

Al igual que en las pruebas anteriores nos ayudaremos del DevTools[33] de Chrome.

Para realizar esta prueba seleccionaremos uno de nuestros almacenes, por ejemplo JAEN. Esto realizará una petición al backend y solicitará todos los artículos de este almacén

Como podemos ver en la ilustración 61, la petición se ha realizado correctamente y hemos podido pintar todos los artículos en nuestra aplicación, indicando por cada uno de ellos su stock actual.

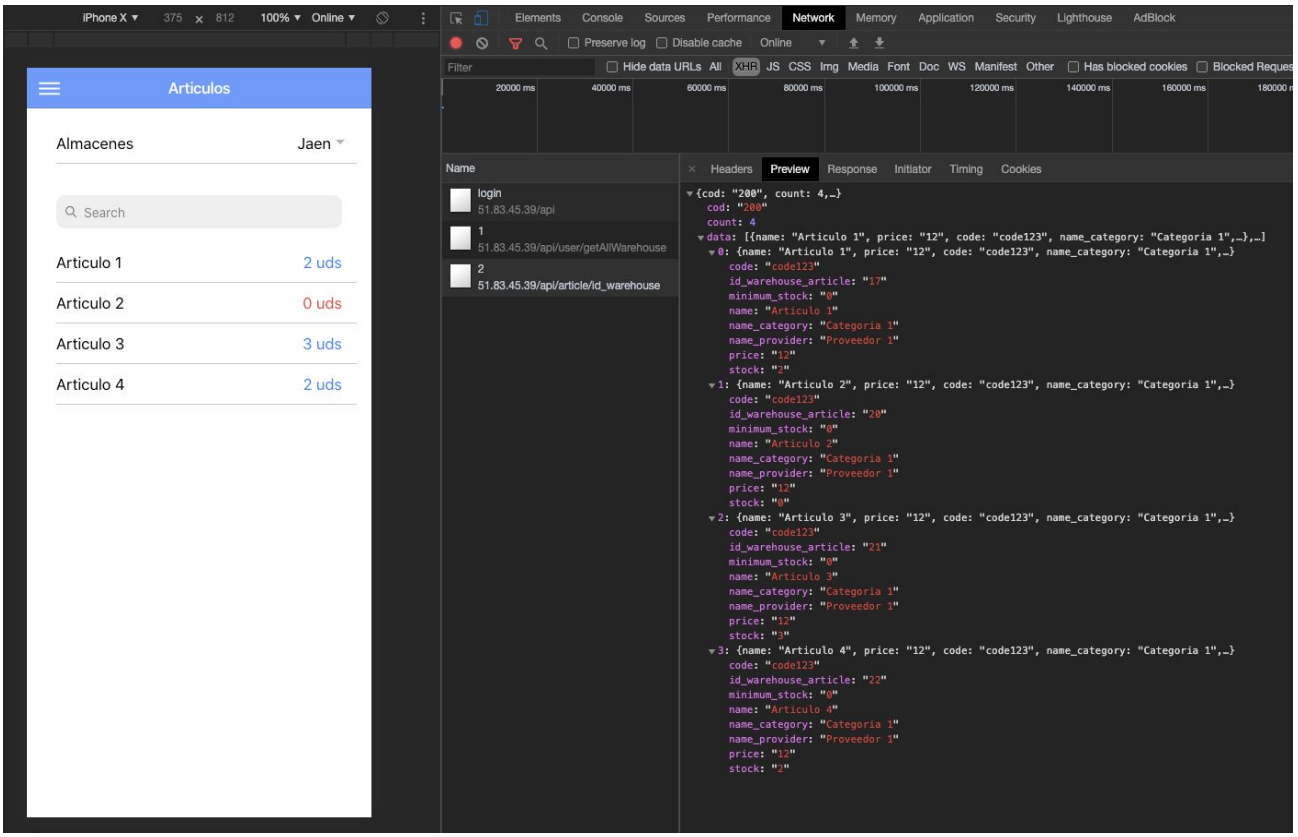


Ilustración 61. Recuperar artículos de un almacén

[33] https://developers.google.com/web/tools/chrome-devtools?hl=es

6.4 Recuperar todas nuestras salidas

Para realizar esta prueba tenemos que ir al apartado SALIDAS, para ello tenemos que dirigirnos al menú lateral de la izquierda y seleccionar SALIDAS.

Esta realizará una petición al backend y solicitará todas las salidas que hayamos realizado.

Tal y como vemos en la ilustración 62, la petición se ha resuelto correctamente y nos ha devuelto todas las salidas con los respectivos artículos de cada una de ellas, los cuales se han pintado en la aplicación.

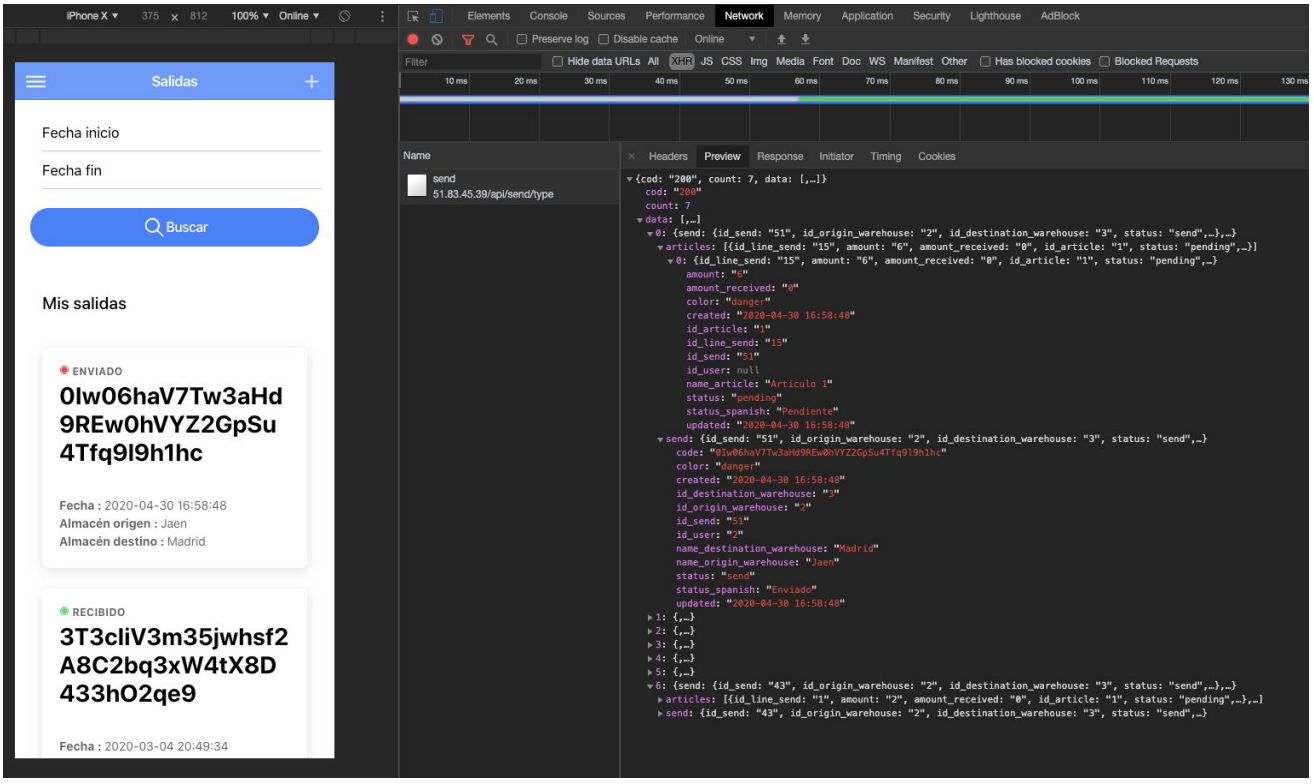


Ilustración 62. Recuperar todas las salidas

6.5 Generar salida de artículos de un almacén a otro

Para generar una salida tenemos que pulsar en el botón **+** ubicado en la parte superior derecha de la página **SALIDAS**.

Para esta prueba realizaremos una salida del almacén de JAEN al almacén de MADRID y enviaremos dos artículos, tal y como se muestra en la ilustración 63.

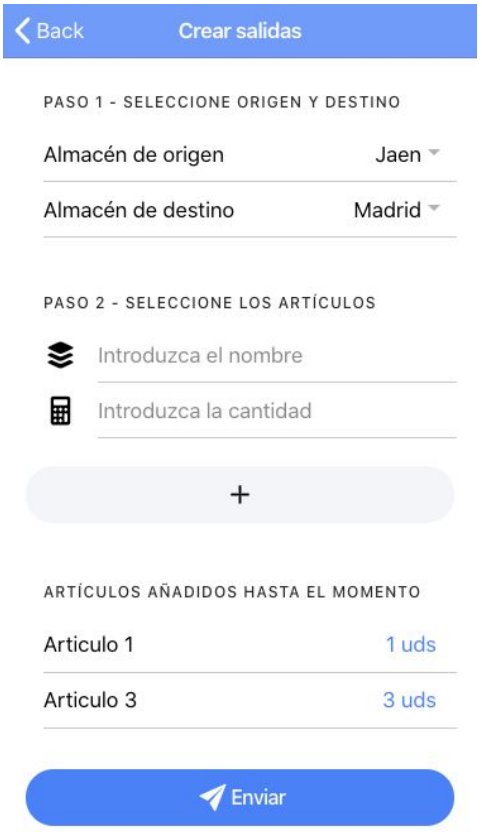


Ilustración 63. Generar salida de artículos - VISTA

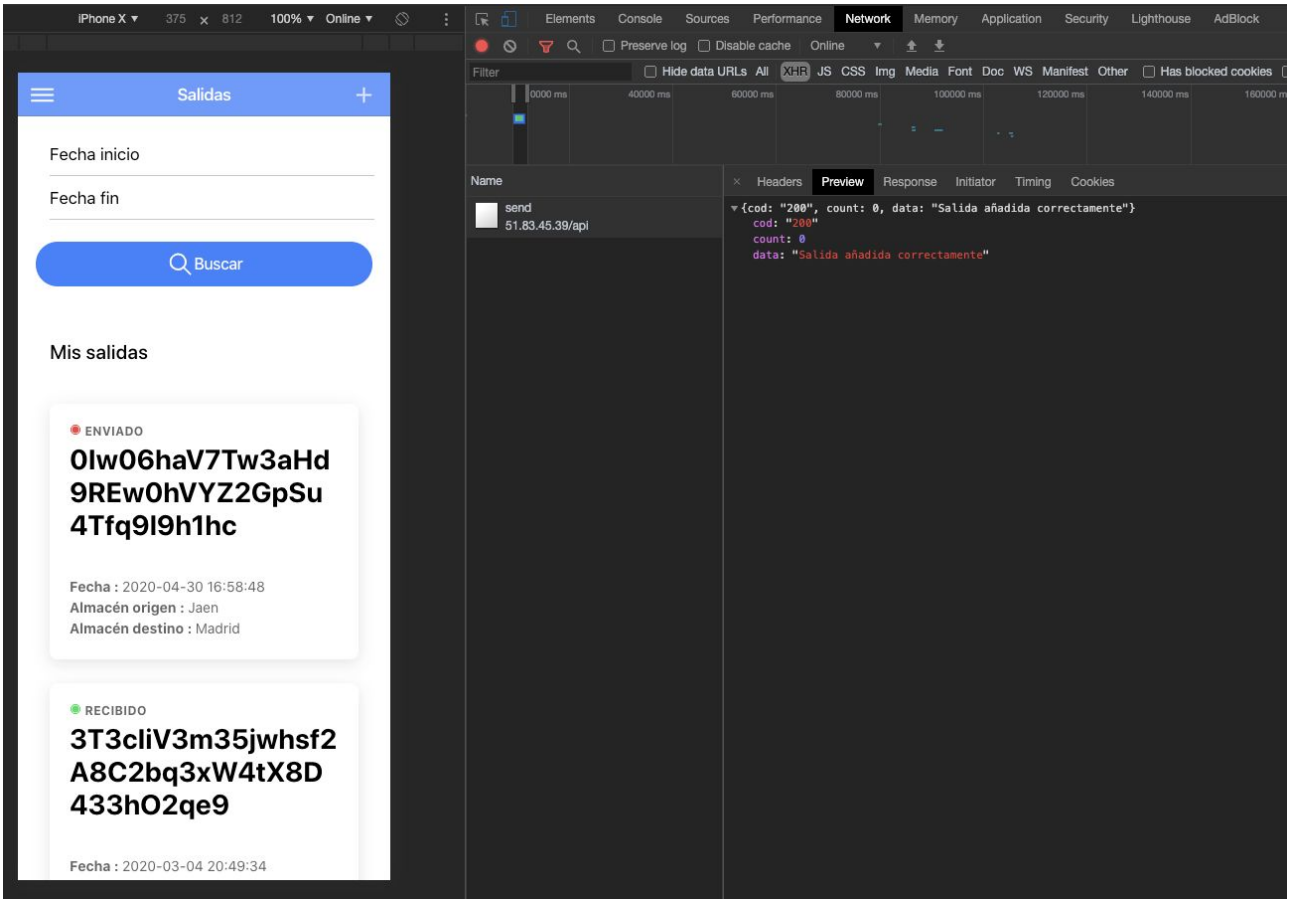


Ilustración 64. Petición generar salida

Como podemos ver en la ilustración 64, la salida se ha realizado correctamente y en la página **SALIDAS** aparece el **CARD** con la instancia de la salida creada.

A primera vista podemos ver que el **CARD** tiene un código alfanumérico el cual es único por salida, también podemos ver el estado de la salida (en este caso **ENVIADO**) la fecha, el origen y destino.

6.6 Recuperar todas nuestras entradas

Para recuperar las entradas, al igual que las salidas lo que tenemos que hacer es seleccionar ENTRADAS en el menú lateral de la izquierda.

Tal y como vemos en la ilustración 65, la petición al backend se ha realizado correctamente y nos ha devuelto todas las entradas que se han realizado en todos nuestros almacenes.

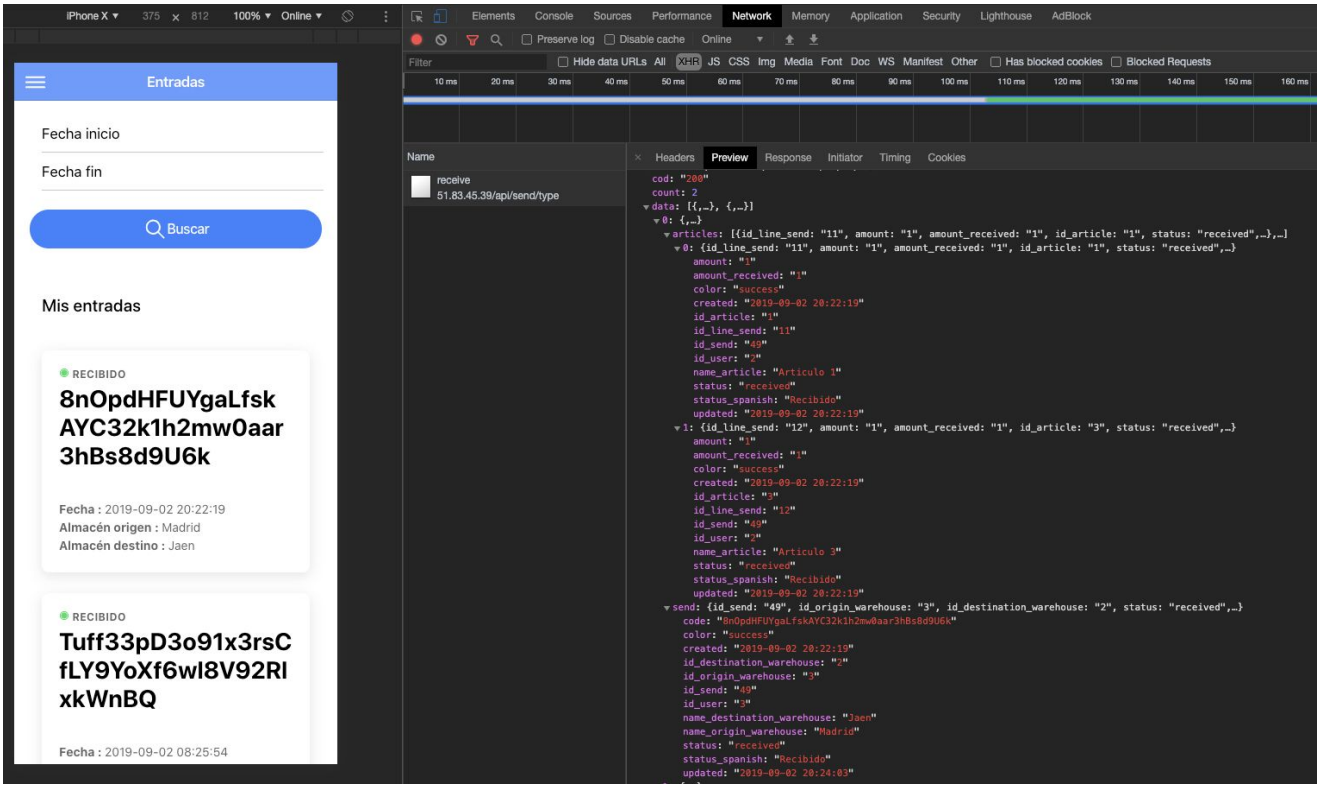


Ilustración 65. Recuperar todas las entradas

6.7 Confirmar entrada de artículos

Para confirmar la entrada de artículos lo primero que tenemos que hacer es buscar el CARD que se corresponda con la entrada, luego pulsaremos sobre el mismo y nos mostrará una página con todos los artículos que lo conforman.

En esta vista tenemos la posibilidad de indicar si los artículos han llegado correctamente o no, tal y como se muestra en la ilustración 66.

Ilustración 66. Confirmar entrada - VISTA

Al pulsar en guardar se realizará una petición al backend y este comprobará los artículos, el estado del envío y actualizará los distintos parámetros, como son el estado (que pasará a RECIBIDO) y el incremento del stock de los artículos previamente confirmados.

En la ilustración 67 podemos ver que la petición se ha realizado correctamente y el stock del CARD ha cambiado.

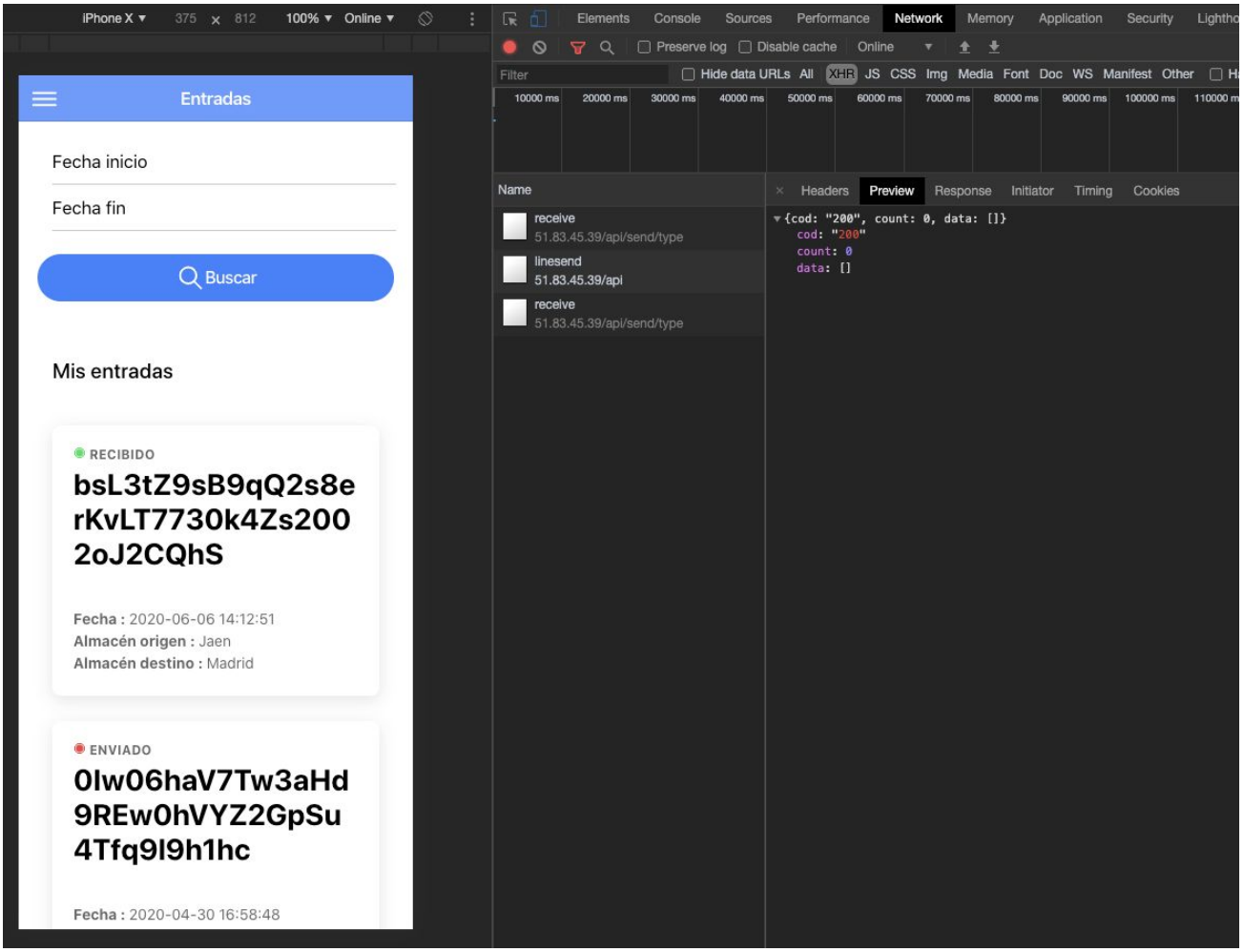


Ilustración 67. Respuesta confirmar entrada

6.8 Ver detalles de cada salida y entrada

Para ver los detalles tanto de las salidas como de las entradas el procedimiento es el mismo, con la diferencia que cada uno se encuentra en su respectivo apartado. Para ver los detalles de una entrada iremos al apartado ENTRADA y pulsaremos sobre una entrada, esto nos mostrará todos los artículos que conforman la entrada, además del origen, destino, fecha y código único, tal y como se muestra en la ilustración 68.

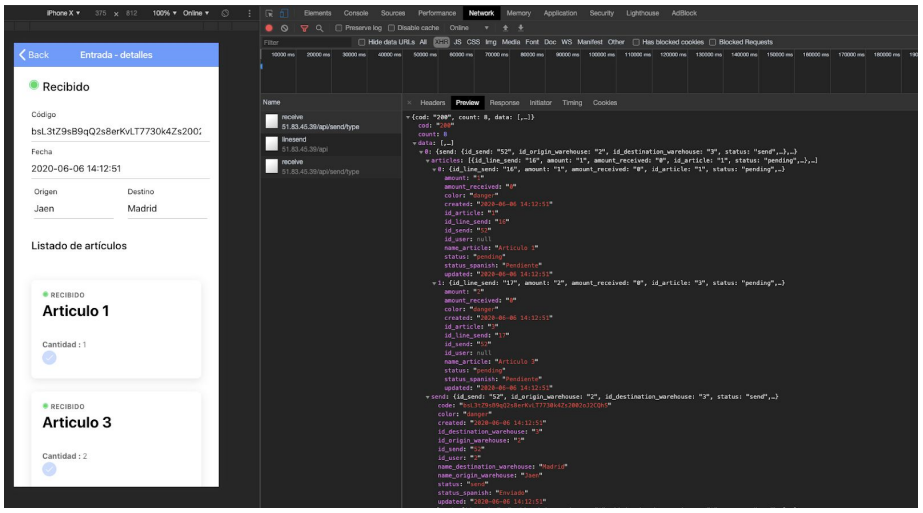


Ilustración 68. Detalles de una entrada

Por contra si queremos ver los detalles de una salida realizaremos los mismos pasos pero desde el apartado SALIDA, ilustración 69.

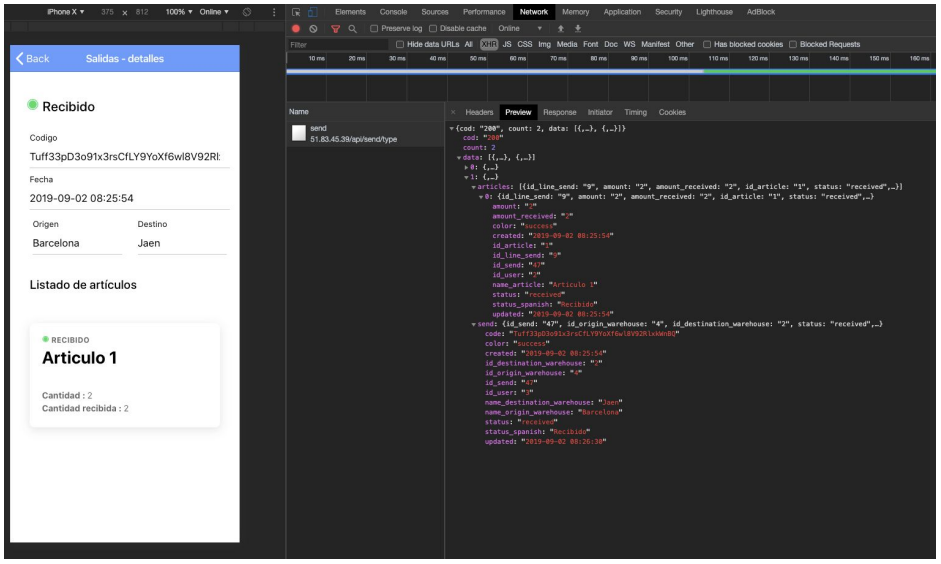


Ilustración 69. Detalles de una salida

6.9 Filtrar las salidas y entradas por rango de fecha

Para realizar esta prueba nos iremos al apartado SALIDA o ENTRADA y seleccionaremos la fecha de inicio y fecha final en los buscaremos ubicados al principio de la página. Cuando hayamos seleccionado el rango de fechas pulsaremos en buscar y este evento enviará una petición al backend para recuperar las SALIDAS o ENTRADAS realizadas en el rango de fecha seleccionado.

Como vemos en la ilustración 70, la petición se ha realizado correctamente y nos ha devuelto cinco salidas, cada una de ellas con sus respectivos artículos.

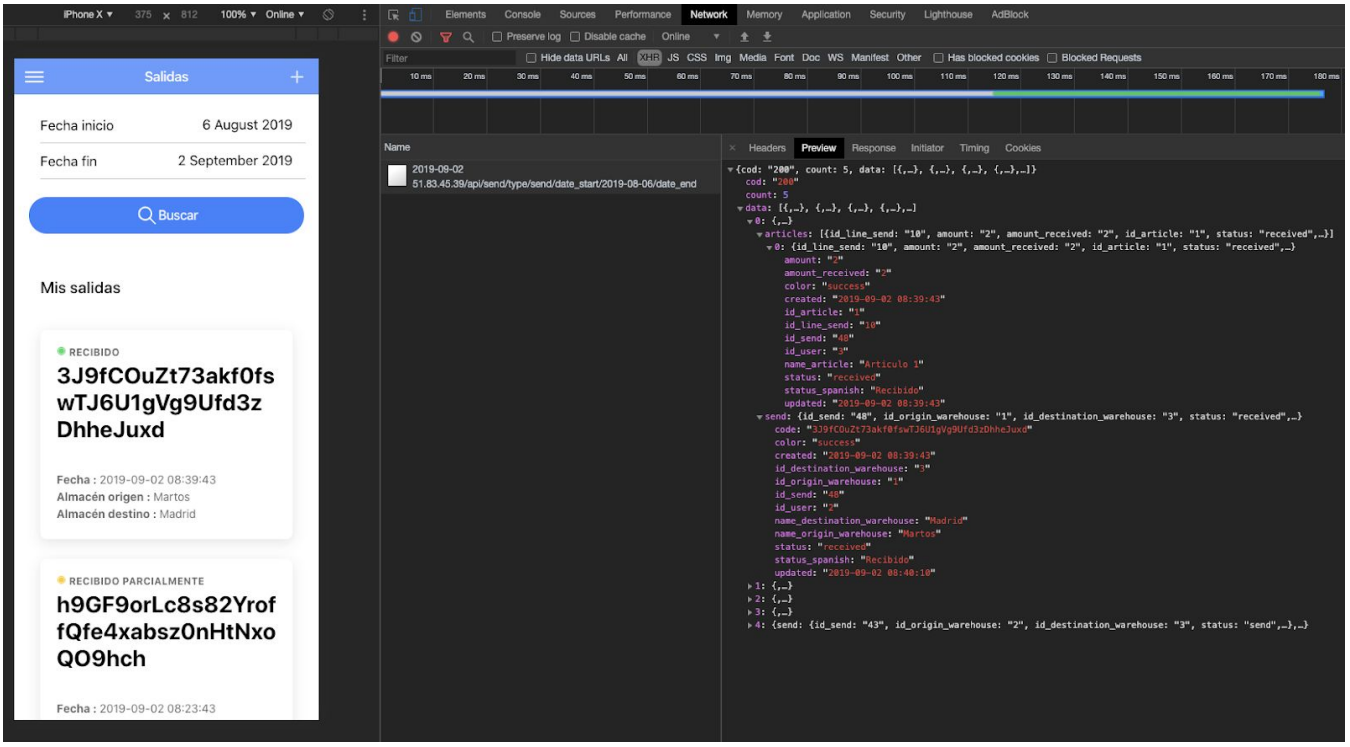


Ilustración 70. Filtrar salida y entrada por rango de fecha

6.10 Enviar notificaciones con Firebase

Para enviar las notificaciones lo podemos hacer de dos formas, la primera es usando la consola de Firebase, para ello iremos al apartado Cloud Messaging y pulsaremos con el mismo, esto no abrirá un formulario como podemos ver en la ilustración 71.

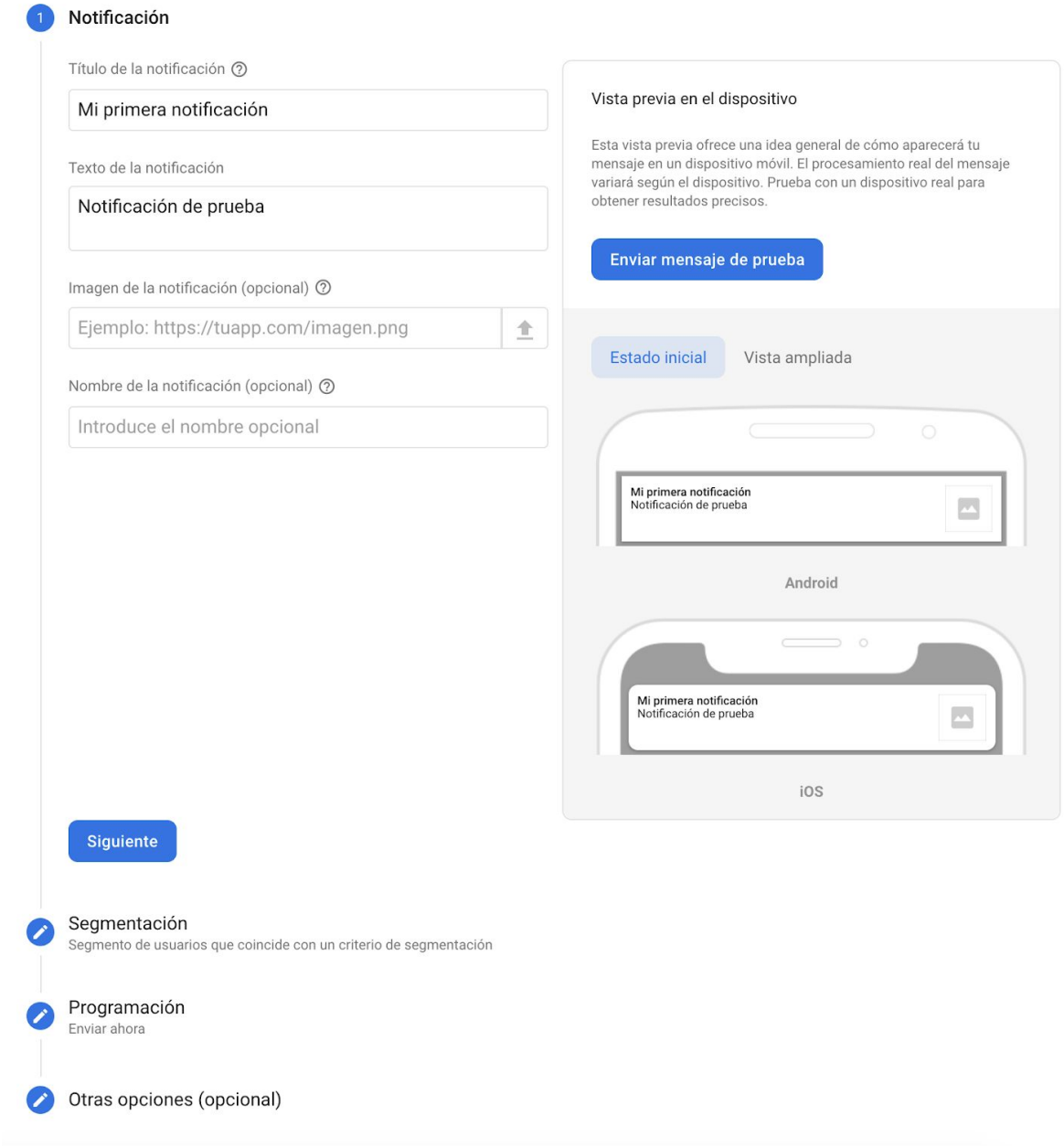


Ilustración 71. Enviar notificación desde consola de Firebase

Las partes a tener en cuenta son la primera y la segunda, la primera es donde tenemos que indicar el título de la notificación y la descripción y la segunda es donde tenemos que elegir la segmentación, es decir a qué clientes queremos enviar la notificación, como podemos ver en la ilustración 72 firebase nos da la opción de enviar una notificación a los dispositivos Android e iOS.

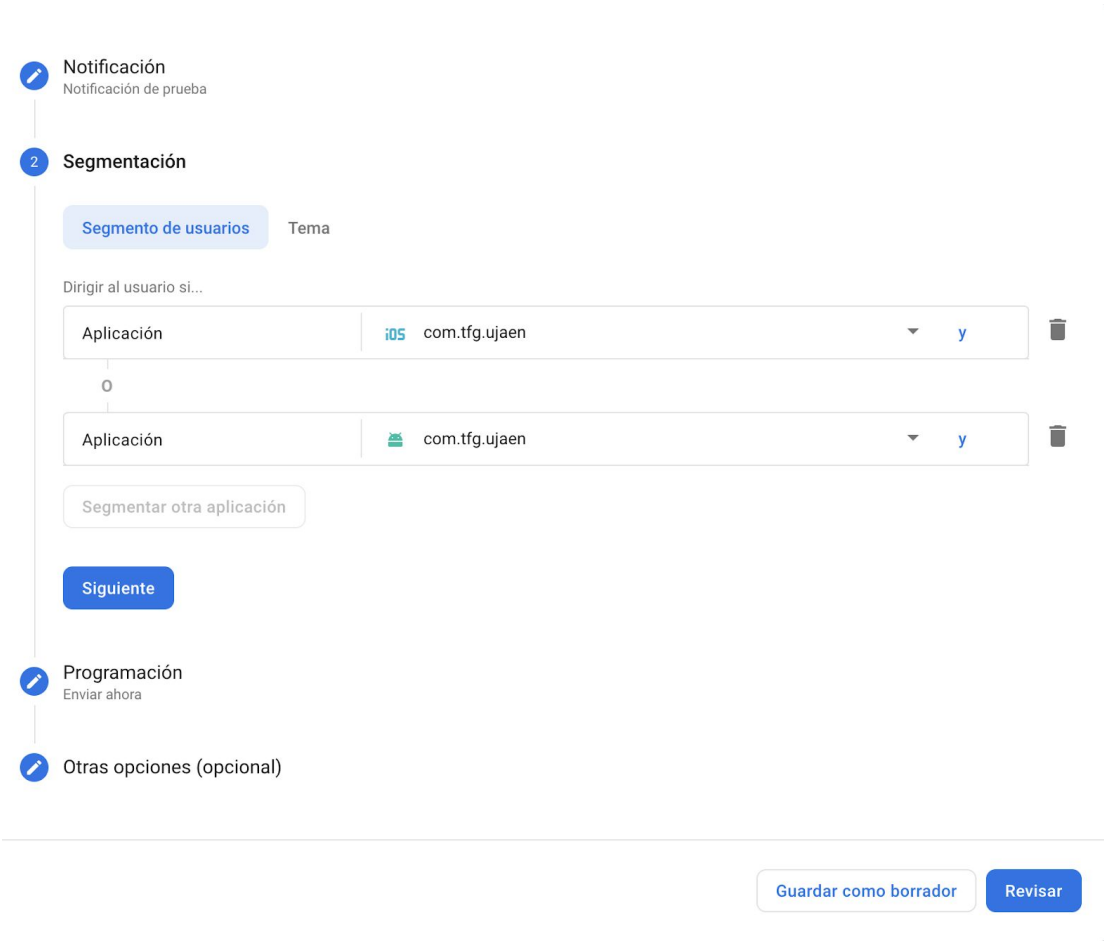


Ilustración 72. Segmentación de la notificación con firebase

Por último se pulsará en revisar y se enviará la notificación. Si volvemos al panel FCM podemos ver que la notificación se ha enviado correctamente, ilustración 73.

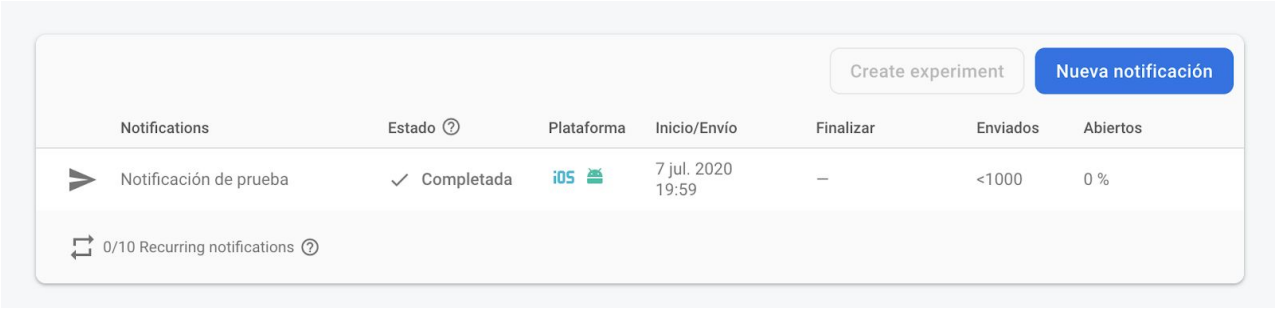


Ilustración 73. Notificación enviada con firebase

La segunda opción es enviar la notificación mediante el protocolo http, para ello utilizaremos postman. Postman es un cliente el cual nos permite realizar peticiones http de distintos tipos GET, POST, PUT, etc.

Tal y como nos dice la documentación de Firebase[34] la petición que tenemos que hacer es con el verbo POST, en ella tenemos que indicar la url que nos proporciona firebase e indicaremos la clave de nuestra aplicación mediante la cabecera **Authorization** y el **Content-Type** de tipo **JSON**.

El contenido de la notificación la enviaremos en el body de tipo raw mediante un json, tal y como podemos ver en la ilustración 74.

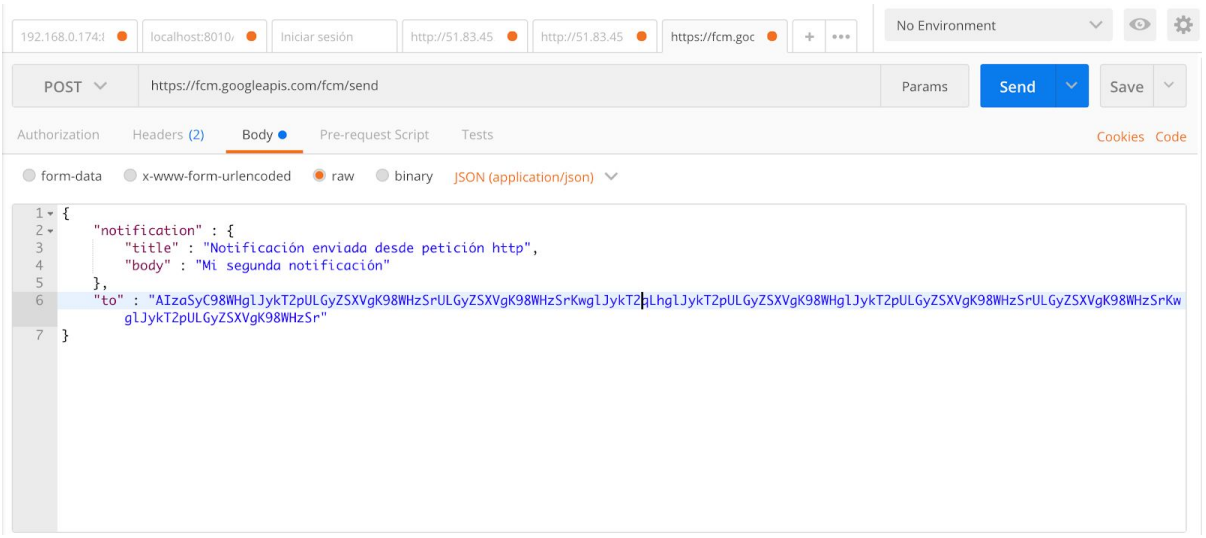


Ilustración 74. Envío de notificaciones mediante HTTP

Después de realizar todas las pruebas podemos comprobar que los resultados han sido positivos y no se han detectado errores relevantes durante la realización de las mismas.

6.11 Probando swagger

Utilizar swagger es muy simple y para una mejor comprensión se explicará brevemente cómo consultar datos de un endpoint.

Lo primero que tenemos que hacer para consumir un endpoint es introducir nuestro token de acceso ya que si no lo hacemos todas las peticiones serán rechazadas tal y como se muestra en la ilustración 75.

[34] https://firebase.google.com/docs/cloud-messaging/http-server-ref?hl=es

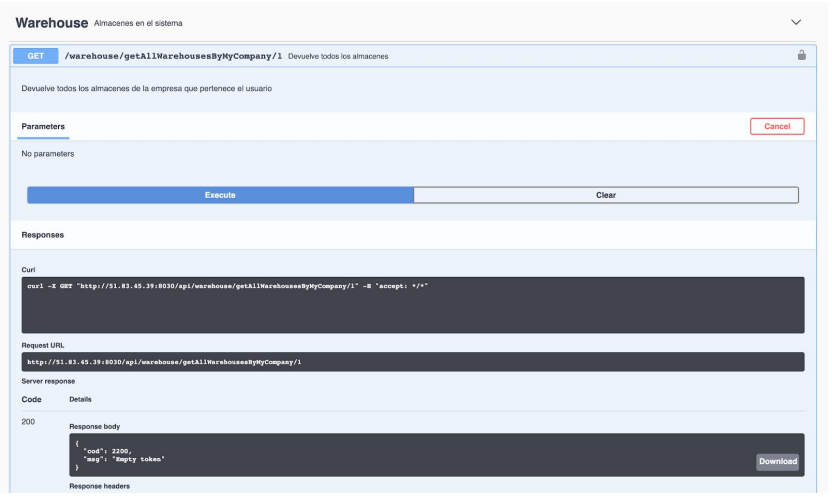


Ilustración 75. Probando swagger

Para introducir el token de acceso lo podemos hacer directamente desde cualquier endpoint pulsando en el candado que está en la parte superior derecha o pulsando en “Authorize” ubicado al inicio de la documentación, en cualquier caso mostrará un modal en el que tenemos que copiar el token. Cuando hayamos introducido un token correcto si volvemos a hacer la petición anterior vemos que esta vez sí nos devuelve los datos relativos al endpoint que en este caso son los almacenes a los cuales tenemos acceso, tal y como vemos en la ilustración 76.

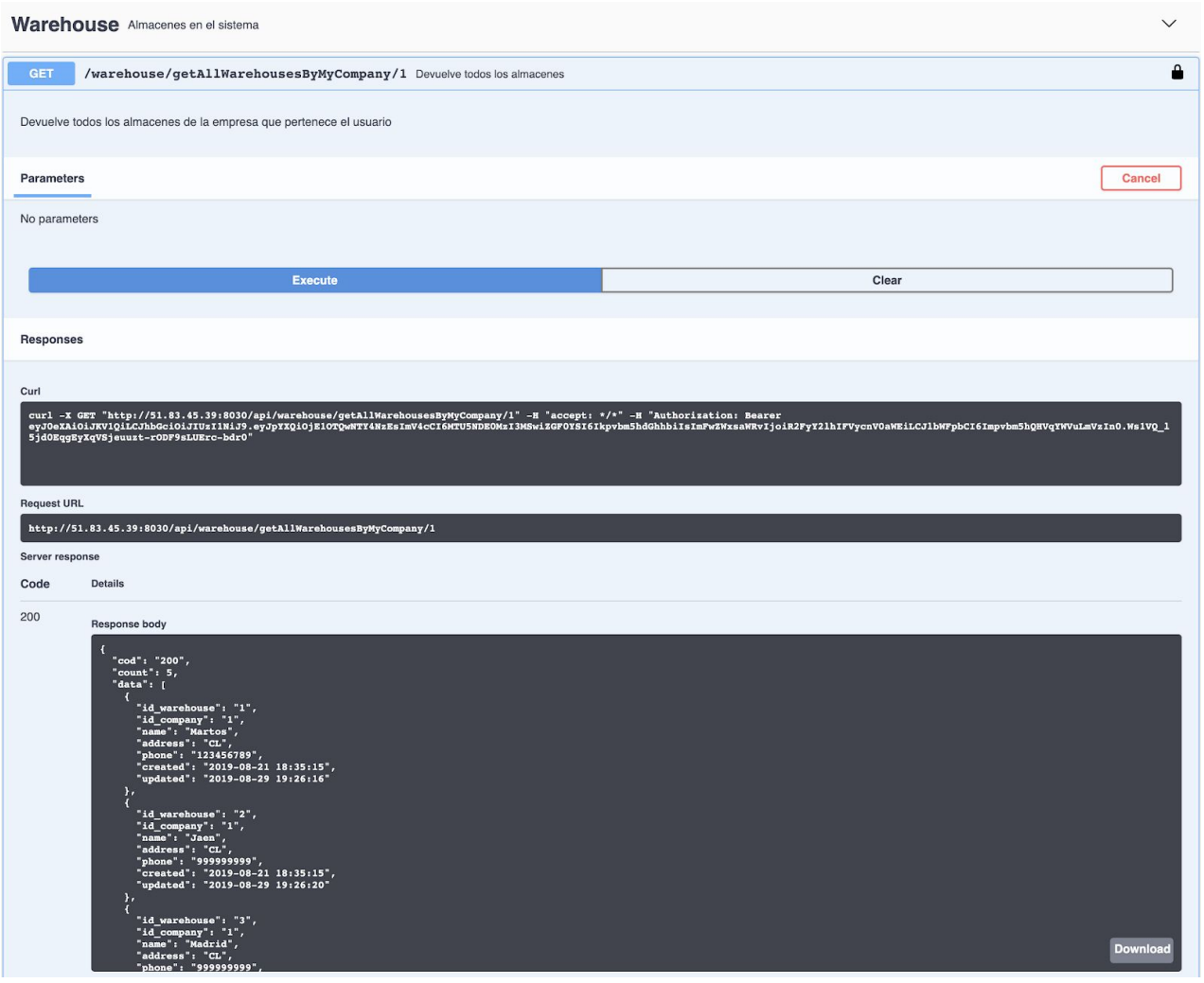


Ilustración 76. Swagger - atacando a un endpoint

Este sería el procedimiento para probar los endpoints desde swagger, como hemos visto en este caso no hemos necesitado pasarle ningún parámetro pero en caso que haga falta swagger lo indicará y nos mostrará una caja de texto en la cual indicaremos el/los parámetro(s) de entrada(s), como se puede ver en la ilustración 77.

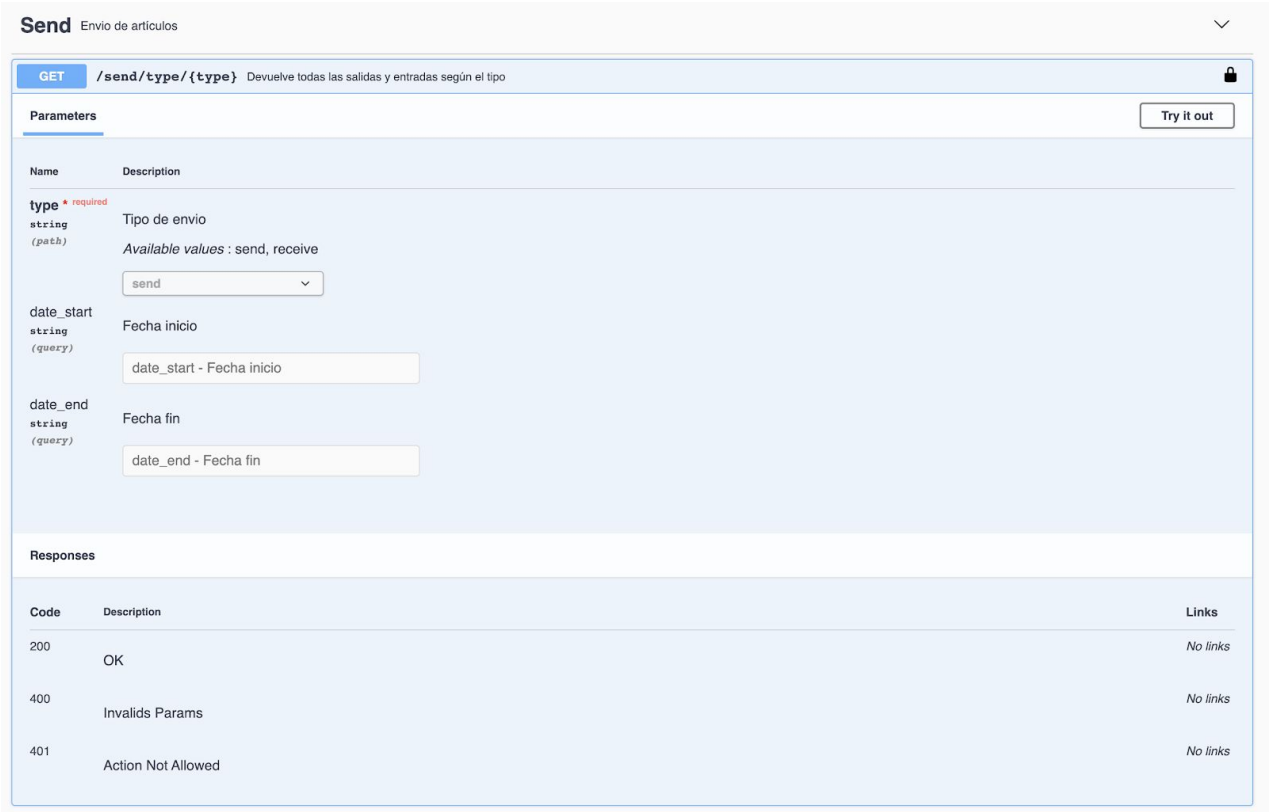


Ilustración 77. Swagger - Endpoint pasando parámetros

Otro de los puntos importante en la documentación de un API REST son los códigos de respuestas, los cuales los podemos encontrar en el apartado inferior de cada endpoint tal y como se muestra en la ilustración 77, en ese caso tiene los siguientes códigos de respuestas.

- 200 → OK
- 400 → Invalid params
- 401 → Action not allowed

Estos códigos pueden cambiar en función del endpoint pero lo bueno que tiene swagger es que podemos personalizar estos códigos por cada endpoint.

7. Conclusiones

Hemos llegado al apartado en el que tenemos que hacer una retrospectiva para sacar conclusiones de este proyecto.

Por una parte considero que se han cumplido los objetivos principales propuestos al inicio del TFG:

- Realizar un estudio de entornos de desarrollo multiplataforma basados en tecnologías web, como podemos ver en el apartado de Análisis.

- Realizar una API REST con toda la lógica de negocio que soporte las peticiones de la aplicación móvil multiplataforma que también se ha desarrollado, realizado con Ionic para el front-end y Zend Framework para el back-end.

- Diseño y desarrollo de un cliente móvil multiplataforma con soporte de noticiones desde el servidor, realizado con Ionic y Firebase.

Al empezar este TFG ya tenía conocimientos sobre HTML, CSS y JAVASCRIPT pero realizando esta aplicación he podido estudiar distintas tecnologías del frontend, tecnologías actualmente muy demandadas en el mercado laboral tales como Angular, React o Dart.

Después de investigar sobre estas tecnologías decidí realizar el TFG utilizando Ionic, un SDK que permite realizar aplicaciones híbridas con tecnologías web tales como Angular, React o Vue.js.

De estas tres me decanté por Angular ya que vi una buena oportunidad para obtener esta destreza que me vendría bien en mi actual trabajo.

Para realizar este TFG hacía falta realizar una arquitectura adecuada la cual soportará las distintas peticiones de las aplicaciones móviles, es por ello que se implementó un API REST el cual se encarga de servir todos los recursos que la aplicación solicite siempre que se cumpla con las restricciones.

Si queremos que nuestra aplicación sea accesible por el mayor número de personas es importante tener un servidor que pueda respaldarlo, en este TFG se utilizó una VPS del proveedor OVH en la cual se instaló y configuró ionic, apache, mysql y webmin.

Un punto a destacar de este TFG es que encontré aplicaciones realmente útiles como es el caso de Figma, este software me permitió plasmar las ideas que tenía sobre el diseño y poder tener una impresión casi al 100% del diseño y las interacciones entre las distintas páginas, todo esto antes de empezar a escribir la primera línea de código. Un punto importante es que consigues resultados profesionales de forma muy rápida y estoy completamente seguro que la usaré en futuros proyectos.

A día de hoy para realizar aplicaciones web, aplicaciones móviles, aplicaciones híbridas, etc. tenemos tantas tecnologías a elegir que no considero que una sea mejor que otra, creo que la tecnología que se utiliza depende mucho del producto a desarrollar.

Escribiendo estas conclusiones aún me hago las preguntas ¿este TFG se podría haber desarrollado con otras tecnologías?, sí; ¿creo que las elegidas son las mejores?, quizás no y explico brevemente el por qué. Creo que debería haber usado otra tecnología para el back-end y no porque Zend Framework no sea una válida si no porque actualmente es una tecnología que tiene una demanda muy baja. Si tuviera que decidir una tecnología a día de hoy para desarrollar el back-end me decantaría más por Nodejs, Spring o Django con Python. Aun así considero que este TFG me ha ayudado a aprender algunas que no tenía y reforzar otras cuantas.

Personalmente me ha resultado interesante y he podido darme cuenta de lo difícil e importante que es realizar una buena documentación ya no solo para nosotros, es más creo que nosotros seremos los que menos veamos la documentación, si no para las personas que puedan realizar futuras mejoras al producto. Al final, si nos dedicamos al código o al desarrollo del software en cualquiera de las etapas pasaremos un gran porcentaje del tiempo leyendo código o cualquier documentación sobre el software y este tipo de documentos nos puede evitar verdaderos quebraderos de cabeza.

7.1 Mejoras y trabajos futuros

En este TFG se pueden realizar mejoras, por ejemplo.

- Realizar un panel de administración para asignar a los trabajadores a los distintos almacenes.
- Añadir opciones de crear, editar y eliminar artículos, categorías y subcategorías.
- Convertir el código único de los envíos de artículos en código QR los cuales serían escaneados desde la APP y automáticamente se localizaría el envío.
- Generar avisos automáticos cuando un artículo llegue al stock mínimo.
- Incluir imágenes de los artículos.
- Generar informes de las salidas, entradas, artículos, etc.

Todos estos puntos de mejora se pueden realizar sin problemas gracias al diseño, arquitectura y tecnologías utilizadas para realizar este TFG, por ejemplo podemos utilizar Angular para crear el panel de administración, consumir datos atacando a nuestra API identificado el origen de la petición (app móvil o app web) y servir los distintos recursos. Podemos generar los avisos automáticos programando un script que se ejecute automáticamente todos los días gracias a un cron job el cual podemos configurar fácilmente desde el panel de webmin[35], y por último generar informes y que estos puedan ser exportados gracias a las muchas librerías de JavaScript que nos permiten realizar dichas acciones.

[35] https://doxfer.webmin.com/Webmin/Scheduled_Cron_Jobs

8. Bibliografía

[1]"AEC - Gestión de stocks", Aec.es, 2020. [Online]. Available: https://www.aec.es/web/guest/centro-conocimiento/gestion-de-stocks. [Accessed: 23- Jul- 2020]

[2]"INVENTARIO DE EXISTENCIAS: 5 PUNTOS A TENER EN CUENTA", *Blog CE Rios Rosas*, 2020. [Online]. Available: https://www.asesoriafiscallaboralmadrid.es/inventario-de-existencias-necesario/. [Accessed: 23- Jul- 2020].

[3]R. Pressman, *Ingeniería del software*, 7th ed. .

[4]M. Piattini Velthuis, *Análisis y diseño de aplicaciones informáticas de gestión*. México, D.F.: Alfaomega, 2004..

[5]"Think with Google - Discover Marketing Research & Digital Trends", *Think with Google*, 2020. [Online]. Available: https://www.thinkwithgoogle.com/. [Accessed: 23- Jul- 2020].

[6]"Todas las estadísticas sobre móviles que deberías conocer 2019", *Mktefa.ditrendia.es*, 2020. [Online]. Available: https://mktefa.ditrendia.es/blog/todas-las-estad%C3%ADsticas-sobre-m%C3%B3viles-que-deber%C3%ADas-conocer-mwc19. [Accessed: 23- Jul- 2020].

[7]R. Khanna, S. Yusuf and H. Phan, *Ionic : Hybrid Mobile App Development,* 2017. [Online]. Available: https://buscaenbuja.ujaen.es/discovery/fulldisplay?context=L&vid=34CBUA_UJA:VU1&search_scope=MyInst_and_CI&tab=Jaen&docid=alma991003849699204994

[8]"BOE.es - Documento BOE-A-2019-14977", *Boe.es*, 2020. [Online]. Available: https://www.boe.es/diario_boe/txt.php?id=BOE-A-2019-14977. [Accessed: 23- Jul- 2020].

Apéndice I. Manual de usuario

En este apartado se creará un manual para tener una visión más global de la aplicación.

La interfaz se ha planteado para que sea lo más intuitiva posible, consiguiendo así mejorar la experiencia de los usuarios y que la curva de aprendizaje sea mínima.

Lo primero que tenemos que hacer para poder acceder a la aplicación es indicar nuestras credenciales y pulsar en entrar (ilustración 78).

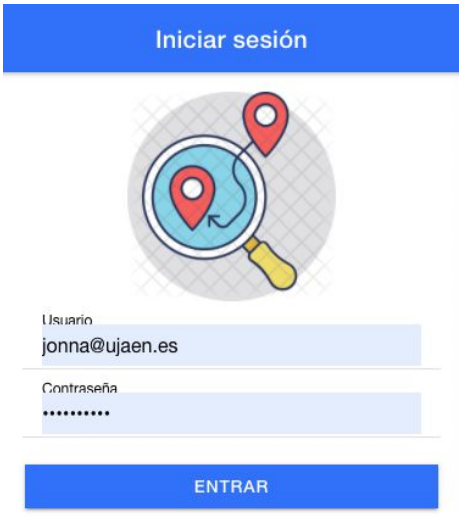


Ilustración 78. Manual - Iniciar sesión

Si las credenciales son correctas nos llevará a la página principal. En esta página podemos ver los artículos disponibles en cada uno de nuestros almacenes, para ello seleccionaremos en el desplegable el almacén en cuestión (Ilustración 79) y automáticamente nos mostrarán los artículos (Ilustración 80).

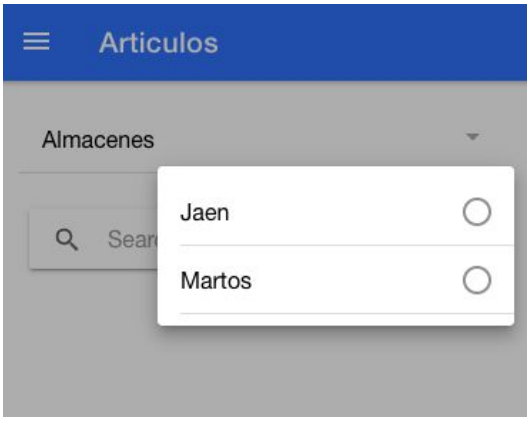


Ilustración 79. Manual - Seleccionar almacén

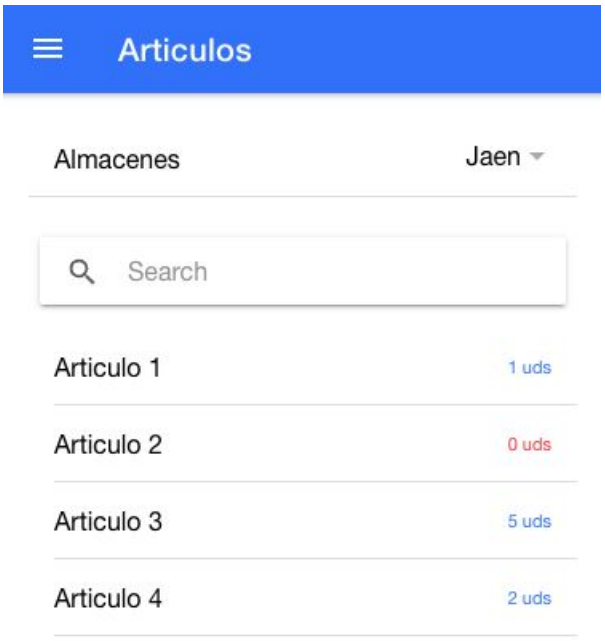


Ilustración 80. Manual - Listado de artículos

Como podemos ver en la ilustración 80, nos aparecen los distintos artículos y el stock de cada uno del almacén seleccionado, en este caso Jaén.

Cuando un artículo tiene stock 0 éste se marcará en rojo para llamar la atención del operario y que lo pueda reponer.

Otra posibilidad en esta página es filtrar entre los distintos artículos de nuestro almacén, para ello escribiremos en el buscador el artículo que buscamos y este nos irá mostrando sugerencias conforme estemos escribiendo (ilustración 81).

Ilustración 81. Manual - Buscar artículo

Otro característica de la aplicación es poder generar envíos de artículos desde un almacén A a otro B, para ello nos iremos al menú lateral y pulsaremos en **SALIDAS** (ilustración 82).

Ilustración 82. Manual - Menú salida

En esta página podemos consultar todas las salidas realizadas desde nuestros almacenes, ver los detalles, crear y buscar una salida.

Tal y como vemos en la ilustración 83 las salidas están organizadas por TARJETAS las cuales son interactivas y podemos ver los detalles de cada una si pulsamos sobre ellas.

Ilustración 83. Manual - Salidas

Si pulsamos sobre alguna de las tarjetas se abrirá otra página con todos los artículos que conforman la salida (imagen 84).

Ilustración 84. Manual - Detalle salida

En esta página nos encontramos con los datos más relevantes de la salida, como son el código el cual usaremos para identificar de forma única una salida o entrada y eventualmente generar un código QR el cual podrá ser leído por algún operario y así localizar el envío de una forma más rápida.

También podemos ver la fecha en la que se realizó la salida, el origen y destino y la lista de todos los artículos.

Un punto importante a tener en cuenta son los **estados**, tanto de la salida en global, como de cada uno de los artículos. Como vemos en la ilustración 84 cada artículo tiene un estado, estos pueden tener tres valores:

- **Pendiente** → Aún no se ha recibido ningún artículo.
- **Recibido** → Se ha recibido la cantidad exacta de un artículo.

- **Recibido parcialmente** → Se ha recibido la cantidad parcial de uno de los artículos.

El estado global de la salida también puede tomar distintos valores, estos son:

- **Enviado** → Cuando ningún artículo ha llegado a su destino.
- **Recibido** → Cuando todos los artículos de la salida han llegado correctamente a su destino.
- **Recibido parcialmente** → Cuando queda algún artículo de la salida por confirmar en el destino.

Otra de las opciones en la vista SALIDA es la de buscar alguna salida en concreto, para ello usaremos el buscador, los criterios de búsqueda con dos, fecha inicio y fecha fin (ilustración 85).

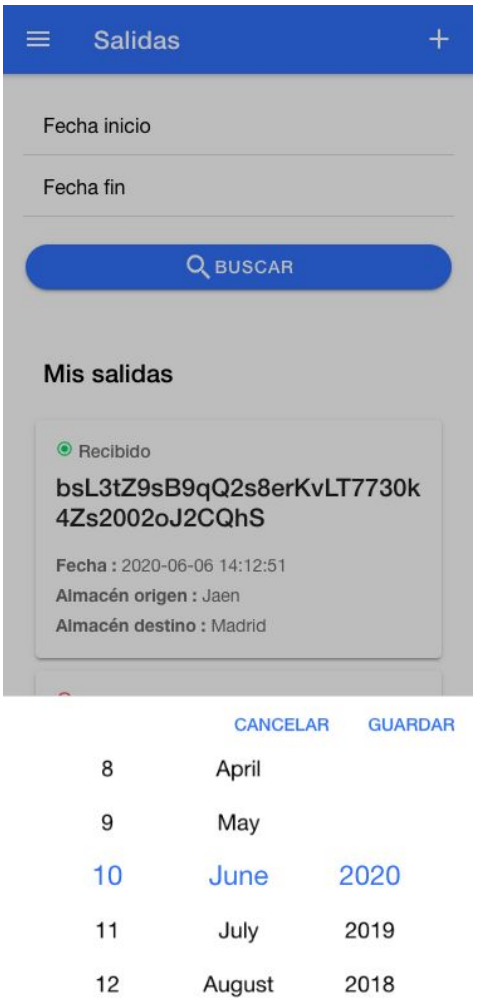


Ilustración 85. Manual - Filtrar salida

La última opción es la de crear una salida, para ello pulsaremos en el botón **+** situado en la parte superior derecha.

Tal y como vemos en la ilustración 86 nos encontramos con un formulario separado en dos pasos.

- El primero es seleccionar el almacén de origen y el almacén destino.
- El segundo paso es seleccionar los artículos que serán enviados en esta salida y la cantidad de cada uno de ellos.

Todos los artículos que vamos añadiendo nos aparecerán en la tabla *"Artículos añadidos hasta el momento",* desde aquí podemos eliminarlos en caso que nos hayamos equivocado.

Ilustración 86. Manual - Crear salida

Por último pulsaremos en ENVIAR lo cual nos mostrará un mensaje de confirmación, como podemos ver en la ilustración 87.

Ilustración 87. Manual - Confirmación para crear una salida

El último apartado de la aplicación son las ENTRADAS. Para acceder a ellas al igual que con las salidas tenemos que ir al buscador lateral y pulsar sobre ENTRADAS (ilustración 88).

Ilustración 88. Manual - Menú entrada

En este apartado nos encontraremos con una página muy parecida a SALIDAS. Como vemos en la ilustración 89, las entradas están separadas por CARS los cuales también son interactivos y desde los cuales confirmaremos las entradas de los distintos artículos.

También tenemos la opción de filtrar las entradas por dos criterios de búsqueda, fecha inicio y fecha fin.

Ilustración 89. Manual - Entradas

Las entradas al igual que las salidas cuentan con un estado global y un estado por cada artículo, como vemos en la ilustración 90.

Ilustración 90. Manual - Confirmar entrada

En esta página tenemos datos como el código, fecha, origen y destino y la lista de artículos que conforman la entrada.

Esta lista de artículos tiene una peculiaridad y es que podemos confirmar la entrada de ellos, pulsando en el check de cada artículo.

Cuando hayamos confirmado la entrada de los artículos pulsaremos en guardar y este generará actualizará el estado de la entrada según los artículos que hayan sido recepcionados.

Como podemos ver la aplicación es intuitiva lo cual es positivo para que los usuarios puedan familiarizarse con ella lo más pronto posible.

Apéndice II. Despliegue de la aplicación

En este apartado hablaremos sobre las herramientas utilizadas para desplegar el proyecto.

1. Servidor OVH

OVH[36] es el proveedor de alojamiento web y telecomunicaciones que se ha utilizado para este TFG. Ofrece servidores dedicados, alojamiento, VPS y servicios cloud computing.

El servicio que tenemos contratado es una VPS SSD con las siguientes características técnicas.

- CPU : 1 vCore
- RAM : 2 GiB
- ALMACENAMIENTO : 40 GB

2. Webmin

Webmin un un panel de control con interfaz gráfica para administrar servidores. La interfaz es muy amigable y nos permite realizar todo tipo de configuración en nuestro servidor, desde crear usuario, servidores virtuales, configurar firewall, hasta programar un cron job para que se ejecute cada determinado tiempo.

Este panel nos ayuda a poder realizar todas estas tareas de configuración que a menudo resultan tediosas y nos permite centrarnos más en realizar una buena infraestructura.

Uno de los puntos más importantes de Webmin, además de los claramente mencionados es que es accesible desde la web por lo que podremos realizar nuestras configuraciones desde cualquier lugar.

Personalmente creo que todos deberíamos de saber los principios básicos de administración de servidores mediante líneas de comando y tener un buen manejo de la terminal, pero a día de hoy es cada vez más común manejar servidores a través de una interfaz gráfica, ya sea un servidor propio o alguna solución cloud como AWS, Azure o Google Cloud.

Para acceder al panel de control de Webmin tenemos que dirigirnos a la siguiente dirección.

https://{dirección_ip}:1000

[36] https://www.ovhcloud.com/es-es/vps/

Siendo ip, la dirección de nuestro servidor. Al entrar en esta dirección nos aparecerá el formulario para introducir nuestras credenciales de acceso, tal y como se muestra en la ilustración 91.

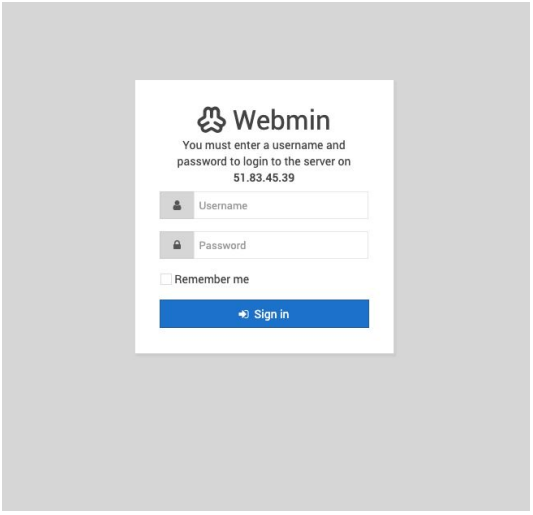


Ilustración 91. Webmin

¿Cómo crear un virtual server?

Esto lo podemos hacer muy fácilmente gracias a Webmin para ello solo tenemos que pulsar sobre “CREATE VIRTUAL SERVER” lo cual nos abrirá un formulario como el que vemos en la ilustración 92.

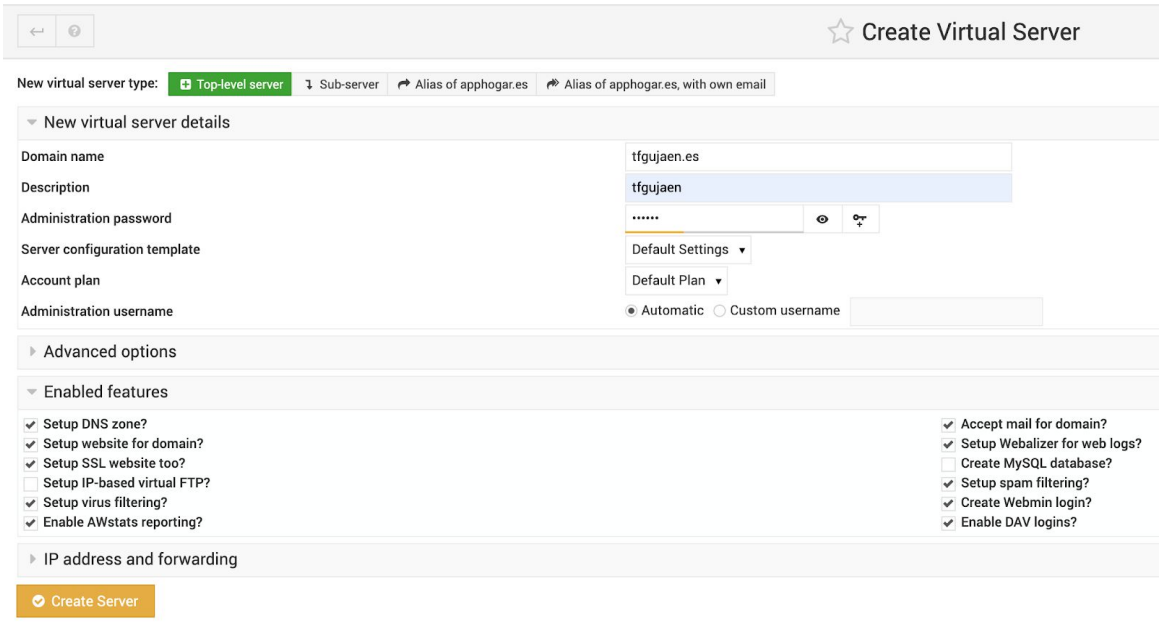


Ilustración 92. Crear virtual server

Aquí podemos realizar muchas configuración pero para este tfg sólo indicaremos cual es el nombre de nuestro virtual server y la contraseña, esto es importante recordarlo ya que serán las credenciales con las que nos conectaremos por ssh.

Lo siguiente que tenemos que hacer cuando hayamos creado nuestro virtual server es asignarle un puerto, para ellos nos vamos a **Change IP Address**, como vemos en la ilustración 93.

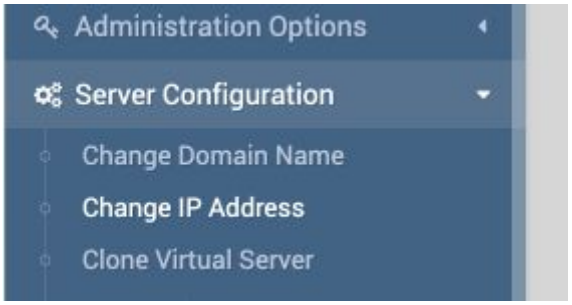


Ilustración 93. Cambiar puerto del virtual server

A este virtual server le asignaremos el puerto 8030 como vemos en la ilustración 94.

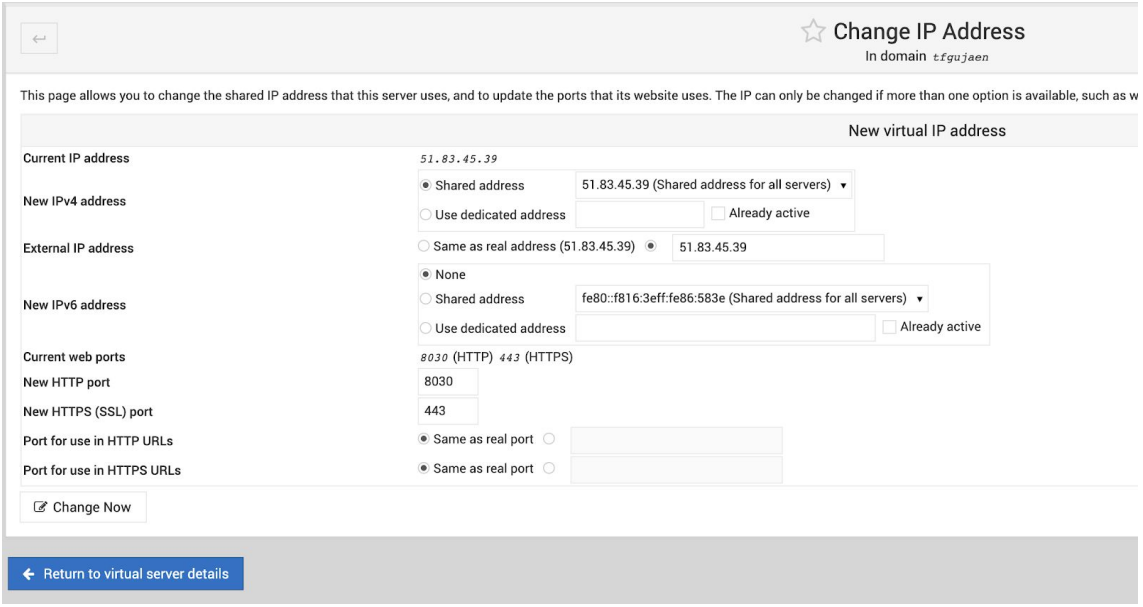


Ilustración 94. Cambiar puerto HTTP

Con esto habremos configurado nuestro virtual server y podremos acceder a él indicando la dirección ip seguida del puerto que le acabamos de asignar.

Después de tener nuestro virtual server funcionando tenemos que desplegar el proyecto, para ello lo primero que hacemos es conectarnos mediante ssh a nuestra VPS, para ellos abrimos nuestra terminal y escribimos el siguiente comando e introducimos nuestra contraseña tal y como vemos en la ilustración 95.

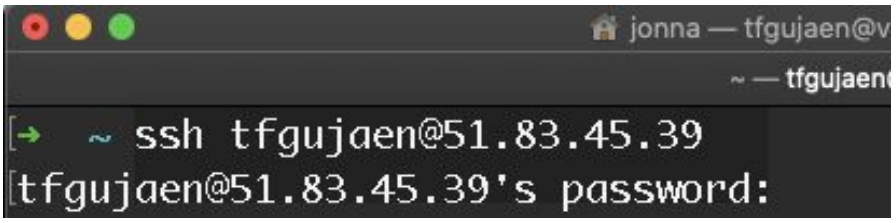


Ilustración 95. Conectarse al servidor

Lo siguiente es situarnos en la carpeta public_html y desplegar el proyecto, en este caso el back-end, para esto se utilizó GitHub y se subió todo el backend en un repositorio el cual clonaremos con el siguiente comando tal y como vemos la ilustración 96.

Ilustración 96. Clonar el proyecto

Ya que tenemos solo una rama lo único que tenemos que hacer es cambiar nuestro archivo application.ini y apuntar a nuestra base de datos y listo con esto tenemos desplegado nuestro backend.

Para desplegar el front-end los pasos a seguir son los mismos, excepto que después de clonar el proyecto de github tenemos que ejecutar el siguiente comando.

```
> ionic serve --address [IP]
```

3. Virtualmin

Virtualmin es una extensión muy útil de Webmin que nos permite gestionar todo lo relacionado con dominios, virtualhost, correos, usuarios, etc.

Al igual que Webmin es gratis aunque cuenta con una versión de pago. Virtualmin tiene el panel integrado en Webmin por lo que para acceder a este usaremos el mismo portal.

Apéndice III. Descripción de contenidos suministrados

- Memoria.pdf
- Contenido complementario.zip
    - Código fuente de la aplicación cliente
    - Código fuente de la aplicación servidor API REST
    - Fichero .sql de la base de datos