TRABAJO FIN DE GRADO

Grado en Ingeniería Informática

UNIVERSIDAD DE ALMERIA

ESCUELA SUPERIOR DE INGENIERÍA

App multiplataforma de automatización de mensajes

Curso 2020/2021

Alumno/a:

Antonio Gutiérrez Villegas

Director/es:

Rafael Guirado Clavijo

AGRADECIMIENTOS

Agradecer a mi familia, en especial a mis padres Adolfo y María José y a mi hermano Adolfo, por ayudarme y apoyarme en este y en todos los proyectos de mi vida.

Agradecer a mis amigos, por haber andado tantos caminos junto a mi, creciendo juntos y aprendiendo de cada experiencia.

Por último, agradecer a la Universidad de Almería y sus docentes por todas las enseñanzas y oportunidades dadas. Y en concreto a mi tutor Rafael Guirado por su apoyo en todo momento y su colaboración con todas las ideas de este proyecto.

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

RESUMEN

Hoy en día los servicios de mensajería son cada vez más utilizados y por ello necesitan estar actualizados a las últimas tecnologías para facilitar su desempeño. Es de aquí donde nace la idea de este proyecto, la creación de una app multiplataforma de automatización de mensajes a través de GPS orientado a empresas de reparto. Esta herramienta facilita y hace más seguro el trabajo de los repartidores, permitiéndoles agilizar el proceso de las entregas sin necesidad de hacer uso de su dispositivo durante la conducción. Mediante el cálculo de distancias entre ellos y las entregas a través de GPS se enviarán mensajes a una determinada distancia de los clientes a través de "Whatsapp", todo esto sumado a una interfaz cómoda para la gestión de sus repartos y centralización de los datos con un servidor.

Palabras clave: *App multiplataforma, automatización de mensajes y GPS.*

ABSTRACT

Today messaging services are increasingly used and therefore need to be updated to the latest technologies to facilitate their performance. This is where the idea of this project was born, the creation of a multiplatform app for automating messages through GPS aimed at delivery companies. This tool makes the work of delivery men easier and safer, allowing them to streamline the delivery process without having to use their device while driving. By calculating the distances between them and the deliveries through GPS, messages will be sent to a certain distance from the clients through "WhatsApp", all this added to a comfortable interface for managing their deliveries and centralizing the data with A server.

Keywords: *Cross-platform app, message automation and GPS.*

1. INTRODUCCIÓN

1.1 Motivación

Hoy en día el volumen de compras online ha crecido exponencialmente con respecto a hace 20 años, esto supone un aumento prolífico de la cantidad de repartos llevados a cabo durante el día en ciudades de todo el mundo. Anteriormente los sistemas de reparto funcionaban de manera correcta con respecto a los volúmenes de trabajo, pero actualmente, que estos volúmenes han crecido tanto, los sistemas deben de ser cada vez más eficientes para poder dar servicio a tan alta demanda [1].

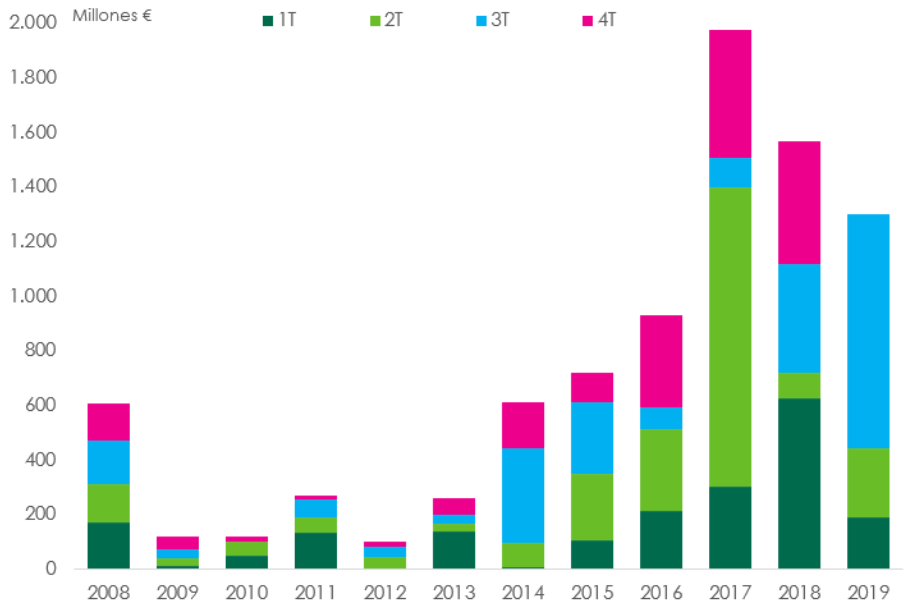


Figura 1 Inversión en logista hasta 2019 en España

A pesar del gran aumento de repartos, las tecnologías asociadas a éstos, que conectan al cliente con su pedido, no han avanzado tanto. Muchas empresas de reparto ofrecen aplicaciones para conocer el estado de los envíos del cliente, pero no solo la información proporcionada es escasa, sino que, en el punto más crítico, como es el proceso de entrega, es casi nula. Esto supone que el cliente no conozca el estado de su pedido en proceso de entrega hasta que el repartidor toque a la puerta, impidiendo realizar otras tareas fuera de casa con tranquilidad y haciendo perder el tiempo a repartidores que no puedan realizar la entrega por ausencia del cliente.

Por otro lado, debido a la gran cantidad de empresas de logística, para que el cliente pueda conocer el estado de su pedido de forma oficial necesita tener instalada la aplicación de cada una de estas empresas o consultarlas vía web de forma poco cómoda desde un dispositivo móvil.

Surge así el objetivo de este proyecto, dar una solución a estos dos problemas facilitando el servicio tanto para repartidores como clientes. La solución desarrollada para este proyecto es la creación de una app compatible tanto para dispositivos Android como IOS que avise a los clientes del proceso de entrega de sus pedidos a través de la aplicación de mensajería instantánea más extendida en el mundo como es “Whatsapp”, haciendo posible a los clientes

interactuar con su pedido próximamente entregado, cancelándolo en caso de no poder estar en el proceso de entrega a través de un simple mensaje y ahorrando tiempo a los repartidores en ese caso.

Este proyecto se desarrolla con tecnologías como son Ionic [2] para la creación de la aplicación del dispositivo y Node.js [3] para la parte del servidor.

1.2 Objetivos

El principal objetivo de este TFG es crear una app híbrida para la cómoda gestión de los clientes por parte de los repartidores, con funcionalidades útiles para la agilización de repartos e interconectada con un servidor para, mediante la gestión de usuarios, poder trabajar con múltiples dispositivos desplegados en una flota de repartidores.

- Desarrollar una aplicación híbrida o multidispositivo Ionic (que a su vez está basado en el framework Angular) para la gestión de repartos de un repartidor.
- Crear una aplicación que corra en el servidor para atender peticiones http, manteniendo conectadas la aplicación desplegada en diferentes dispositivos en un mismo punto mediante Node.js y Express [4].
- Desarrollar y trabajar con bases de datos del sistema MongoDB [5] para la gestión de los diferentes usuarios de la aplicación de gestión de repartos.
- Desplegar la aplicación del servidor y la base de datos en servicios de hosting online para poder acceder a ellas desde cualquier lugar.
- Conectar las aplicaciones del dispositivo y del servidor con las principales opciones del mercado para mapas y mensajería como son “Google Maps” y “Whatsapp” haciendo lo más familiar posible su uso para el usuario y para el cliente.

1.3 Planificación estimada

A principios de 2021 se elaboró una planificación temporal estimada sobre el tiempo que conllevaría el desarrollo de este proyecto.

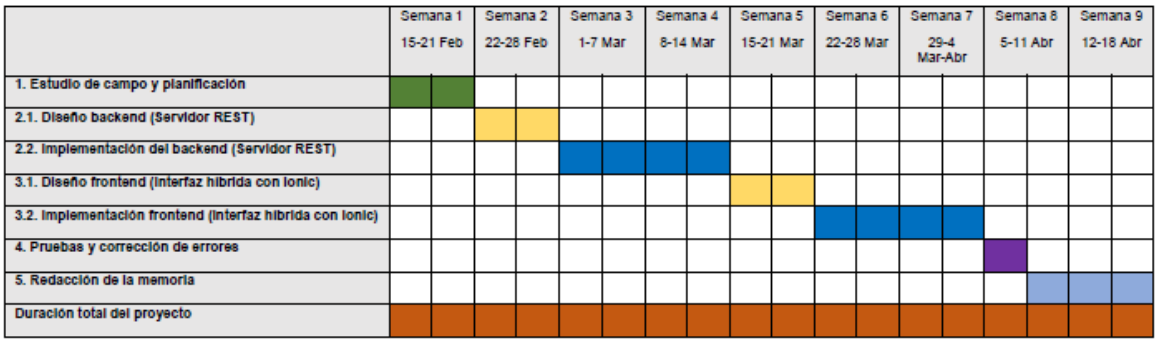


Figura 2 Planificación temporal del anteproyecto

Esta planificación se ha visto finalmente aplazada hasta después del primer visto bueno por la comisión (sujeto a ciertos cambios) y alterada durante el desarrollo habiendo sido cambiados de orden ciertos puntos y modificado la duración temporal de otros. Se ha mantenido la estimación inicial de 35 horas semanales de media para el desarrollo del proyecto menos la última semana que se estimaba en 20, haciendo un total de 300 horas.

1.4 Estructura de la memoria

Esta memoria se divide en los siguientes puntos principales:

- Introducción: Tiene el objetivo de mostrar la naturaleza del proyecto, así como los objetivos que se persiguen y su planificación de desarrollo en el tiempo.
- Estado del arte: Pretende dar a conocer las posibilidades relacionadas con el problema de origen que están disponibles actualmente en el mercado y por qué merece la pena desarrollar una nueva solución.
- Tecnologías y herramientas: Su objetivo es explicar las distintas tecnologías usadas en el desarrollo del proyecto y por qué se escogieron éstas en lugar de otras alternativas.
- Desarrollo del proyecto: Punto en el que se recorrerá el desarrollo del proyecto detalladamente. Veremos su estructura, el análisis de requisitos, su diseño y el proceso de desarrollo.
- Conclusiones y líneas de trabajo futuras: Dará una valoración final del proyecto y estudiará posibles líneas para continuar con el desarrollo y mejora de éste en un futuro.

2. ESTADO DEL ARTE

De un tiempo a esta parte muchas empresas de logística nacionales e internacionales han desarrollado sus aplicaciones para dar servicio de seguimiento en los pedidos de sus clientes, pero éstas no han avanzado al mismo ritmo que lo ha hecho la demanda de pedidos, habiendo quedado obsoletas con respecto a las necesidades de clientes y repartidores a la hora de realizar una entrega.

En este momento para recibir una pequeña información sobre el estado de nuestro pedido necesitamos acceder a cada una de las aplicaciones o webs de empresas de logística que hay, ya que cada tienda online en la que pedimos trabaja con una distinta.

La información proporcionada es vaga muchas veces y prácticamente nula en el proceso final de la entrega, siendo éste uno de los puntos más críticos del proceso. Repartidores no tienen forma de avisar con antelación a los clientes debido al gran volumen de trabajo que tienen y los clientes solo conocen el día en el que el pedido será entregado de tal modo que, si por necesidad tienen que salir de casa, fallan al proceso de entrega.

2.1 Correos

Figura 3 Logo de Correos

Correos [6] es la mayor empresa estatal de capital público propiedad del gobierno español, es la encargada del servicio postal universal del país, sus orígenes datan de 1716.

Actualmente correos da servicio al 49% de los comercios online en España debido a su eficacia a nivel nacional y su bajo coste. Su objetivo es ser el principal servicio de paquetería del mercado español.

Desde el año 2002 correos comenzó un plan de automatización para sus centros logísticos, un año más tarde creó su oficina virtual en la web y en 2005 los repartidores y carteros de correos comenzaron a utilizar PDA para el registro de las entregas. En los últimos 20 años Correos no ha dejado de adaptarse al contexto socioeconómico del momento y a aplicar el uso de nuevas tecnologías para saciar la demanda de comunicación que existe en el presente.

En la actualidad entre sus servicios de oficina virtual está una app para dispositivos móviles lanzada a finales de 2019 y con más de 7.000 valoraciones en la tienda de aplicaciones de Google Play de Android y 950 en la App Store de IOS.

2.2 SEUR

Figura 4 Logo de SEUR

SEUR [7] fundada en 1942 y a día de hoy es una de las empresas de logística más importantes en España y mediante las redes de GeoPost entrega en más de 230 países.

Actualmente SEUR cuenta con una plantilla de 7.500 empleados y una flota de 4.500 vehículos con la que da servicio a más de 1.200.000 clientes. SEUR está considerada la principal compañía de transporte urgente en el país. Fue una de las empresas pioneras en trabajar con los comercios electrónicos gracias a su servicio tradicional de entregas acompañado de un servicio de devoluciones sin cobro de sobrecostes.

La web de SEUR es actualmente el único lugar disponible para el seguimiento de pedidos para los clientes que contratan su servicio ya que no dispone de aplicaciones oficiales en el Google Play ni en el App Store.

2.3 MRW

Figura 5 Logo de MRW

MRW [8] fue fundada en 1977 en Barcelona para cubrir la demanda de paquetería nacional, pero a fecha actual ya da servicio a España, Portugal, Gibraltar, Andorra y Venezuela.

Desde 2011 cuenta con más de 10.000 empleados en sus filas y 1.300 franquicias y desde hace años están especializados en transporte de animales vivos, medicamentos y comercios online, siendo los únicos del sector habilitados en España para alguno de estos servicios.

Al igual que le ocurre a SEUR no dispone de apps de seguimiento en las tiendas de aplicaciones de Google Play y App Store, por lo que el único método que dispone el cliente para acceder a la información de su pedido es acceder a su web oficial.

2.4 Amazon Logistics

Figura 6 Logo de Amazon Logistics

Fundada en 1994 para la venta online de libros fue una de las primeras compañías de venta online y es considerada actualmente como la empresa más poderosa del mundo por encima de gigantes como Google o Facebook.

Tras el año 2020 Amazon [9] suma un total de 1.298.000 empleados en todo el mundo y más de 175 centros logísticos repartidos por todo el mundo los cuales se encargan de llevar a cabo todo el proceso de gestión desde la recepción de los productos del fabricante hasta el envío al cliente final.

Durante muchos años Amazon se ha encargado del abastecimiento de sus propios almacenes, pero no de los envíos a clientes. Tras la navidad de 2013 debido a un mal servicio por parte de las empresas de mensajería con las que colabora decidió hacerse cargo de parte de estos envíos y en la actualidad se ha convertido en la empresa de logística y distribución más eficiente del mundo.

Amazon Logistics está en constante actualización y de las opciones del mercado es la más puntera en cuanto a nuevas tecnologías puestas al servicio de empleados y clientes. A través de la aplicación de Amazon permite hacer un seguimiento en tiempo real de las paradas restantes hasta la entrega al cliente a través de GPS y aunque no siempre funciona correctamente, bien aprovechada es la opción de mensajería más completas del mercado.

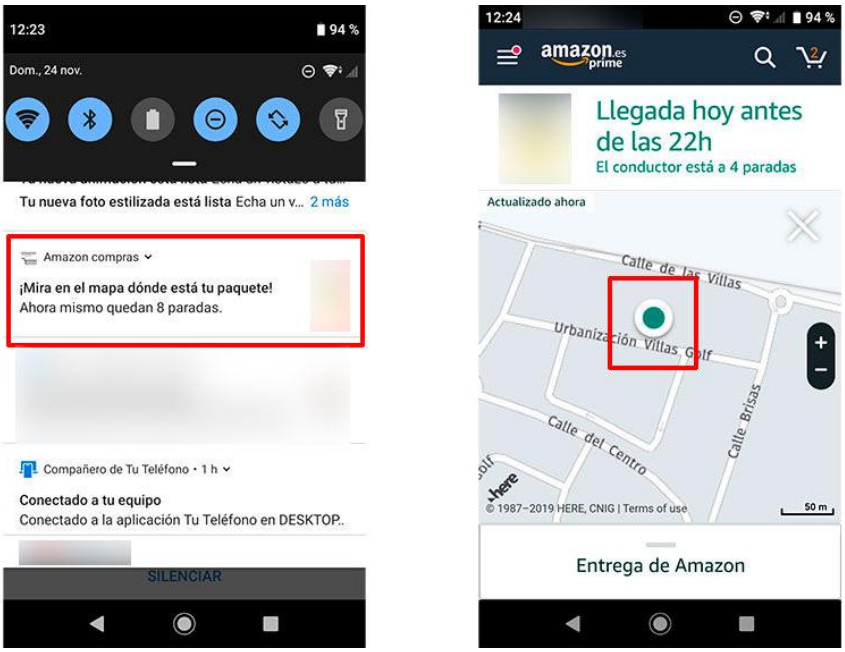


Figura 7 Seguimiento en mapa de Amazon Logistics

2.5 Conclusión

Existen distintos servicios asociados a cada empresa de mensajería para realizar un seguimiento por parte del cliente de los estados de sus pedidos, aunque como hemos visto anteriormente algunos de los principales servicios de mensajería españoles solo disponen de web para este cometido y, tanto los que disponen de aplicaciones para dispositivos como los que solo disponen de web, proporcionan una pobre información sobre los estados del proceso. Amazon es el único servicio de los vistos anteriormente que proporciona algo más de información al cliente en el punto de la entrega.

Los servicios de mensajería que no disponen de la funcionalidad que dispone Amazon hacen que el proceso de entrega del paquete sea poco ágil e incómodo para muchos clientes debido a que no se tiene información de en qué momento del día su pedido será entregado, haciendo imposible en muchos casos atender la llegada del repartidor debido a la necesidad de salir de casa por diversos motivos. Este problema también supone un impedimento al repartidor ya que los paquetes no entregados no se traducen en menos intentos de entrega y el aviso previo a los clientes no siempre es posible debido a la imposibilidad de usar el teléfono durante la conducción y el alto volumen de trabajo.

Otro punto en el que fallan estos servicios es que todos requieren de acceso a sus webs oficiales haciendo un proceso laberíntico para el cliente el consultar los diferentes pedidos que puede estar esperando de forma paralela.

En el caso de Amazon, pese a tener un servicio más moderno con un mapa de paradas mediante GPS, solo es válido para pedidos de su tienda online.

Todo esto, sumado a que en la actualidad casi todo cliente dispone de un dispositivo móvil con acceso a internet y la mayoría de éstos hacen uso de la aplicación de mensajería “Whatsapp”, podría suponer una gran mejora para los servicios de mensajería. Esta mejora sería alcanzable a

través de la integración para el repartidor de un servicio de aviso automático a los clientes mediante “Whatsapp” de una próxima entrega, junto a una posibilidad del cliente para cancelar la misma desde el propio servicio de mensajería y agilizar el trabajo para los repartidores.

A través de la conclusión de este problema y su posible solución nace la idea de este proyecto plasmado en una aplicación multiplataforma para la gestión de repartos.

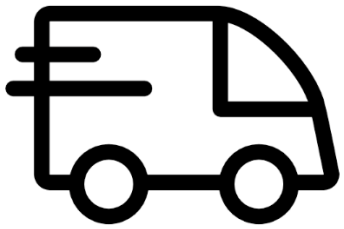


Figura 8 Icono de la aplicación de Gestión de Repartos del proyecto

3. TECNOLOGÍAS Y HERRAMIENTAS

El desarrollo de este proyecto ha requerido del uso de una gran variedad de tecnologías orientadas al desarrollo web y, en este caso, aplicadas al despliegue nativo en dispositivo y la creación de aplicaciones de servidor.

Gracias al uso de frameworks de programación híbrida es posible que de un mismo código base se puedan desplegar aplicaciones que a los ojos del usuario funcionen nativamente en el dispositivo e interactúen con los sensores y propiedades internas de éste. En gran medida, gracias a JavaScript que permite trasladar parte del código, que normalmente se encontraba en el backend del servidor, hasta el navegador web haciendo posible así la creación de aplicaciones web con gran dinamismo y muchas posibilidades.

En la actualidad el uso de herramientas que ayudan a la creación de software es cada vez más imprescindible para la creación de este tipo de productos debido a todas las facilidades y posibilidades que nos ofrecen. Éstos son algunos de los beneficios del uso de éstas herramientas:

- Permiten la creación de productos software con más eficiencia y calidad.
- Dan la posibilidad de realizar un seguimiento de versiones.
- Permiten buscar errores de manera exhaustiva en cualquier punto del código.
- Facilitan el testeo de bloques concretos del software.
- Hacen más fluida la comunicación dentro del equipo.

3.1 Tecnologías

3.1.1 Ionic

Figura 9 Logo de Ionic

Ionic [2] es un framework de código abierto, basado a su vez en el framework Angular de diseño web, pero orientado hacia el desarrollo de aplicaciones móviles híbridas.

Ionic de forma transparente corre su aplicación en un navegador web incorporado a ésta, pero sin que sea visible para el usuario, dando lugar a un aspecto y un desempeño similar al de una aplicación nativa creada para ese dispositivo móvil. Para poder acceder a características internas del dispositivo hacen uso de Apache Cordova y sus librerías JavaScript.

Gracias a las tecnologías en las que Ionic se basa existe una gran comunidad asociada a cada una de éstas para resolver cualquier tipo de problema y contribuir en el crecimiento de este gran framework.

Con Ionic la creación de aplicaciones híbridas para Android, IOS y PWA mediante un único código es posible de una forma eficiente, con estándares web modernos, de fácil acceso para desarrolladores web, de forma rápida, grandes posibilidades y estética vanguardista.

3.1.2 Apache Cordova

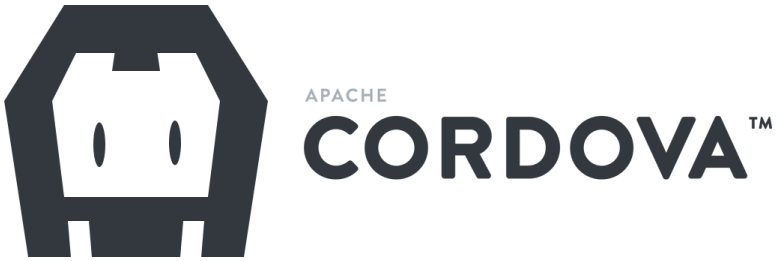


Figura 10 Logo de Apache Cordova

Apache Cordova [10] consiste en un conjunto de librerías escritas en el lenguaje JavaScript que en conjunto con HTML5 [11] y CSS3 [12] dan un gran resultado para el desarrollo de aplicaciones móviles híbridas. Estas librerías dan acceso a componentes nativos del dispositivo haciendo posible el desarrollo para dispositivos móviles desde un único código y sin necesidad de usar el lenguaje nativo con el que trabaja el sistema operativo del dispositivo.

3.1.3 TypeScript

Figura 11 Logo de TypeScript

TypeScript [13] es un lenguaje de programación escrito encima del lenguaje JavaScript aportando facilidades y herramientas muy beneficiosas a la hora de desarrollar proyectos.

Gracias a TypeScript conseguimos una amplia variedad de utilidades que son necesarias para convertir a JavaScript en un lenguaje escalable y a la altura de las necesidades más exigentes con que es un lenguaje verdaderamente orientado a objetos con herramientas como herencias, sobrecargas, etc.

Como hemos dicho, TypeScript está basado y compila código JavaScript, el cual es un lenguaje de programación ligero, interpretado y orientado a objetos. Está basado en el estándar ECMAScript el cual desde 2012 es soportado en todos los navegadores web modernos.

TypeScript puede ser usado tanto en el lado del cliente como en el lado del servidor y se encarga de darle funcionalidad y dinamismo a la aplicación web.

3.1.4 HTML5

Figura 12 Logo de HTML5

HTML5 [11] es un lenguaje de marcado encargado de dar la estructura básica para la creación de páginas web, el cual se encuentra en su versión 5. Sus siglas significan lenguaje de marcado de hipertexto (HyperText Markup Languaje).

Con HTML se crea el esqueleto de contenido mediante el uso de marcadores o etiquetas que estructuran y contienen el contenido del sitio web. Es un estándar internacional para el desarrollo web desde el año 2000.

Hoy en día es imprescindible su uso en conjunto con CSS3 ya que le proporciona estilo a la estructuración hecha con HTML5, permitiendo así crear webs estáticas muy atractivas visualmente.

3.1.5 CSS3

Figura 13 Logo de CSS3

CSS3 [12] es el lenguaje de hojas de estilos encargado de estilizar los diferentes elementos escritos en lenguajes de marcado como HTML, éste se encuentra en su versión 3. Sus siglas significan hojas de estilo en cascada (Cascading Style Sheets).

La relación entre HTML y CSS es muy estrecha ya que mientras HTML se encarga de crear la base del sitio web, CSS es el encargado de darle estilo y estética a esa base. HTML podría trabajar sin esa estética, pero en la actualidad sería impensable el desarrollo de un sitio web sin ella ya que éste se vería completamente anticuado e incómodo de ver y usar.

Mediante las hojas de estilo CSS es posible generar un estilo para un sitio web HTML sin tener que describir cada elemento por separado y de forma repetida, facilitando así el desarrollo y haciendo un código más corto y menos propenso a errores.

3.1.6 Node.js + Express

Figura 14 Logo de Node.js y Express

Node.js [3] es un entorno en tiempo de ejecución, basado en JavaScript, multiplataforma y encargado de ejecutar aplicaciones y servicios en el lado del servidor.

Dispone de un gran rendimiento ya que compila el código JavaScript en código máquina nativo en lugar de interpretarlo y correrlo en un navegador web. Además, es de código abierto por lo que le permite ser un entorno multiplataforma como hemos comentado anteriormente.

Node.js dispone de Bucle de Eventos (Event Loop), lo que hace que sea capaz de gestionar grandes cantidades de cliente de forma asíncrona.

A su vez Express [4] es el framework más popular basado en Node y es la base de muchos otros frameworks web de Node. Proporciona manejo de peticiones HTTP, ajustes para las conexiones del servidor como son el puerto de éste y procesamiento de peticiones middleware para las peticiones entre otras funcionalidades.

3.1.7 MongoDB

Figura 15 Logo de MondoDB

MongoDB [5] es una base de datos no relacional orientada a documentos. En lugar de guardar los datos en registros, éste los guarda en documentos del tipo JSON.

Este tipo de bases de datos no relacionales se adaptan perfectamente a una gran mayoría de aplicaciones actuales orientadas a dispositivos móviles y aplicaciones web, proporcionando un servicio de base de datos flexible, escalable y de alto rendimiento.

El hecho de que los datos sean guardados como objetos JSON hace debido a su sintaxis que sea muy cómodo el envío y recepción de estos archivos para más tarde interpretarlos como objetos.

```
{
 Nombre: "Pedro",
 Apellidos: "Martínez Campo",
 Edad: 22,
 Aficiones: ["fútbol","tenis","ciclismo"],
 Amigos: [
  {
   Nombre:"María",
   Edad:22
  },
  {
   Nombre:"Luis",
   Edad:28
  }
 ]
}
```

Figura 16 Ejemplo de un archivo JSON

3.1.8 Principales APIs

Figura 17 Logo de Google Maps

Google Maps es el servicio de mapas y navegación ofrecido por Google con acceso a mapas desplazables con información de negocios, monumentos y más puntos de interés.

La API de Google Maps [14] contiene el acceso a los mapas que gestiona su plataforma. Para su uso Google ofrece un crédito gratuito de 200$ mensuales, lo que sería equivalente a 100.000 cargas de sus mapas.

Figura 18 Logo de Twilio

Twilio es una plataforma de desarrollo para comunicaciones habilitada entre otras funciones para el uso de las librerías de Whatsapp. Gracias a ella cualquier empresa puede proveer a sus clientes con un número de contacto en Whatsapp para el envío y respuesta automatizado de mensajes (Chatbot).

A pesar de que la API de Twilio [15] no dispone de una versión gratuita, éste nos ofrece un crédito inicial de 15,50$ para probar sus servicios, siendo 0,005$ el coste por mensaje.

3.2 Herramientas

3.2.1 Visual Studio Code

Figura 19 Logo de Visual Studio Code

Visual Studio Code [16] es un editor de código fuente desarrollado por Microsoft que se encuentra disponible para Windows, Linux y MacOS. Admite una gran variedad de lenguajes de programación y gracias a que es Open Source permite la creación de plugins tanto oficiales como creados por la comunidad que amplían su funcionalidad y personalización.

Gracias a su rapidez, elegancia y lo ligero que es hace que sea una de las opciones favoritas a día de hoy para desarrolladores. Uno de los motivos de su ligereza es que este software requiere del uso de un compilador externo ya que es tan solo un editor de código.

Gracias a sus plugins las posibilidades que nos ofrece Visual Studio Code son inmensas, siendo una gran opción para cualquier desarrollador. Su integración con los lenguajes de programación empleados en este proyecto ha hecho que el proceso de desarrollo haya sido muy eficiente y automatizado.

Con un solo editor es posible crear el código de un proyecto completo, aunque éste se componga de muchas tecnologías distintas.

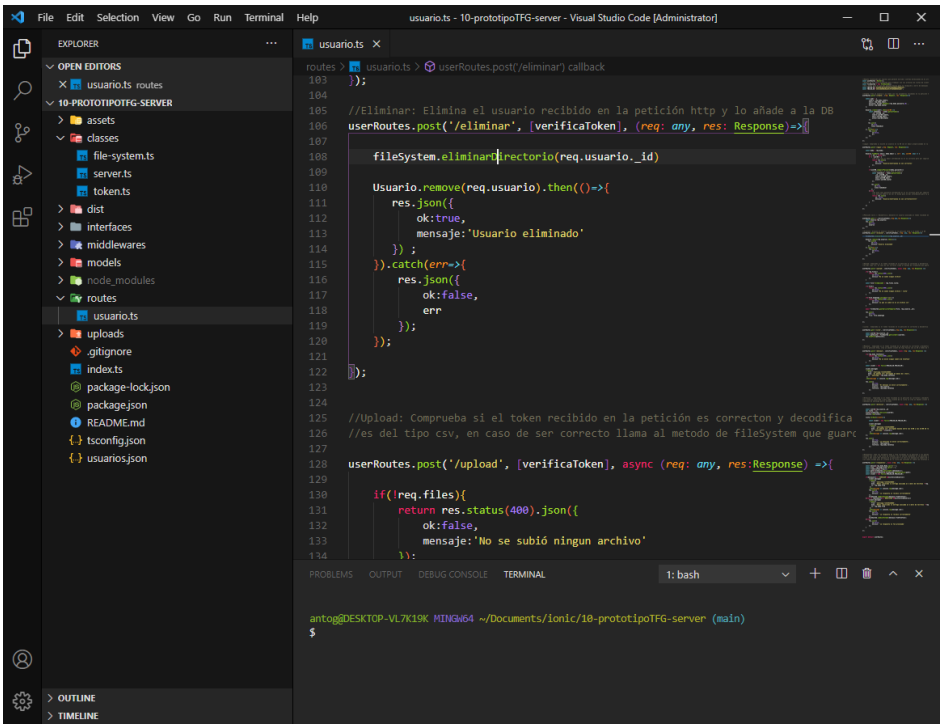


Figura 20 Interfaz de Visual Studio Code

3.2.2 Postman

Figura 21 Logo de Postman

Postman [17] es una herramienta encargada de crear peticiones sobre APIs de manera sencilla con el fin de realizar pruebas sobre éstas. Esta permite realizar peticiones mostrando los resultados de éstas para así facilitar el proceso de testeo y acelerar su desarrollo.

Es una de las herramientas más utilizadas con este fin debido a la atractiva documentación que genera a la hora de hacer las pruebas y su cómoda y sencilla interfaz.

Con postman podemos guardar nuestras peticiones de pruebas y organizarlas por proyectos, así como sicronizarlas con nuestra cuenta de Postman para poder acceder a ellas desde cualquier dispositivo en el que tengamos iniciada sesión. Además de todo esto es multiplataforma por lo que podemos encontrarla disponible tanto en Windows, Linux y MacOS.

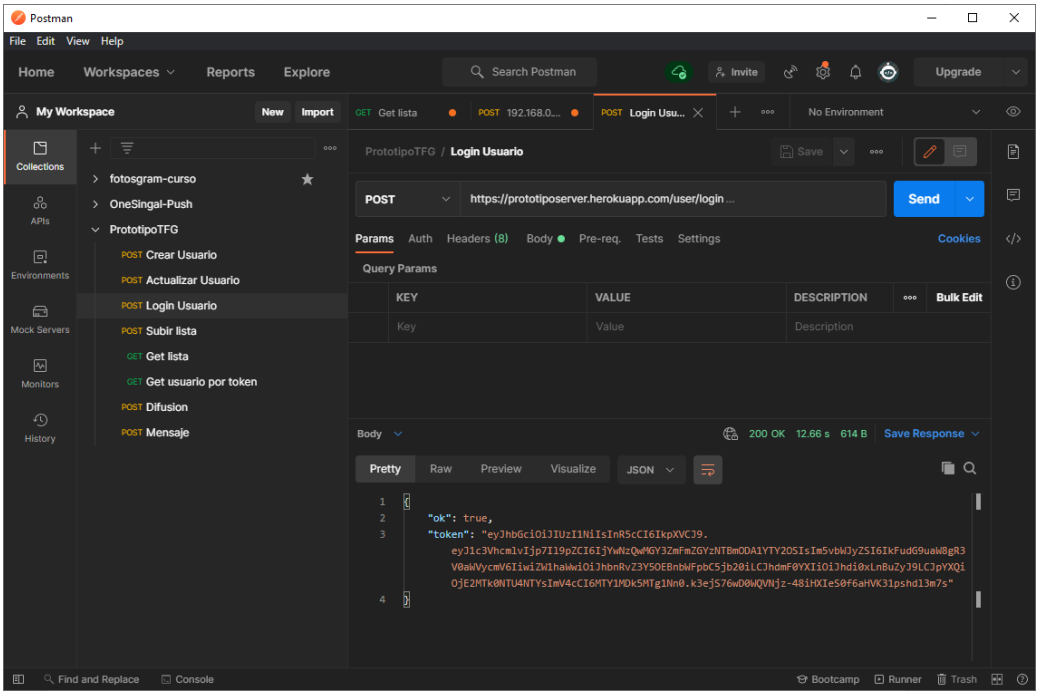


Figura 22 Interfaz de Postman

3.2.3 Git + Github

Figura 23 Logo de Git + GitHub

Git [18] es un software de control de versiones de código, el cual permite llevar un seguimiento de los cambios que ha ido sufriendo el código fuente de un proyecto a lo largo de su vida, facilitando así la recuperación en caso de errores del código y el conocimiento de los cambios realizados en cada versión de éste.

GitHub [19] es un portal creado por Microsoft en 2018 para el alojamiento de código fuente en internet, éste en combinación con el sistema de control de versiones Git permite llevar un seguimiento de la vida del proyecto de forma centralizada desde el portal web de GitHub. El tener centralizado el código fuente en la web permite acceder a éste desde cualquier dispositivo con acceso a internet, agilizando y facilitando el desarrollo del mismo proyecto desde distintas máquinas.

3.2.4 Ionic CLI

Figura 24 Logo de Ionic CLI

Ionic CLI [20] es el intérprete por líneas de comando de Ionic que contiene herramientas útiles para la creación, depuración y exportación de aplicaciones Ionic.

Gracias al CLI de Ionic podemos generar de manera automática proyectos sobre plantillas para agilizar el proceso de creación de nuestra aplicación. También podemos correr nuestro proyecto para poder mostrarlo y depurarlo en el navegador predeterminado de nuestro ordenador con el comando “ionic serve”.

En caso de querer probar nuestra aplicación en un dispositivo tan solo tendremos que conectar éste a nuestro ordenador o tener cargado un dispositivo emulado y ejecutar los comandos “ionic cordova prepare android (o ios)” y “ionic cordova run android (o ios)”

3.2.5 Google Chrome

Figura 25 Logo de Chrome

Google Chrome es un navegador web basado en Chromium y desarrollado por Google. Mediante Google Chrome no solo podremos cargar y compilar nuestras aplicaciones web, sino que también mediante su apartado “inspeccionar elemento” podemos observar el código fuente, realizar cambios locales en el código de la aplicación web, así como depurar el código de éste y ver la consola.

Las posibilidades de “inspeccionar elemento” de Chrome [21] son muy amplias y permiten interactuar con las aplicaciones, así como simular algunos sensores del dispositivo para realizar labores de depuración y testeo.

3.2.6 Heroku

Figura 26 Logo de Heroku

Heroku [22] es una plataforma como servicio (PaaS) en la nube, compatible con múltiples lenguajes de programación que ha ganado mucha popularidad por su facilidad de uso, versatilidad y compatibilidad con GitHub. Fue una de las primeras plataformas de computación en la nube ya que lleva activa desde 2007

Se encarga de resolver el problema del despliegue online de una aplicación, proporcionando la posibilidad de manejar y configurar los servidores, así como escalarlos contratando los servicios necesarios en función de las necesidades de tráfico.

Gracias al servicio gratuito de Heroku pequeños desarrolladores pueden subir sus proyectos a la plataforma de forma gratuita, pero con un modo reposo cuando pasan 30 min de la última llamada al servicio colgado y un uso de 512MB de RAM en sus servidores.

3.2.7 MongoDB Compass + Atlas

Figura 27 Logo de MongoDB Compass

Compass [23] es el entrono oficial de MongoDB, el cual nos permite explorar la estructura de colecciones que tenemos almacenadas en nuestra base de datos. Con este entorno podemos observar la estructura de nuestra base de datos, así como los campos que componen los documentos de ésta.

Figura 28 Logo de MongoDB Atlas

MongoDB Atlas [24] es la plataforma oficial de base de datos como servicio de MongoDB, ésta permite desplegar de manera online nuestra base de datos para poder acceder a ella desde un PaaS como Heroku.

MongoDB Atlas oes escalable permitiendo al usuario contratar los servicios de almacenamiento necesarios para su base de datos. Atlas nos ofrece también un plan gratuito con un almacenamiento máximo de 512MB.

Desde MongoDB Compass podemos conectarnos y gestionar nuestra base de datos subida en MongoDB Atlas.

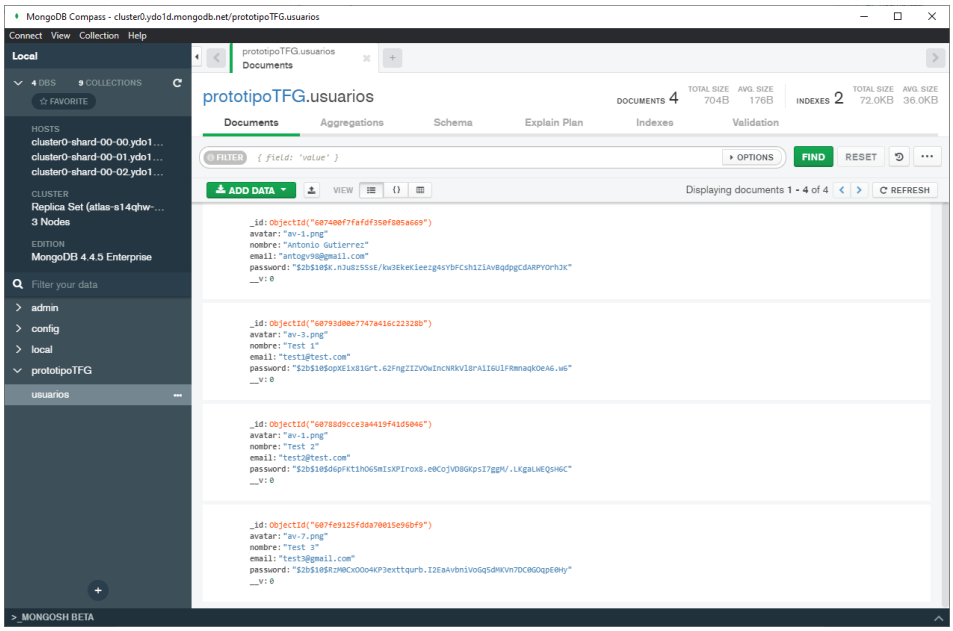


Figura 29 Interfaz de MongoDB Compass conectado a Atlas

4. DESARROLLO DEL PROYECTO

4.1 Análisis

Para el desarrollo de un software solido es muy importante realizar un correcto análisis previo de éste. Debido a un mal análisis de los requisitos el desarrollo puede verse afectado muy negativamente. Un análisis correcto permite establecer con exactitud los requisitos del sistema, lo que es esencial para un desarrollo exitoso del mismo.

4.1.1 Especificación preliminar

Se requiere de un sistema híbrido para plataformas móviles que trabaje en conjunto con un servidor REST que cumplan las siguientes características:

- El sistema se compondrá de un componente cliente y otro servidor. La parte del cliente será desarrollada con el framework Ionic y la del servidor será desarrollada como un API REST utilizando Node.js y MongoDB.
- El sistema permitirá a los usuarios registrarse para acceder a las funciones de éste.
- Para los usuarios ya registrados el sistema permitirá iniciar sesión para acceder a las funciones de éste.
- La autentificación será obligatoria para el uso del sistema.
- Es sistema mostrará tras la autentificación la página principal donde se encontrará la lista de clientes ordenada por distancia entre su dirección y la del usuario.
- Cada cliente tendrá una serie de opciones asociadas para modificar los datos de éste, cambiar su estado o eliminarlo.
- En todas las páginas donde se muestra la lista de clientes habrá una barra de búsqueda para acceder rápidamente a éste.
- Se podrán añadir clientes manualmente desde la página principal.
- Habrá una opción que active el GPS en la aplicación y las coordenadas actualizadas del usuario serán mostradas en todo momento.
- Un botón de acceso rápido permitirá editar la distancia a la que salte la alerta de proximidad con el cliente.
- Una vez alcanzada una distancia mínima se cambiará el estado de éste y se realizará una petición al servidor, el cual enviará un mensaje por WhatsApp al teléfono del cliente.
- Haciendo click en el cliente se abrirá la aplicación de Google Maps en el dispositivo con una ruta creada hasta la dirección de éste.
- Habrá una ventana de mapa donde se mostrarán el mapa centrado en la posición de usuario y con un marcador por cada cliente en la lista en su posición.
- Activando un toggle el mapa se actualizará cada pocos segundos.
- Una página para editar la lista permitirá la selección múltiple de clientes para hacer un cambio de estado a todos éstos o eliminarlos de manera ágil.
- Habrá una página de ajustes.
- Una opción para cambiar el tema del sistema entre claro y oscuro se encontrará en la página de ajustes.

- Un botón para exportar localmente la lista de clientes estará disponible desde la página de ajustes.
- Botones para la subida y descarga de la lista de clientes al servidor estarán disponibles en la página de ajustes.
- Opciones para eliminar el usuario o cerrar sesión en la aplicación también estarán disponibles en la página de ajustes.
- Se incluirá un botón en los ajustes que envíe una petición al servidor para que éste envíe un mensaje por Whatsapp a toda la lista de clientes.
- El estado de los clientes que respondan a dicho mensaje se verá modificado en función de su respuesta por Whatsapp a través del servidor.

4.1.2 Objetivos

A continuación, se detallan los objetivos o requisitos generales que se esperan alcanzar cuando el desarrollo del sistema se haya concluido.

OBJ-001	**Gestionar usuarios**
Descripción	El sistema deberá ser capaz de gestionar autentificación y registro de usuarios, así como de cerrado de sesión o eliminación de éstos
Estabilidad	Alta
Comentarios	Ninguno

OBJ-002	**Gestionar lista de clientes con el servidor**
Descripción	El sistema deberá ser capaz de gestionar la lista de clientes de forma local en la app para dispositivo móvil y de forma online y comunicada con el cliente, mediante conexiones con el servidor
Estabilidad	Alta
Comentarios	Ninguno

OBJ-003	**Gestionar estados de los clientes mediante GPS**
Descripción	El sistema deberá ser capaz de modificar automáticamente el estado del cliente en función de su distancia con el usuario, siempre que éste no esté activo
Estabilidad	Alta
Comentarios	Ninguno

OBJ-004	**Consultar mapa**
Descripción	El sistema deberá ser capaz de mostrar un mapa centrado en la posición GPS del usuario y con un marcador por cada cliente, situados en sus posiciones correspondiente
Estabilidad	Alta
Comentarios	Ninguno

OBJ-005	Gestionar propiedades del sistema
Descripción	El sistema deberá ser capaz de gestionar ciertas propiedades de éste por elección del usuario (estado GPS y tema).
Estabilidad	Alta
Comentarios	Ninguno

4.1.3 Actores

El sistema constará de dos actores con sus respectivos roles, aunque prácticamente todo el sistema solo será accesible para uno de ellos ya que para no existirán funciones más allá del registro y la autenticación para el actor y rol "Usuario no registrado" y la respuesta por mensajes para el actor y rol "Cliente".

ACT-001	Usuario no autenticado
Rol	ROL-001
Descripción	Este actor representa a cualquier usuario que aún no ha iniciado sesión en la aplicación
Comentarios	Ninguno

ACT-002	Usuario autenticado
Rol	ROL-002
Descripción	Este actor representa a cualquier usuario que ya ha iniciado sesión en la aplicación
Comentarios	Ninguno

ACT-003	Cliente
Rol	ROL-003
Descripción	Este actor representa a cualquier cliente que quiera hacer uso del servicio de avisos por Whatsapp
Comentarios	Ninguno

ROL-001	Usuario no autenticado
Actor	ACT-001
Descripción	Accede al sistema para autentificarse o registrarse en él
Requisitos asociados	RF-001 y RF-002
Comentarios	Ninguno

ROL-002	Usuario autenticado
Actor	ACT-002
Descripción	Accede al sistema para gestionar sus repartos del día.
Requisitos asociados	RF-003, RF-004, RF-005, RF-006, RF-007, RF-008, RF-009, RF-010, RF-011, RF-012, RF-013, RF-014, RF-015, RF-016, RF-017, RF-018, RF-019, RF-020, RF-021, RF-022, RF-023, RF-024, RF-025, RF-026, RF-027, RF-028, RF-029, RF-030, RF-031 y RF-032
Comentarios	Ninguno

ROL-003	Cliente
Actor	ACT-003
Descripción	Acepta al sistema para recibir notificaciones y modificar su entrega
Requisitos asociados	RF-014, RF-029, RF-030 y RF-033
Comentarios	Ninguno

4.1.4 Requisitos de información

Éstos serán los requisitos de almacenamiento de información (requisito de información-RI) que han sido identificados y las restricciones asociadas a estos requisitos (condición de requisito de información-RRI).

RI-001	Información sobre usuarios
Objetivos asociados	OBJ-001
Requisitos asociados	RF-001, RF-002, RF-010, RF-011, RF-012, RF-013, RF-014, RF-015, RF-016, RF-026, RF-027, RF-028, RF-029, RF-030, RF-031 y RF-032
Descripción	El sistema almacenará cierta información de los usuarios
Datos específicos	• Identificación del usuario • Nombre • Email • Contraseña • Avatar
Comentarios	Ninguno

CRI-001	Unicidad de los usuarios
Objetivos asociados	OBJ-001
Requisitos asociados	RI-001
Descripción	Los usuarios deben de ser únicos, siendo el campo distintivo el email de éstos. Un usuario no puede registrarse con un email ya registrado.
Comentarios	Ninguno

RI-002	Información sobre las propiedades del sistema
Objetivos asociados	OBJ-005
Requisitos asociados	RF-011, RF-024 y RF-025
Descripción	El sistema almacenará información sobre las propiedades de éste
Datos específicos	• Estado del GPS • Tema del sistema
Comentarios	Ninguno

RI-003	**Información sobre clientes**
Objetivos asociados	OBJ-002, OBJ-003 y OBJ-004
Requisitos asociados	RF-002, RF-003, RF-004, RF-005, RF-006, RF-007, RF-008, RF-009, RF-010, RF-012, RF-013, RF-014, RF-015, RF-016, RF-017, RF-018, RF-019, RF-020, RF-021, RF-022, RF-023, RF-026, RF-027, RF-028, RF-029, RF-030, RF-033 y RF-034
Descripción	El sistema almacenará cierta información de los clientes
Datos específicos	• Nombre • Dirección • Teléfono • Estado • Coordenadas • Distancia • Selección • Información adicional • Notificaciones activadas
Comentarios	Ninguno

4.1.5 Requisitos funcionales

A continuación, se muestran los requisitos relacionados con las funcionalidades que se pretende que el sistema sea capaz de realizar tras su desarrollo.

Todos los RF a excepción del 1 y 2 requieren como precondición haber iniciado sesión.

RF-001	**Registrarse**	
Objetivos asociados	OBJ-001	
Requisitos asociados	RI-001	
Descripción	Un usuario no autenticado (ACT-001) solicita registrarse en el sistema	
Precondición	El usuario no debe haber iniciado sesión	
Secuencia normal	**Paso**	**Acción**
	1	El usuario solicita a la aplicación comenzar el proceso de registro
	2	La aplicación solicita los datos del usuario
	3	El usuario introduce los datos requeridos
	4	La aplicación almacena los datos del usuario
	5	La aplicación inicia sesión al usuario nuevo
Postcondición	La aplicación registra y autentifica al nuevo usuario	
Excepciones	**Paso**	**Acción**
	4	Si el email proporcionado por el usuario ya se encuentra en la base de datos, el registro no se completa y se muestra un mensaje al usuario
Comentarios	Ninguno	

RF-002	**Iniciar sesión**	
Objetivos asociados	OBJ-001	
Requisitos asociados	RI-001	
Descripción	Un usuario no autenticado (ACT-001) solicita iniciar sesión en el sistema	
Precondición	El usuario no debe haber iniciado sesión	
Secuencia normal	**Paso**	**Acción**
	1	El usuario solicita a la aplicación comenzar el proceso de inicio de sesión
	2	La aplicación solicita los datos del usuario
	3	El usuario introduce los datos requeridos
	4	La aplicación comprueba los datos del usuario
	5	La aplicación inicia sesión al usuario nuevo usuario
Postcondición	El usuario inicia sesión y puede ejecutar la funcionalidad relativa a su rol	
Excepciones	**Paso**	**Acción**
	4	Si los datos introducidos por el usuario no se encuentran en la base de datos, el sistema muestra un mensaje al usuario
Comentarios	Ninguno	

RF-003	**Mostrar lista ordenada de clientes**	
Objetivos asociados	OBJ-002	
Requisitos asociados	RI-003	
Descripción	La aplicación muestra la lista de clientes con la información asociada a cada cliente	
Precondición	El usuario debe de haber añadido clientes manualmente o descargado su lista del servidor	
Secuencia normal	**Paso**	**Acción**
	1	La aplicación carga la lista de clientes del usuario
	2	La aplicación muestra la lista
Postcondición	Se muestra la lista	
Excepciones	**Paso**	**Acción**
	1	Si no existen clientes en la lista, ésta se mostrará vacía
Comentarios	Ninguno	

RF-004	**Calcular coordenadas del cliente**	
Objetivos asociados	OBJ-002 y OBJ-004	
Requisitos asociados	RI-003	
Descripción	La aplicación calcula las coordenadas geográficas asociadas a la dirección del cliente	
Precondición	El cliente debe de haber proporcionado una dirección válida	
Secuencia normal	**Paso**	**Acción**
	1	La aplicación obtiene las coordenadas de cada cliente
	2	La aplicación muestra las coordenadas entre los datos del cliente
Postcondición	Se muestran las coordenadas de cada cliente en la lista	
Excepciones	**Paso**	**Acción**
	1	Si no se puede obtener las coordenadas se muestra un texto de error
Comentarios	Ninguno	

RF-005	**Buscar cliente en la lista**	
Objetivos asociados	OBJ-002	
Requisitos asociados	RI-003	
Descripción	La aplicación desde una barra de búsqueda permite filtrar la lista de clientes por el nombre	
Precondición	La lista no debe de estar vacía y el nombre buscado ha de estar en ella	
Secuencia normal	**Paso**	**Acción**
	1	El usuario introduce el nombre del cliente a buscar en la barra de búsqueda
	2	La aplicación filtra la lista cargada por el nombre introducido
	3	La aplicación muestra la lista filtrada
Postcondición	Se muestra la lista de clientes que coinciden con el nombre introducido	
Excepciones	**Paso**	**Acción**
	2	En caso de no encontrar ningún cliente con ese nombre, la aplicación muestra una lista vacía
Comentarios	Ninguno	

RF-006	**Modificar datos del cliente**	
Objetivos asociados	OBJ-002	
Requisitos asociados	RI-003	
Descripción	La aplicación permite cambiar los datos del cliente (nombre, dirección, teléfono e información adicional)	
Precondición	Debe de existir al menos un cliente en la lista de clientes para poder modificar su información	
Secuencia normal	**Paso**	**Acción**
	1	El usuario selecciona la opción de modificación de datos de un cliente
	2	La aplicación muestra un formulario prellenado con los datos actuales del cliente
	3	El usuario introduce los datos que quiere modificar
	4	El usuario confirma los nuevos datos
	5	La aplicación guarda los nuevos datos asociados al cliente
	6	La aplicación muestra la lista con los nuevos datos del cliente
Postcondición	Se muestra la lista con los datos del cliente modificados	
Excepciones	**Paso**	**Acción**
	4	Si se cancela la confirmación se mantienen los datos previos a la modificación
Comentarios	Ninguno	

RF-007	**Modificar estado del cliente manualmente**	
Objetivos asociados	OBJ-002	
Requisitos asociados	RI-003	
Descripción	La aplicación permite cambiar el estado de un cliente individualmente (inactivo, activo, completado y cancelado)	
Precondición	Debe de existir al menos un cliente en la lista de clientes para poder modificar su estado	
Secuencia normal	**Paso**	**Acción**
	1	El usuario selecciona la opción de modificación del estado del cliente
	2	La aplicación modifica internamente el estado del cliente
	3	La aplicación muestra el nuevo identificador de estado del cliente en la lista
Postcondición	Se muestra la lista con el estado del cliente modificado	
Excepciones	**Paso**	**Acción**
Comentarios	Ninguno	

RF-008	**Eliminar cliente**	
Objetivos asociados	OBJ-002	
Requisitos asociados	RI-003	
Descripción	La aplicación permite eliminar clientes individualmente	
Precondición	Debe de existir al menos un cliente en la lista de clientes para poder ser eliminado	
Secuencia normal	**Paso**	**Paso**
	1	El usuario selecciona la opción eliminar del cliente
	2	La aplicación modifica internamente la lista de clientes
	3	La aplicación muestra la nueva lista sin el cliente eliminado
Postcondición	Se muestra la lista sin el cliente eliminado	
Excepciones	**Paso**	**Acción**
Comentarios	Ninguno	

RF-009	**Abrir ruta a cliente en Google Maps**	
Objetivos asociados	OBJ-002	
Requisitos asociados	RI-003	
Descripción	La aplicación permite abrir la aplicación de Google Maps del dispositivo con una ruta precargada hacia el cliente	
Precondición	Debe de existir al menos un cliente en la lista de clientes para poder abrir su ruta en Google Maps	
Secuencia normal	**Paso**	**Acción**
	1	El usuario selecciona el cliente que quiere para iniciar su ruta
	2	La aplicación modifica carga un enlace a Google Maps con las coordenadas del cliente seleccionado
	3	La aplicación hace al dispositivo navegar hasta la aplicación de Google Maps en el punto correspondiente a las coordenadas del cliente
Postcondición	Se abre la aplicación de Google Maps con la ruta hasta el cliente	
Excepciones	**Paso**	**Acción**
	2	Si las coordenadas no se había podido calcular correctamente la aplicación de Google Maps se abrirá sin ninguna ruta
Comentarios	Ninguno	

RF-010	**Cambiar distancia de alerta**	
Objetivos asociados	OBJ-003	
Requisitos asociados	RI-001 y RI-003	
Descripción	La aplicación permite modificar la distancia a la que cambia el estado del cliente y le envía un mensaje	
Precondición	El valor introducido debe de ser un número entero o decimal positivo	
Secuencia normal	**Paso**	**Acción**
	1	El usuario selecciona la opción de modificación de distancia de alerta
	2	La aplicación muestra un formulario prellenado con la distancia de alerta previa
	3	El cliente introduce la nueva distancia de alerta
	4	El cliente confirma la nueva distancia
	5	La aplicación guarda la nueva distancia internamente
Postcondición	Se modifica la distancia a la que cambiará el estado y se enviará el mensaje al cliente	
Excepciones	**Paso**	**Acción**
Comentarios	Ninguno	

RF-011	**Activar-Desactivar GPS**	
Objetivos asociados	OBJ-003 y OBJ-005	
Requisitos asociados	RI-001 y RI-002	
Descripción	La aplicación permite activar o desactivar la localización GPS	
Precondición	Se debe dar permisos a la aplicación para la gestión de la localización GPS	
Secuencia normal	**Paso**	**Acción**
	1	El usuario selecciona la opción de cambiar el estado GPS
	2	La aplicación modifica internamente el estado del GPS
	3	La aplicación muestra el indicador del nuevo estado del GPS
Postcondición	Se muestra el nuevo estado del GPS	
Excepciones	**Paso**	**Acción**
Comentarios	Ninguno	

RF-012	**Calcular distancia**	
Objetivos asociados	OBJ-003	
Requisitos asociados	RI-001 y RI-003	
Descripción	La aplicación calcula la distancia entre las coordenadas de los clientes y del usuario	
Precondición	Se debe haber podido calcular las coordenadas del cliente, el estado del cliente debe estar en activo y el estado del GPS debe estar activo	
Secuencia normal	**Paso**	**Acción**
	1	La aplicación obtiene las coordenadas geográficas del usuario
	2	La aplicación calcula la distancia en km entre las coordenadas del usuario y de los clientes activos
	3	La aplicación muestra la distancia con cada cliente en la lista
Postcondición	Se muestra la distancia con cada cliente activo en la lista de clientes	
Excepciones	**Paso**	**Acción**
Comentarios	Ninguno	

RF-013	**Cambiar estado del cliente automáticamente**	
Objetivos asociados	OBJ-002 y OBJ-003	
Requisitos asociados	RI-001 y RI-003	
Descripción	La aplicación cambia el estado del cliente activo si se alcanza la distancia de alerta con el usuario	
Precondición	Se debe haber recalculado la distancia entre cliente y usuario y que esta sea menor que la distancia de alerta	
Secuencia normal	**Paso**	**Acción**
	1	La aplicación calcula la distancia entre usuario y cliente
	2	La aplicación comprueba si la distancia es menor que la distancia de alerta
	3	La aplicación cambia internamente el estado del cliente (nuevo estado: en rango)
	4	La aplicación muestra en la lista el indicativo del nuevo estado
Postcondición	Se muestra la lista con el estado del cliente modificado	
Excepciones	**Paso**	**Acción**
	2	Si no se cumple la condición el estado no es modificado
Comentarios	Ninguno	

RF-014	**Enviar mensaje de proximidad al cliente**	
Objetivos asociados	OBJ-002 y OBJ-003	
Requisitos asociados	RI-001 y RI-003	
Descripción	La aplicación envía una petición al servidor para enviar un mensaje de Whatsapp al cliente en rango	
Precondición	Se debe de haber establecido el estado del cliente como “en rango”	
Secuencia normal	**Paso**	**Acción**
	1	La aplicación envía una petición al servidor para enviar un mensaje al cliente
	2	El servidor llama a la API de Twilio con los datos del cliente y el usuario que envía la petición
	3	La API envía un Whatsapp al teléfono del cliente
	4	La aplicación modifica el estado del cliente (nuevo estado: mensaje enviado)
	5	La aplicación muestra en la lista el indicativo del nuevo estado
Postcondición	Se envía un mensaje al cliente en rango	
Excepciones	**Paso**	**Acción**
	3	Si el teléfono del cliente era erróneo o no estaba dado de alta en Twilio el mensaje no es enviado
Comentarios	Ninguno	

RF-015	**Mostrar mapa**	
Objetivos asociados	OBJ-002 y OBJ-004	
Requisitos asociados	RI-001 y RI-003	
Descripción	La aplicación muestra un mapa de Google Maps embebido, centrado en la posición del usuario y con un marcador por cada cliente de la lista	
Precondición	La posición GPS del usuario debe de haber sido obtenida al menos una vez	
Secuencia normal	**Paso**	**Acción**
	1	La aplicación llama a la API de Google Maps para generar el mapa centrado en el usuario
	2	La aplicación recorre la lista de clientes y crea un marcador por cada cliente
	3	La aplicación muestra el mapa
Postcondición	Se muestra el mapa centrado en el usuario y con los marcadores añadidos	
Excepciones	**Paso**	**Acción**
	1	Si no se ha obtenido la posición del usuario el mapa se centra en Madrid
Comentarios	Ninguno	

RF-016	**Actualizar mapa automáticamente**	
Objetivos asociados	OBJ-002 y OBJ-004	
Requisitos asociados	RI-001 y RI-003	
Descripción	La aplicación actualiza y vuelve a generar el mapa cada 3 segundos	
Precondición	El mapa debe de poder ser generado sin actualizaciones	
Secuencia normal	**Paso**	**Acción**
	1	El usuario activa el toggle de actualización automática del mapa
	2	La aplicación activa un temporizado de 3 segundos
	3	La aplicación genera un nuevo mapa y unos nuevos marcadores
	4	La aplicación refresca la pantalla, mostrando el nuevo mapa
Postcondición	Se muestra un nuevo mapa cada 3 segundos	
Excepciones	**Paso**	**Acción**
Comentarios	Ninguno	

RF-017	**Añadir cliente**	
Objetivos asociados	OBJ-002	
Requisitos asociados	RI-003	
Descripción	El usuario añade manualmente un cliente introduciendo sus datos en un formulario	
Precondición	Se debe poseer los datos de un nuevo cliente	
Secuencia normal	**Paso**	**Acción**
	1	El usuario selecciona la opción de añadir cliente
	2	La aplicación muestra un formulario con los campos necesarios para crear un nuevo cliente
	3	El usuario introduce los datos del nuevo cliente
	4	El usuario confirma los nuevos datos
	5	La aplicación guarda el nuevo cliente dentro de la lista
	6	La aplicación muestra la lista con el nuevo cliente incluido
Postcondición	Se muestra la lista con el nuevo cliente	
Excepciones	**Paso**	**Acción**
	4	Si se cancela la confirmación no es añadido ningún cliente
Comentarios	Ninguno	

RF-018	**Mostar lista clientes seleccionable**	
Objetivos asociados	OBJ-002	
Requisitos asociados	RI-003	
Descripción	La aplicación muestra una lista seleccionable de los clientes	
Precondición	El usuario debe de haber añadido clientes manualmente o descargado su lista del servidor.	
Secuencia normal	**Paso**	**Acción**
	1	El usuario selecciona la opción de mostrar lista de edición
	2	La aplicación carga la lista de clientes
	3	La aplicación muestra la lista de clientes con su atributo seleccionable
Postcondición	Se muestra la lista seleccionable	
Excepciones	**Paso**	**Acción**
	1	Si no existen clientes en la lista, ésta se mostrará vacía
Comentarios	Ninguno	

RF-019	**Seleccionar-Deseleccionar cliente**	
Objetivos asociados	OBJ-002	
Requisitos asociados	RI-003	
Descripción	El usuario podrá marcar como seleccionado o no a los clientes individualmente	
Precondición	Se debe de haber podido mostrar la lista de clientes seleccionable	
Secuencia normal	**Paso**	**Acción**
	1	El usuario elije un cliente y cambia su valor selección
	2	La aplicación cambiar el valor selección del cliente
	3	La aplicación muestra al cliente en la lista como seleccionado-deseleccionado
Postcondición	Se selecciona-deselecciona un cliente individualmente	
Excepciones	**Paso**	**Acción**
Comentarios	Ninguno	

RF-020	**Seleccionar-Deseleccionar todos los clientes**	
Objetivos asociados	OBJ-002	
Requisitos asociados	RI-003	
Descripción	El usuario podrá marcar como seleccionado o no a todos los clientes al mismo tiempo	
Precondición	Se debe de haber podido mostrar la lista de clientes seleccionable	
Secuencia normal	**Paso**	**Acción**
	1	El usuario marca la opción de cambiar selección a todos los clientes
	2	La aplicación cambiar el valor de selección de todos los clientes para que sea el mismo
	3	La aplicación muestra al cliente en la lista clientes con el valor de selección correspondiente
Postcondición	Se cambia el valor de selección de todos los clientes ha seleccionado o no seleccionado	
Excepciones	**Paso**	**Acción**
Comentarios	Si los clientes se encontraban en distinto estado de selección individualmente se unifican todos con el mismo estado de selección	

RF-021	**Cambiar estado de clientes seleccionados**	
Objetivos asociados	OBJ-002	
Requisitos asociados	RI-003	
Descripción	La aplicación podrá cambiar el estado de todos los clientes seleccionados	
Precondición	Se debe de haber seleccionado al menos un cliente	
Secuencia normal	**Paso**	**Acción**
	1	El usuario hace click en la opción de cambiar estado de los seleccionados
	2	La aplicación modifica internamente el estado de los clientes seleccionados
	3	La aplicación muestra el indicativo del estado de los clientes modificados
Postcondición	Se cambia el estado de los usuarios seleccionados	
Excepciones	**Paso**	**Acción**
Comentarios	Si los clientes seleccionados se encontraban en distinto estado se unifica el estado de todos al mismo	

RF-022	**Eliminar clientes seleccionados**	
Objetivos asociados	OBJ-002	
Requisitos asociados	RI-003	
Descripción	La aplicación podrá eliminar todos los clientes seleccionados	
Precondición	Se debe de haber seleccionado al menos un cliente	
Secuencia normal	**Paso**	**Acción**
	1	El usuario hace click en la opción de eliminar los clientes seleccionados
	2	La aplicación elimina de la lista de clientes a los clientes seleccionados
	3	La aplicación muestra la lista de clientes modificada
Postcondición	Se eliminan los clientes seleccionados de la lista de clientes	
Excepciones	**Paso**	**Acción**
Comentarios	Ninguno	

RF-023	**Subir al inicio de la lista de clientes**	
Objetivos asociados	OBJ-002	
Requisitos asociados	RI-003	
Descripción	La aplicación podrá subir al inicio de la lista desde un botón sin necesidad de hacer scroll	
Precondición	La lista debe de superar el largo de la pantalla del dispositivo	
Secuencia normal	**Paso**	**Acción**
	1	El usuario selecciona la opción de scroll al inicio
	2	La aplicación realiza un scroll automático hasta el inicio del contenido de la página
Postcondición	Se vuelve al inicio de la página	
Excepciones	**Paso**	**Acción**
Comentarios	Ninguno	

RF-024	**Mostrar lista de ajustes**	
Objetivos asociados	OBJ-005	
Requisitos asociados	RI-002	
Descripción	La aplicación mostrará una lista de ajustes de la aplicación	
Precondición		
Secuencia normal	**Paso**	**Acción**
	1	El usuario selecciona la opción de mostrar lista de ajustes
	2	La aplicación carga la lista de ajustes
	3	La aplicación muestra la lista de ajustes de la aplicación
Postcondición	Se muestra la lista de ajustes	
Excepciones	**Paso**	**Acción**
Comentarios	Ninguno	

RF-025	**Cambiar tema del sistema**	
Objetivos asociados	OBJ-005	
Requisitos asociados	RI-002	
Descripción	La aplicación podrá cambiar el tema de la aplicación entre oscuro y claro	
Precondición		
Secuencia normal	**Paso**	**Acción**
	1	El usuario selecciona la opción de cambiar de tema
	2	La aplicación cambia el valor interno de su propiedad darkOn
	3	La aplicación muestra el sistema con el tema seleccionado
Postcondición	Se modifica el tema de la aplicación	
Excepciones	**Paso**	**Acción**
Comentarios	Ninguno	

RF-026	**Exportar lista de clientes localmente**	
Objetivos asociados	OBJ-002	
Requisitos asociados	RI-001 y RI-003	
Descripción	La aplicación podrá exportar localmente la lista de clientes y compartirla por distintos canales	
Precondición	La lista no debe estar vacía	
Secuencia normal	**Paso**	**Acción**
	1	El usuario selecciona la opción de exportar lista localmente
	2	La aplicación genera un archivo csv con la información de un cliente en cada fila y el id del usuario como nombre
	3	La aplicación ofrece la opción de guardar localmente el archivo o compartirlo por varias vías
Postcondición	Se exporta localmente la lista de clientes y se da la opción de enviarla	
Excepciones	**Paso**	**Acción**
Comentarios	Ninguno	

RF-027	**Subir lista de clientes al servidor**	
Objetivos asociados	OBJ-002	
Requisitos asociados	RI-001 y RI-003	
Descripción	La aplicación podrá subir la lista de clientes al servidor	
Precondición	La lista no debe estar vacía	
Secuencia normal	**Paso**	**Acción**
	1	El usuario selecciona la opción de subir lista al servidor
	2	La aplicación genera un archivo csv con la información de un cliente en cada fila y el id del usuario como nombre
	3	La aplicación realiza un post http al servidor con la lista
	4	El servidor coloca el archivo en el directorio destinado a ese cliente, sobrescribiendo si había una anterior.
Postcondición	Se sube la lista de clientes al servidor	
Excepciones	**Paso**	**Acción**
Comentarios	Ninguno	

RF-028	**Descargar lista de clientes del servidor**	
Objetivos asociados	OBJ-002	
Requisitos asociados	RI-001 y RI-003	
Descripción	La aplicación podrá descargar la lista de clientes al servidor	
Precondición	Debe de haber una lista subida al servidor para dicho usuario	
Secuencia normal	**Paso**	**Acción**
	1	El usuario selecciona la opción de descargar lista al servidor
	2	La aplicación realiza una petición http al servidor
	3	El servidor envía la lista, en formato csv, correspondiente al usuario
	4	La aplicación lee el archivo csv y lo guarda en la lista de clientes
Postcondición	La lista de clientes de la aplicación es rellenada	
Excepciones	**Paso**	**Acción**
	3	La debe de existir una lista para el usuario
Comentarios	Ninguno	

RF-029	**Enviar difusión a todos los clientes**	
Objetivos asociados	OBJ-002	
Requisitos asociados	RI-001 y RI-003	
Descripción	La aplicación podrá enviar una petición al servidor para enviar un mensaje de Whatsapp a todos los clientes	
Precondición	La lista no debe estar vacía	
Secuencia normal	**Paso**	**Acción**
	1	La aplicación envía una petición al servidor para enviar un mensaje a todos los clientes
	2	El servidor recorre la lista de clientes del usuario
	3	El servidor llama a la API de Twilio con los datos del usuario y el cliente que está siendo recorrido de la lista
	4	La API envía un Whatsapp al teléfono del cliente
Postcondición	Se envía un mensaje a todos los clientes de la lista	
Excepciones	**Paso**	**Acción**
	3	Si el teléfono del cliente era erróneo o no estaba dado de alta en Twilio el mensaje no es enviado
Comentarios	Ninguno	

RF-030	**Procesar respuesta del cliente**	
Objetivos asociados	OBJ-002	
Requisitos asociados	RI-001 y RI-003	
Descripción	El servidor podrá procesar ciertas respuestas de los clientes para modificar su estado	
Precondición	El cliente debe de responder un mensaje determinado	
Secuencia normal	**Paso**	**Acción**
	1	El cliente envía el mensaje por Whatsapp
	2	El servidor procesa la respuesta recibida y cambia el estado del cliente
	3	El servidor envía un mensaje sobre el procesamiento de su acción al cliente por Whatsapp
	4	El servidor guarda la nueva lista con el estado modificado en el directorio del usuario al que está asociado el cliente
Postcondición	Se modifica el estado del cliente en función de su respuesta por Whatsapp	
Excepciones	**Paso**	**Acción**
	2	Si la respuesta recibida no tiene el formato adecuado no es realizada ninguna acción
Comentarios	Ninguno	

RF-031	**Eliminar usuario**	
Objetivos asociados	OBJ-001	
Requisitos asociados	RI-001	
Descripción	La aplicación elimina al usuario de la base de datos del servidor	
Precondición	El usuario debe de haber iniciado sesión	
Secuencia normal	**Paso**	**Acción**
	1	El usuario selecciona la opción de eliminar usuario
	2	La aplicación muestra un mensaje de confirmación al usuario
	3	El usuario confirma la eliminación
	4	La aplicación manda una petición http al servidor
	5	El servidor elimina al usuario de la base de datos
	6	La aplicación traslada al usuario a la página de inicio de sesión
Postcondición	Se elimina el usuario de la base de datos	
Excepciones	**Paso**	**Acción**
	3	Si el usuario cancela la confirmación no se realiza ninguna acción
Comentarios	Ninguno	

RF-032	**Cerrar sesión**	
Objetivos asociados	OBJ-001	
Requisitos asociados	RI-001	
Descripción	La aplicación saca al usuario a la página de inicio de sesión y se elimina la configuración realizada en la aplicación	
Precondición	El usuario debe de haber iniciado sesión	
Secuencia normal	**Paso**	**Acción**
	1	El usuario selecciona la opción de cerrar sesión
	2	La aplicación elimina la configuración realizada en la aplicación
	3	La aplicación traslada al usuario a la página de inicio de sesión
Postcondición	La sesión y la configuración realizada en ella es reestablecida	
Excepciones	**Paso**	**Acción**
Comentarios	Ninguno	

RF-033	**Activar recepción de mensajes**	
Objetivos asociados	OBJ-002	
Requisitos asociados	RI-003	
Descripción	El cliente activará la posibilidad de recibir notificaciones vía Whatsapp	
Precondición	El cliente debe de formar parte de la lista de un usuario para recibir las notificaciones	
Secuencia normal	**Paso**	**Acción**
	1	El cliente manda un mensaje de conexión al número de Whatsapp del sistema
	2	El servidor añade el número del cliente a los números activos para enviar y recibir mensajes
Postcondición	El número del cliente se queda registrado para poder enviar y recibir mensajes	
Excepciones	**Paso**	**Acción**
Comentarios	Ninguno	

RF-034	**Enviar respuestas al servidor**	
Objetivos asociados	OBJ-002	
Requisitos asociados	RI-003	
Descripción	El cliente envía un mensaje al servidor para cambiar su estado	
Precondición	El usuario debe de haber iniciado sesión	
Secuencia normal	**Paso**	**Acción**
	1	El cliente manda un mensaje de cambio de estado al número de Whatsapp del sistema
	2	El servidor procesa la respuesta del cliente y cambia su estado temporalmente
	3	El servidor envía un mensaje de confirmación al cliente por Whatsapp
	4	El servidor guarda el nuevo estado del cliente en la lista del usuario asociado
Postcondición	El nuevo estado del cliente es guardado en la lista del usuario que hará su reparto	
Excepciones	**Paso**	**Acción**
	2	Si la respuesta no corresponde con ninguna de las posibles, no se procesará ningún cambio
Comentarios	Ninguno	

Esta es la lista de los requisitos funcionales del sistema:

- **RF-001** Registrarse
- **RF-002** Iniciar Sesión
- **RF-003** Mostrar lista ordenada de clientes
- **RF-004** Calcular coordenadas de cliente
- **RF-005** Buscar cliente en la lista
- **RF-006** Modificar datos del cliente
- **RF-007** Modificar estado del cliente manualmente
- **RF-008** Eliminar cliente
- **RF-009** Abrir ruta a cliente en Google Maps
- **RF-010** Cambiar distancia de alerta
- **RF-011** Activar-Desactivar GPS
- **RF-012** Calcular distancia
- **RF-013** Cambiar estado del cliente automáticamente
- **RF-014** Enviar mensaje al Cliente
- **RF-015** Mostrar mapa
- **RF-016** Actualizar mapa automáticamente
- **RF-017** Añadir cliente
- **RF-018** Mostrar lista de clientes seleccionable
- **RF-019** Seleccionar-Deseleccionar cliente
- **RF-020** Seleccionar-Deseleccionar todos los clientes
- **RF-021** Cambiar estado de clientes seleccionados
- **RF-022** Eliminar clientes seleccionados
- **RF-023** Subir al inicio de la lista de clientes
- **RF-024** Mostrar lista de ajustes
- **RF-025** Cambiar tema del sistema

- **RF-026** Exportar lista de clientes localmente
- **RF-027** Subir lista de clientes al servidor
- **RF-028** Descargar lista de clientes del servidor
- **RF-029** Enviar difusión a todos los clientes
- **RF-030** Procesar respuesta del cliente
- **RF-031** Eliminar usuario
- **RF-032** Cerrar sesión
- **RF-033** Activar notificaciones
- **RF-034** Enviar respuestas al servidor

4.1.6 Requisitos no funcionales

Éstos serán los requisitos no relacionados con la funcionalidad que se pretenden cumplir para incrementar la comodidad de uso y calidad del sistema.

RNF-001	**Soporte multidispositivo**
Descripción	Todos los componentes funcionales y elementos de la interfaz de usuario deben de ser utilizables, aunque se instale en dispositivos con diferente sistema operativo.
Comentarios	Por ejemplo, ciertos usuarios pueden utilizar la aplicación desde un iPhone y otros desde un Android.

RNF-002	**Fiabilidad**
Descripción	El sistema debe ser fiable y proporcionar un desempeño estable que no vaya a realizar acciones incorrectas o dejar de funcionar.
Comentarios	Por ejemplo, durante su uso no debe de bloquearse ni cerrarse inesperadamente.

RNF-003	**Privacidad**
Descripción	Las contraseñas de los usuarios que son almacenadas en la base de datos deben de ser encriptadas para garantizar su privacidad
Comentarios	Por ejemplo, si se realizase una filtración de datos la contraseña no podría ser descifrada por el intruso.

RNF-004	**Facilidad de uso**
Descripción	La interfaz de la aplicación ha de ser intuitiva haciendo que con poco aprendizaje el usuario sea capaz de utilizarla por completo.
Comentarios	Por ejemplo, una interfaz poco intuitiva podría reducir en gran medida la eficiencia con la que el usuario realiza acciones en la aplicación.

RNF-005	Componentes poco intrusivos
Descripción	Se debe mantener una relación entre el tamaño de la pantalla y la cantidad de información visual que hay en ella, ya que no debe obstruirse un componente visual importante con otro, de modo que la utilización de la aplicación sea cómoda.
Comentarios	Por ejemplo, los indicativos del estado del cliente deben de ocupar poco espacio para maximizar la cantidad de información sobre el mismo que está visible en pantalla.

RNF-006	Mapa embebido
Descripción	Se incluirá la visualización embebida del mapa del lugar donde se encuentra el usuario para facilitar la orientación de éste sin necesidad de salir de la aplicación.
Comentarios	En caso de que el usuario requiera visualizar un mapa para orientarse durante la entrega puede acceder rápidamente a éste.

RNF-007	Confirmación de acciones
Descripción	El sistema debe de requerir una confirmación del usuario cuando ciertas acciones críticas van a ser realizadas para prevenir posibles equivocaciones.
Comentarios	Por ejemplo, si el usuario desea eliminar su usuario una confirmación es mostrada para asegurarse de que no está realizando esa acción por error.

RNF-008	Tiempos de carga
Descripción	Los tiempos de carga una vez despierto el servidor de su estado reposo deben de ser cortos para que las labores de reparto sean eficaces.
Comentarios	Si los cálculos de distancia y las peticiones al servidor ser tardasen mucho tiempo podría darse el caso de que el aviso al cliente no fuera con suficiente tiempo para que éste pudiera estar prevenido.

4.1.7 Matriz de trazabilidad

Para poder dar un vistazo rápido hacia las relaciones entre objetivos, requisitos de información y requisitos funcionales se dispone de las siguientes matrices de trazabilidad:

- Matriz de trazabilidad 1: Relación entre requisitos de información y objetivos.
- Matriz de trazabilidad 2: Relación entre requisitos funcionales y objetivos.
- Matriz de trazabilidad 3: Relación entre requisitos funcionales y requisitos de información.

	RI-001	RI-002	RI-003
OBJ-001	*		
OBJ-002			*
OBJ-003	*	*	*
OBJ-004	*	*	*
OBJ-005		*	

Figura 30 Matriz de trazabilidad 1

	OBJ-001	OBJ-002	OBJ-003	OBJ-004	OBJ-005
RF-001	*				
RF-002	*				
RF-003		*			
RF-004		*		*	
RF-005		*			
RF-006		*			
RF-007		*			
RF-008		*			
RF-009		*			
RF-010			*		
RF-011			*		*
RF-012			*		
RF-013		*	*		
RF-014		*	*		
RF-015		*		*	
RF-016		*		*	
RF-017		*			
RF-018		*			
RF-019		*			
RF-020		*			
RF-021		*			
RF-022		*			
RF-023		*			
RF-024					*
RF-025					*
RF-026		*			
RF-027		*			
RF-028		*			
RF-029		*			
RF-030		*			
RF-031	*				
RF-032	*				
RF-033		*			
RF-034		*			

Figura 31 Matriz de trazabilidad 2

	RI-001	RI-002	RI-003
RF-001	*		
RF-002	*		
RF-003			*
RF-004			*
RF-005			*
RF-006			*
RF-007			*
RF-008			*
RF-009			*
RF-010	*		*
RF-011	*	*	
RF-012	*		*
RF-013	*		*
RF-014	*		*
RF-015	*		*
RF-016	*		*
RF-017			*
RF-018			*
RF-019			*
RF-020			*
RF-021			*
RF-022			*
RF-023			*
RF-024		*	
RF-025		*	
RF-026	*		*
RF-027	*		*
RF-028	*		*
RF-029	*		*
RF-030	*		*
RF-031	*		
RF-032	*		
RF-033			*
RF-034			*

Figura 32 Matriz de trazabilidad 3

4.1.8 Diagrama de casos de uso

Aquí se muestra un diagrama de caso de uso que combina la relación de los actores del sistema con los requisitos funcionales a los que éstos pueden acceder.

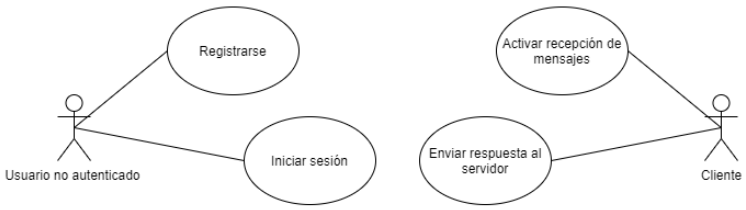


Figura 33 Casos de uso de Usuario no autenticado y Cliente

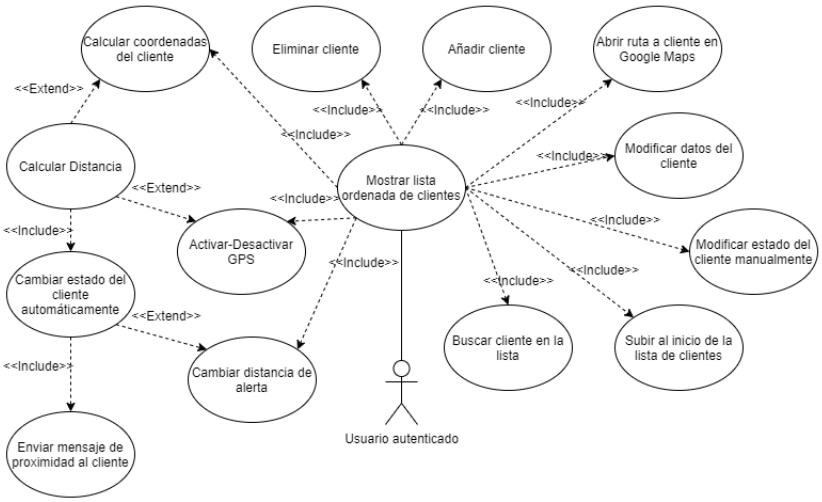


Figura 34 Casos de uso de página Inicio para Usuario autenticado

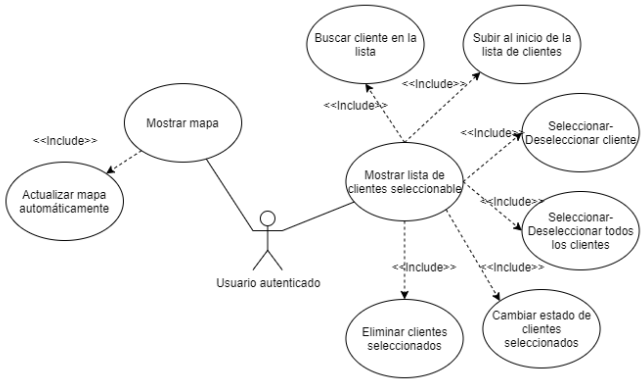


Figura 35 Casos de uso de página Mapa y Editar para Usuario autenticado

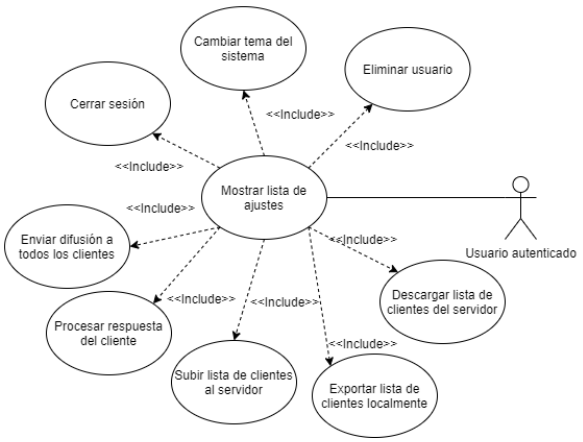


Figura 36 Casos de uso de página Ajustes para Usuario autenticado

4.2 Arquitectura de software

La arquitectura del sistema cuenta con una combinación de aplicación híbrida para dispositivo móvil, junto con una aplicación web y API.

La aplicación web a si mismo es la encargada de realizar las conexiones con la base de datos del sistema mediante el paquete “Mongoose” [25] de Node.js.

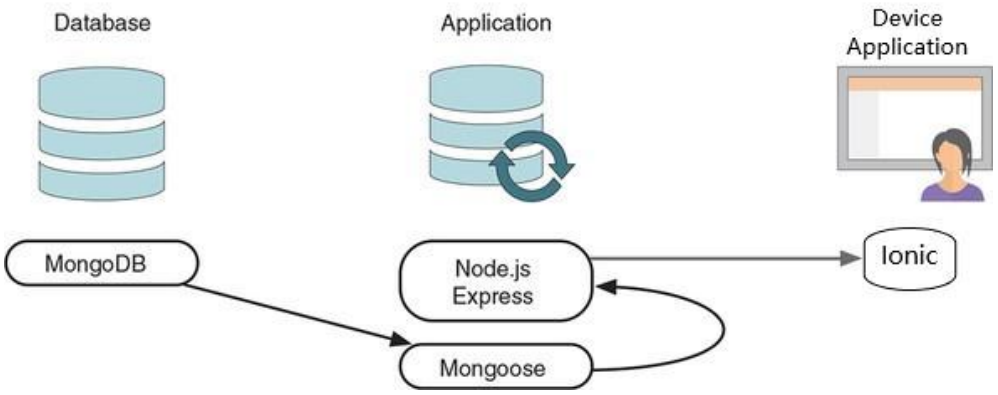


Figura 37 Relación entre Base de datos, App del servidor y App del dispositivo

Para realizar las conexiones entre la API y la aplicación móvil se utilizan peticiones HTTP con objetos embebidos con formato JSON (figura 16).

Ciertas peticiones están protegidas para no poder ser realizadas a no ser que se cumplan unos requisitos, éstas han de pasar por una validación intermedia (middleware) [26], la cual se encarga de verificar que el usuario que realiza esta petición es un usuario válido.

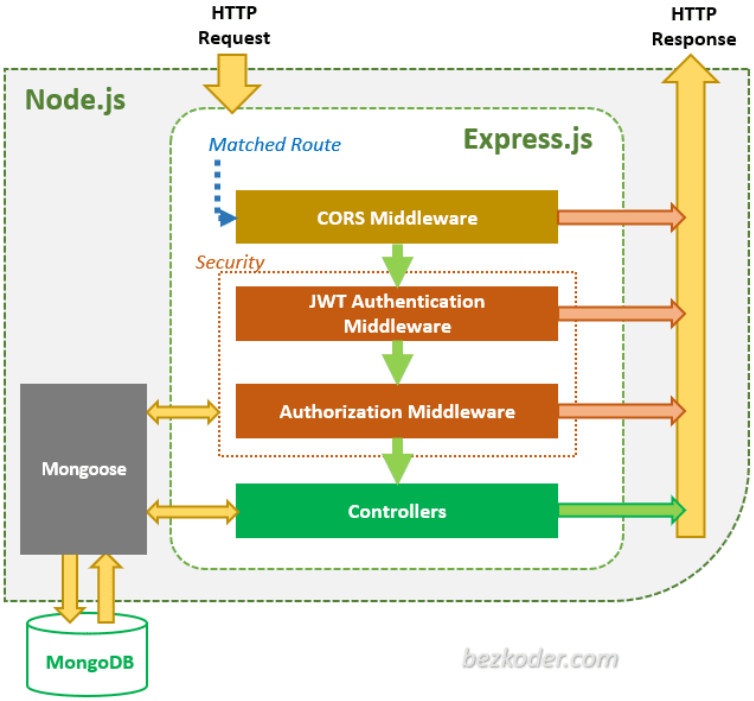


Figura 38 Secuencia de petición HTTP con middleware

4.3 Aplicación móvil

La aplicación móvil es la parte del sistema que se encuentra instalada en su dispositivo y con la que el usuario va a estar en contacto en todo momento. Debido a que hoy en día existe más de un sistema operativo con arquitectura propia para dispositivos móviles, se ha optado por la idea de crear una aplicación móvil híbrida, válida tanto para IOS como para Android. La tecnología utilizada para esto ha sido el framework de código abierto Ionic, del que hemos hablado anteriormente.

4.3.1 Patrón de diseño

El patrón de diseño que utilizan las aplicaciones de Ionic es el mismo que para las aplicaciones de Angular (ya que ionic se basa en Angular) y suele ser el MVVM (Modelo-Vista-Vista Modelo) [27], el cual separa la lógica de negocios de la interfaz. Para realizar esto se pueden distinguir tres componentes principales: Modelo, Vista y Vista Modelo.

- Modelo: Se encarga de hacer peticiones a las bases de datos alojadas de forma local en la app o remota en un servidor externo y con esto enviar o recibir información.

- Vista: Es la parte correspondiente con la presentación de los datos al usuario, en este caso correspondería con el código HTML, el cual nos muestra la salida de los datos procesados.

- Vista Modelo: Es coloca en un punto intermedio entre el modelo y la vista y contiene la lógica de presentación de la interfaz

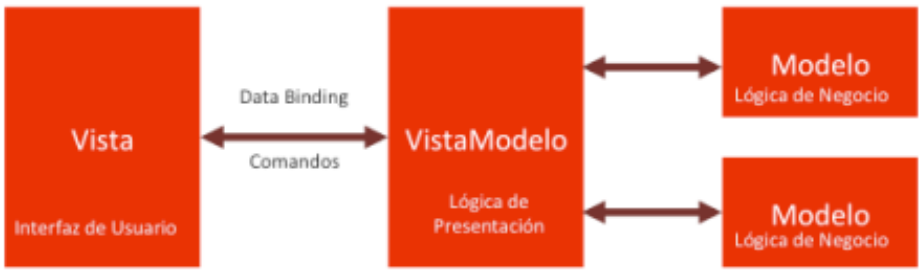


Figura 39 Esquema MVVM

La estructura de ficheros en la que se estructuran las aplicaciones de Ionic y en concreto la de este proyecto es la siguiente:

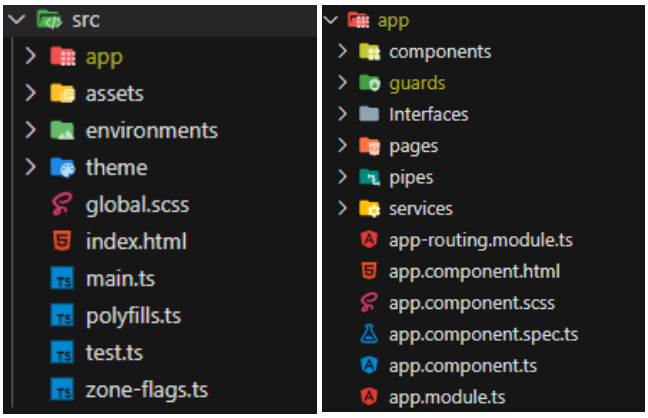


Figura 40 Estructura de ficheros de la aplicación móvil

Según se ha comentado anteriormente la parte correspondiente al apartado de modelo en el patrón de diseño MVVM la encontramos en los *guards*, *interfaces* y *servicios* de la aplicación ya que se encargan de forma aislada de la vista de llevar el peso de la lógica de negocio de la aplicación, así como las conexiones con la API web.

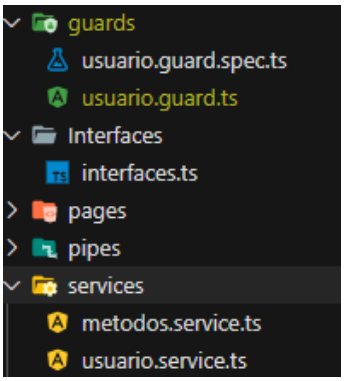


Figura 41 Componentes Modelo de la aplicación móvil

Por otro lado, los componentes correspondientes con la vista y la vista modelo los podemos encontrar dentro de las carpetas *pages*, *components* y pipes ya que cada carpeta contiene un bloque vista-vista modelo en el que la parte de la vista la componen los archivos ".html" y la de la vista modelo los archivos ".ts".

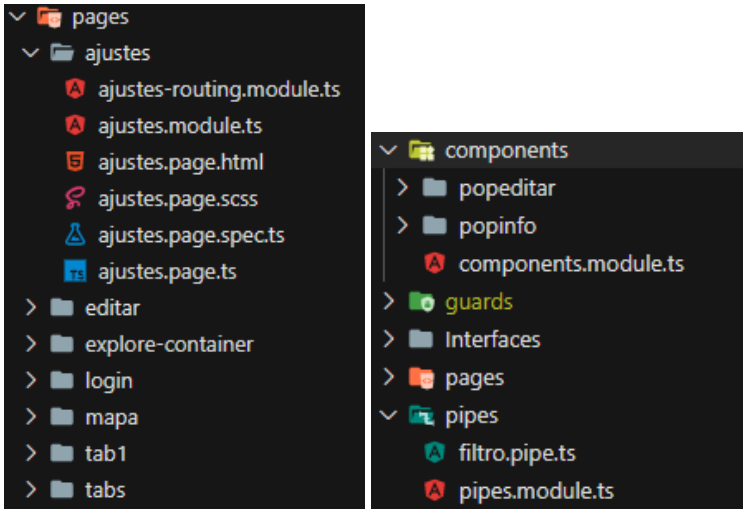


Figura 42 Componentes Vista y Vista Modelo de la aplicación móvil

4.3.2 Diseño y explicación de uso de la interfaz

Para la creación de la interfaz de la aplicación se parte de un prototipado inicial el cual pretende mostrar en forma de boceto las diferentes páginas e interacciones posibles en las que se compone la interfaz del proyecto. Para el prototipado de la interfaz se ha decidido la realización de un prototipo conocido como Prototipo de papel [28] ya que proporciona ventajas ante otros tipos de prototipos como son su rapidez de construcción, su facilidad de manipulación y su bajo coste de recursos (se puede realizar con materiales muy básico).

Figura 43 Prototipo de papel del Login, Registro y sus alertas

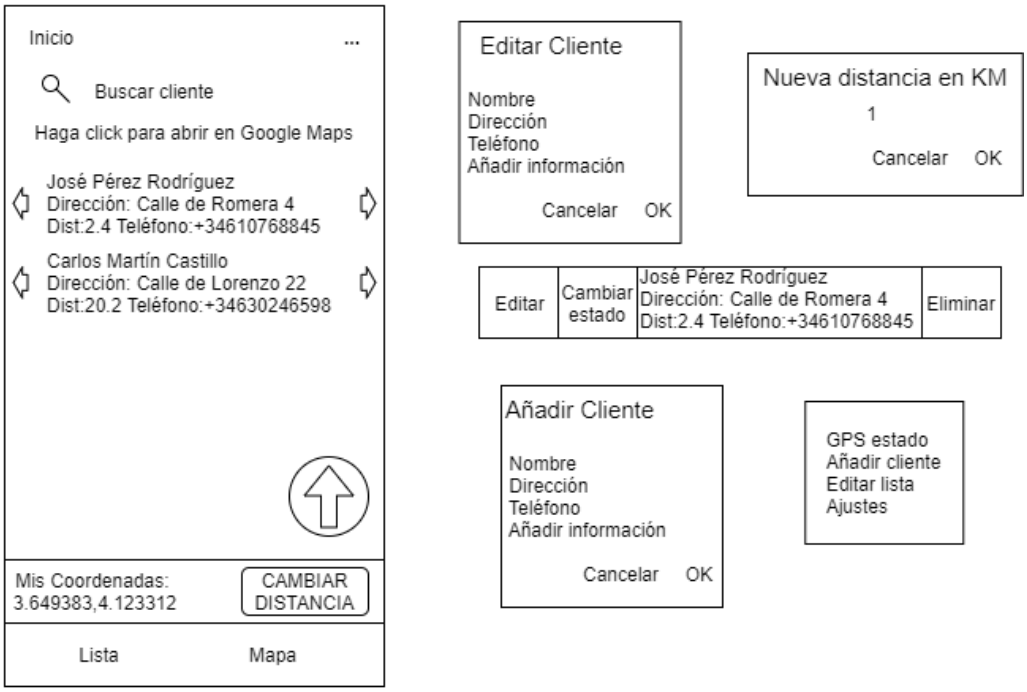


Figura 44 Prototipo de papel del Inicio, sus componentes y formularios

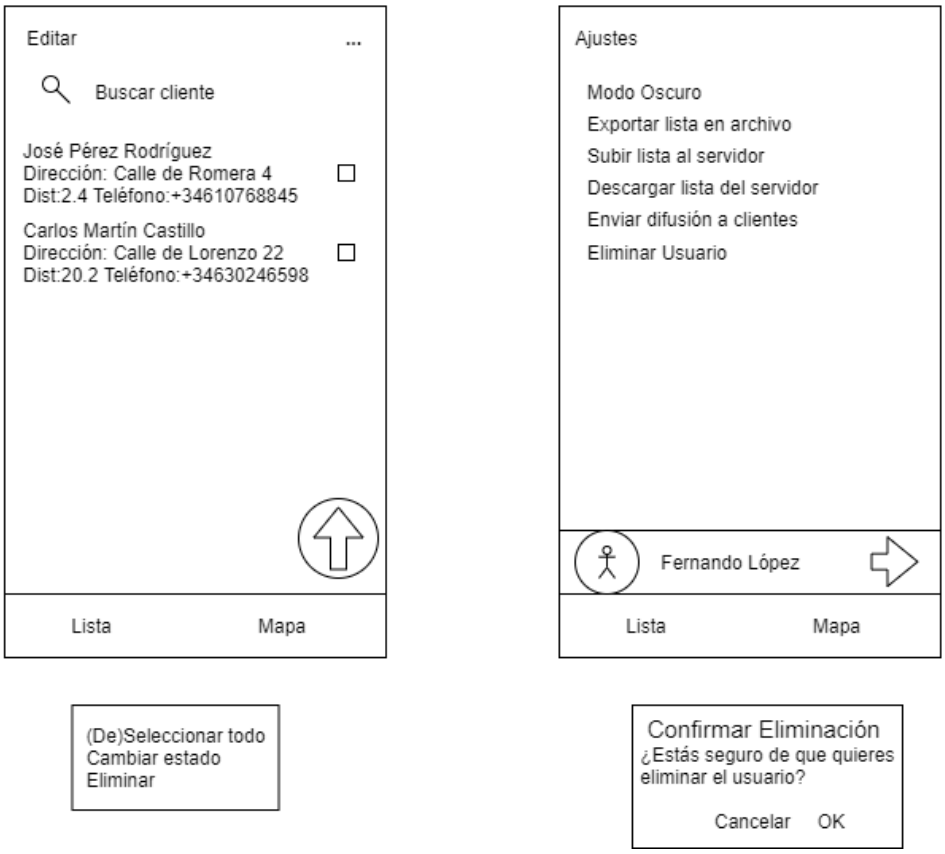


Figura 45 Prototipo a papel de Editar lista, Ajustes, sus componentes y avisos

Figura 46 Prototipo de papel del Mapa

En base a este prototipado inicial se forma la que es la interfaz de la parte correspondiente con la aplicación móvil del sistema. La interfaz también se encuentra por completo en un modo oscuro, el cual puede ser seleccionado desde la página de ajustes de la aplicación.

En primer lugar, se muestra el logotipo que incluye el icono, así como el nombre de la aplicación, la cual se encuentra en su versión 1.0

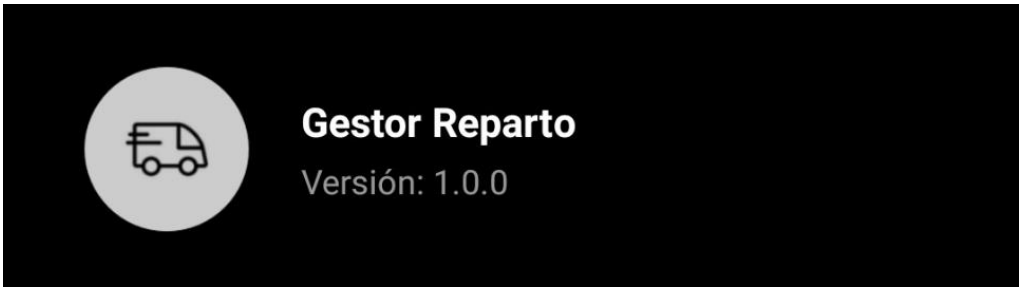


Figura 47 Icono y nombre de la aplicación móvil

A continuación, se encuentra la versión final de la interfaz de la aplicación juntos con las funcionalidades que se encuentran en cada sección de ésta. La versión de la interfaz en modo oscuro será la misma, pero cambiando ciertos colores claros de ésta a tonos negros y grisáceos que hacen que en situaciones de baja luminosidad la fatiga visual se vea reducida, incrementando la comodidad de uso de la app.

Ésta será la vista de la zona de inicio de sesión (Login) de la aplicación. La vista de la zona de Registro de la aplicación contiene también un formulario e incluye un carrusel de selección de posibles avatares para el nuevo usuario. En la parte inferior de estas vistas se encuentran unos botones para la navegación entre ellas.

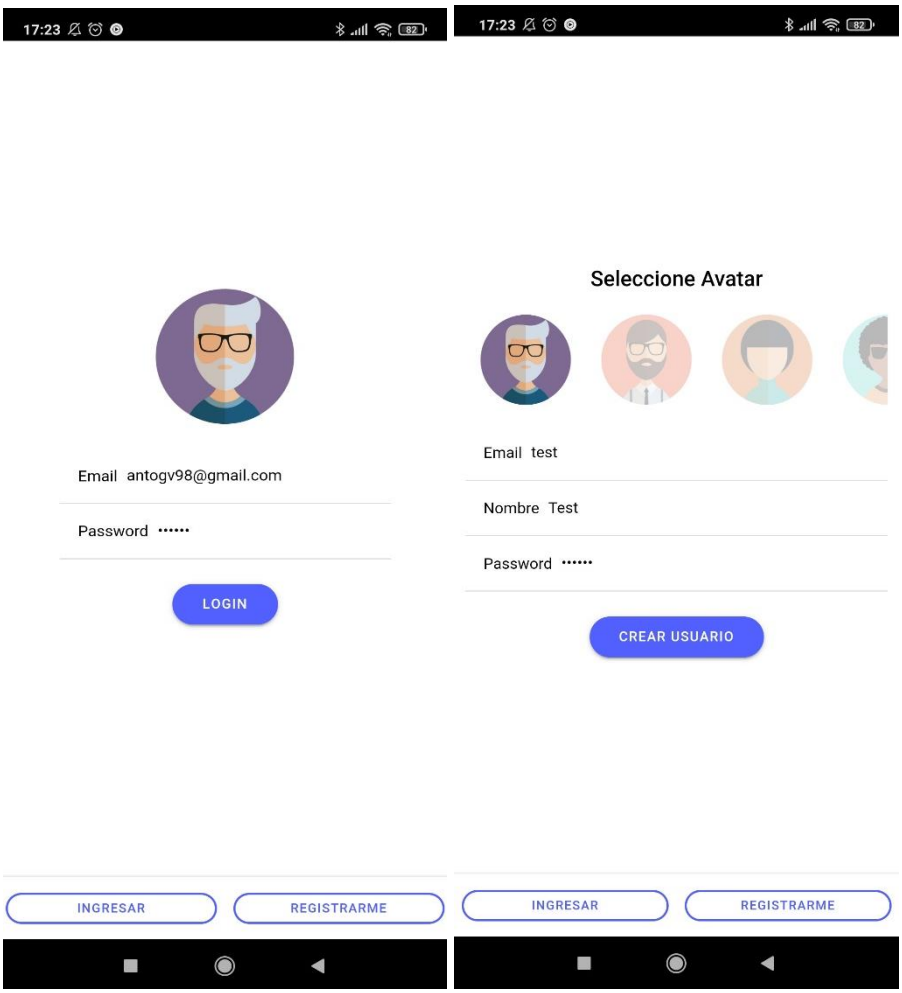


Figura 48 Interfaz Login y Registro

También encontramos dos posibles alertas que se mostrarán en el caso de que los campos no sean correctos en el caso del Login o que se trate de introducir un correo electrónico, ya existente en la base de datos de usuario, en el formulario de registro

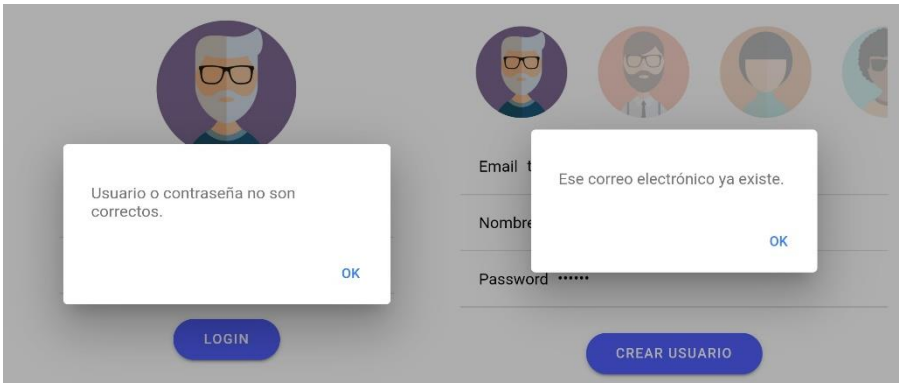


Figura 49 Interfaz las alertas de Login y Registro

Tras iniciar sesión se puede ver la vista de la página de inicio de la aplicación donde se muestra la lista de clientes, así como una barra de búsqueda que filtra la lista de estos.

El botón inferior con una flecha ascendente se encarga de subir al inicio de la lista en el caso de que la cantidad de cliente sea superior a los que se pueden mostrar al mismo tiempo en pantalla

También es accesible desde esta página, a través de la barra inferior, la página del mapa. Un menú desplegable es mostrado haciendo click en los puntos que se encuentran en la parte superior derecha.

Desde un botón de acceso rápido en la parte inferior, junto a la información de las coordenadas del cliente, se despliega una ventana flotante en la cual se puede introducir un nuevo valor para la distancia mínima de alerta entre usuario y clientes.

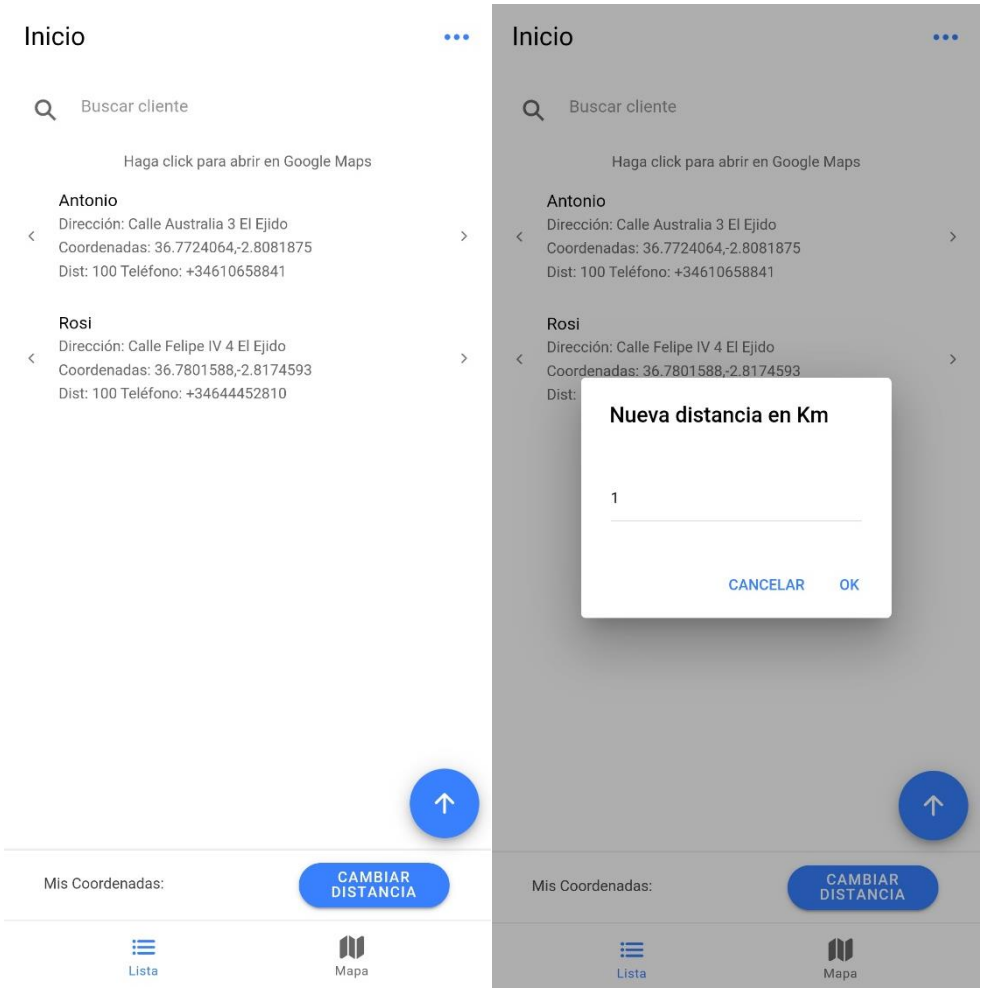


Figura 50 Interfaz del Inicio y el cambio de distancia alerta

Desplazando hacia los laterales cualquiera de los clientes de la lista se encuentran una serie de opciones que permiten individualmente editar los campos del cliente, cambiar su estado o eliminarlo de la lista.

En el caso de que el estado del cliente cambie manual o automáticamente por las funcionalidades internas de la aplicación, el color de la opción de cambiar estado se modifica y después de los datos del cliente es mostrado un icono informativo sobre el nuevo estado de éste.

Figura 51 Interfaz de los botones de editar, cambiar estado y eliminar cliente

El botón de editar cliente muestra una ventana con un formulario prellenado con los datos del cliente para que el usuario pueda editar los campos que considere necesarios.

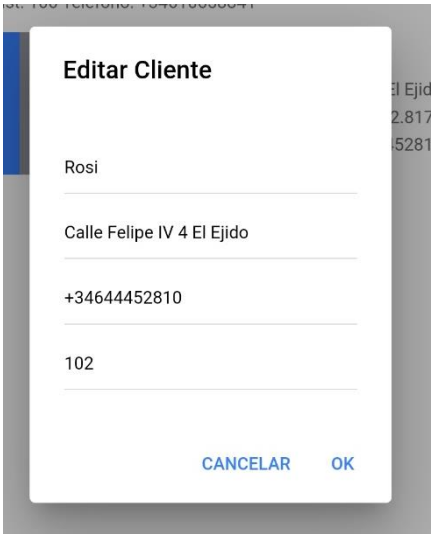


Figura 52 Interfaz de la edición del cliente

En la zona del menú superior que se ha comentado se despliega una ventana de opciones entra las que se encuentra las opciones de activar-desactivar el GPS, añadir cliente manualmente, acceder a la página de editar lista y acceder a la página de ajustes.

El icono asociado a la opción GPS estado varía de color en función de si éste está activado o no.

La opción de añadir cliente manualmente abre una ventana similar a la ventana de modificar cliente, pero con los campos de éste vacíos para ser rellenados por el usuario.

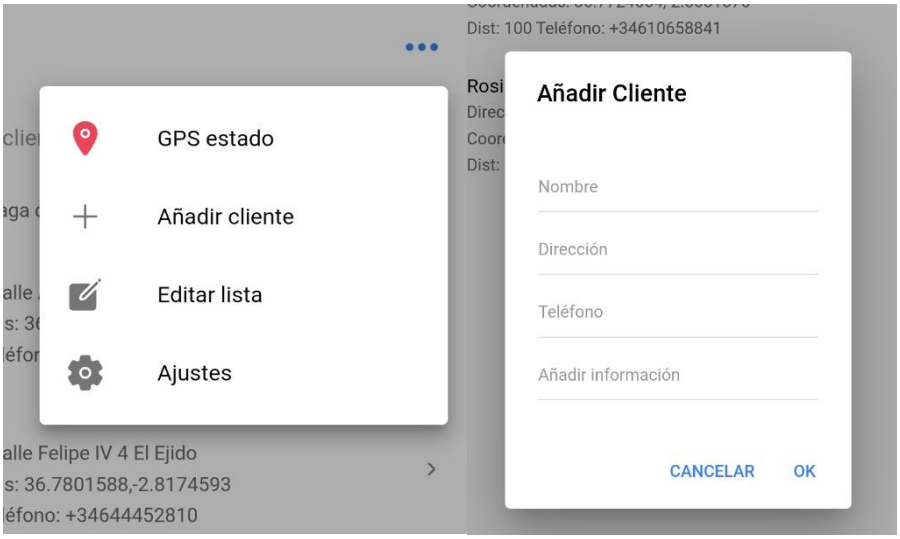


Figura 53 Interfaz del menú de Inicio y añadir cliente

Si desde el menú de Inicio se hace click en editar lista se muestra la vista correspondiente con la página de edición de la lista de cliente, esta vista muestra una lista de clientes seleccionable, una barra de búsqueda, la cual filtra esta lista a través del nombre y un menú de opciones desplegable en la misma posición que en la vista de la página Inicio.

En la zona del menú superior se despliega una ventana de opciones entra las que se encuentra las opciones de seleccionar-deseleccionar todos los clientes, cambiar el estado de los seleccionados, al siguiente estado común para todo, y la opción de eliminar los clientes seleccionados.

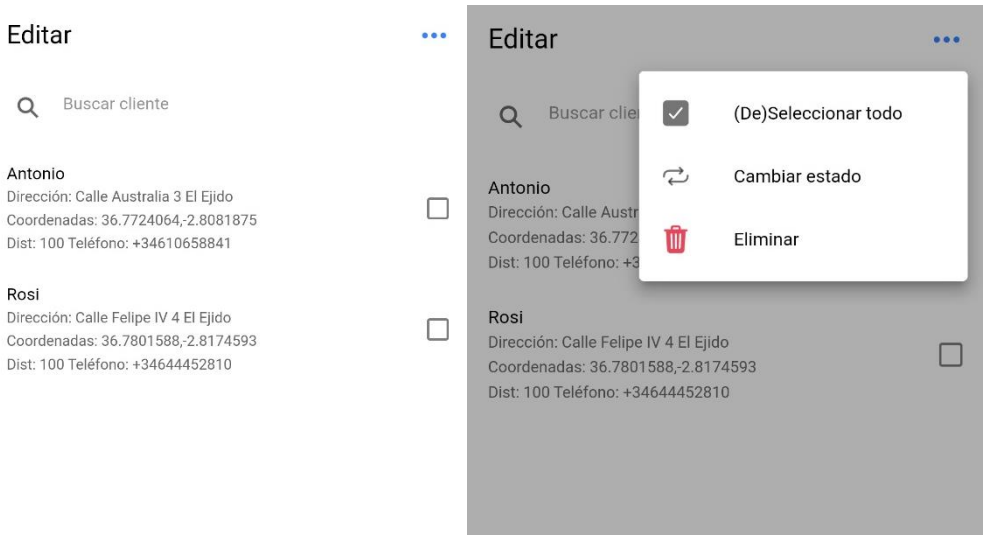


Figura 54 Interfaz de la página de editar y menú de editar

La parte interior de esta página es similar a la página de Inicio, pero sin la sección de coordenadas del usuario ni el acceso rápido al cambio de distancia de alerta. Tan solo se encuentra la barra de navegación desde la cual se puede volver a la página de inicio donde se muestra la lista de clientes o al mapa. También se encuentra el botón de desplazamiento rápido al inicio de la lista.

Figura 55 Interfaz de la zona inferior de la página Editar

Si en lugar de acceder a la sección de editar desde el menú del Inicio, se accede a los ajustes se muestra una lista de opciones posibles como son el cambio de tema a modo oscuro, exportar la lista de clientes en un archivo localmente, subir la lista al servidor, descargar la lista que se encuentre subida en el servidor, enviar un mensaje informativo a todos los clientes de la lista y eliminar el usuario desde el que se ha accedido.

En el caso de que se escoja la opción de eliminar usuario, una ventana de confirmación es mostrada para evitar eliminaciones de usuarios por error.

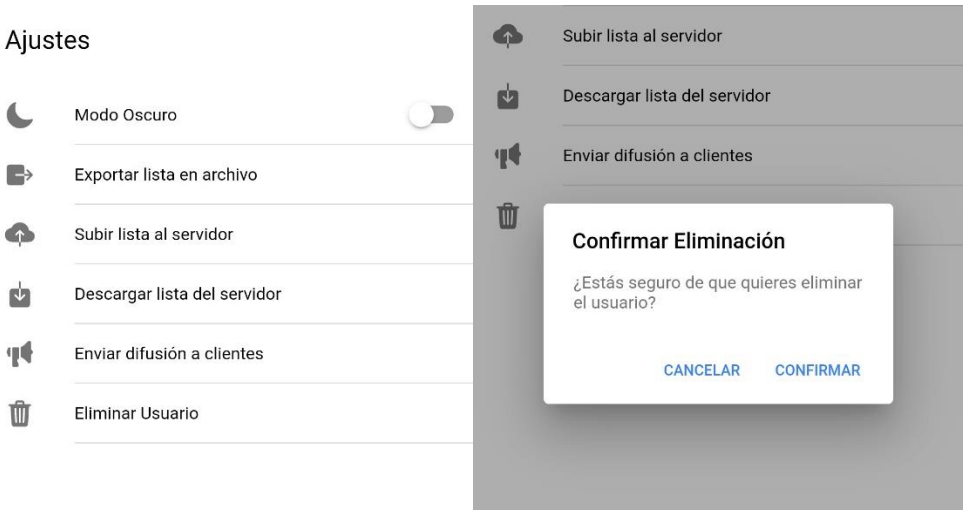


Figura 56 Interfaz de la página Ajustes y de la confirmación de eliminar usuario

Esta es la interfaz de la parte correspondiente con la aplicación móvil del sistema, junto con una explicación general del uso de la misma.

Como se ha comentado anteriormente existe la posibilidad de cambiar el tema de la aplicación a modo oscuro, ésta intercambia los valores de blancos y negros haciendo que los fondos tomen un tema oscuro y los textos claro para resaltar en dichos fondos. El cambio al tema oscuro proporciona una posibilidad de personalización al usuario para que éste puede escoger la opción que más cómoda le resulte, siendo la opción oscura más favorable en momentos de poca luminosidad que el tema por defecto de la aplicación.

La única zona que se mantiene clara dentro de la interfaz cuando se activa el tema oscuro es el mapa de la aplicación ya que éste es un componente externo e independiente a la aplicación.

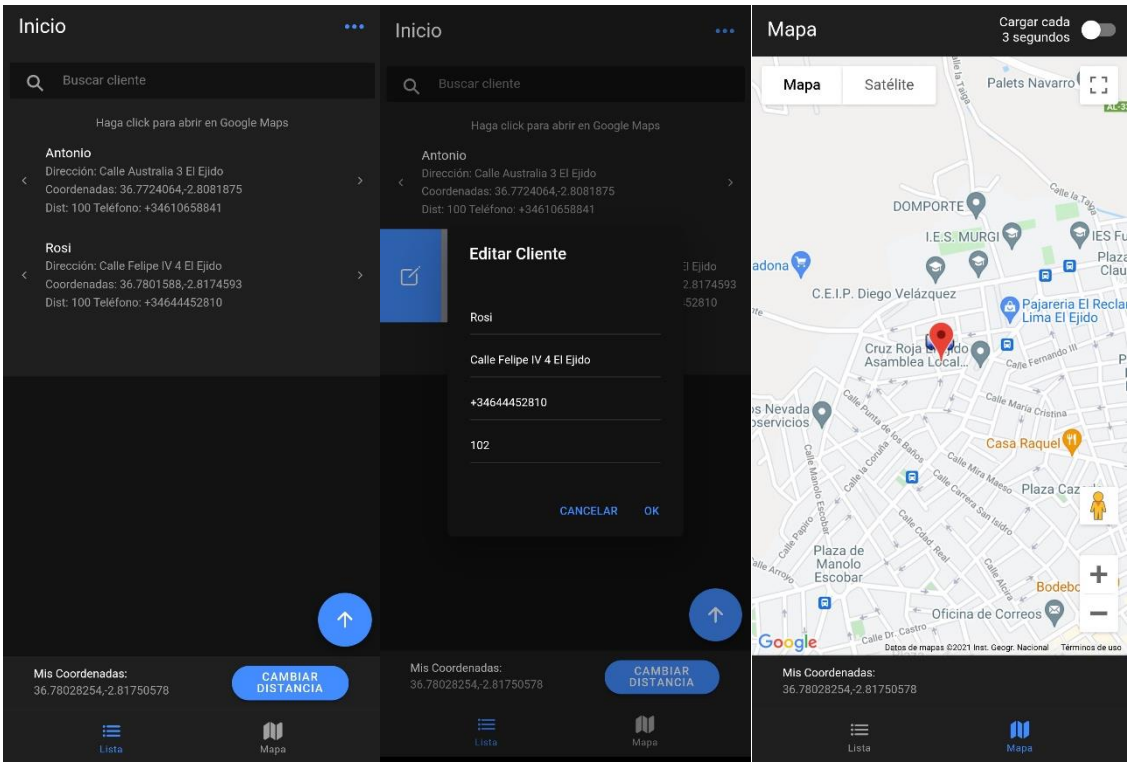


Figura 57 Interfaz de la aplicación en modo oscuro

4.3.3 Aspectos destacados de la implementación

Una vez vista la interfaz de la aplicación a continuación, se muestran los detalles relacionados con la implementación que otorgan funcionalidad a la misma. La implementación se ve apoyada y reflejada en el siguiente diagrama de clases UML que traza claramente la estructura del sistema al modelar sus clases, atributos y métodos, así como la relación entre las clases.

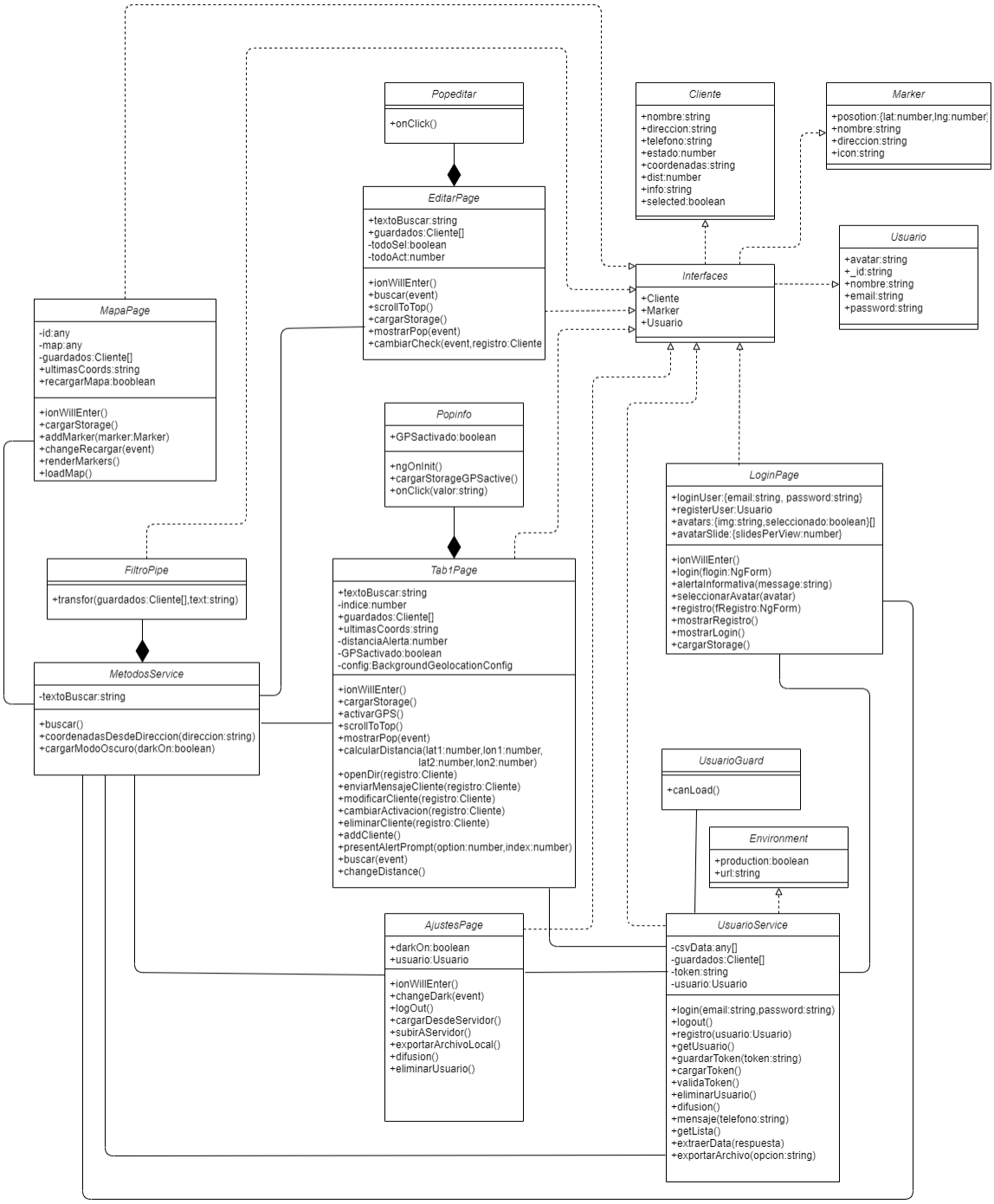


Figura 58 Diagrama de clases de la aplicación móvil

Tras observar las distintas clases que componen la implementación de la parte del sistema correspondiente con la aplicación multiplataforma, se verá de forma más detallada como funciona internamente cada una de éstas.

4.3.3.1 LoginPage

El método *login()* es uno de los principales métodos de la página *LoginPage*, éste recibe un formulario y comprueba que es un formulario válido, si es así llama al método *login()* de la clase *usuarioService*, enviando los valores de las variables *loginUser.email* y *loginUser.password*, éste le devuelve un valor booleano que en caso de ser true desplaza al usuario hasta la página de inicio y en caso de ser false le muestra un mensaje de error.

```
async login(fLogin:NgForm){

  if(fLogin.invalid){return;}
  const valido = await this.usuarioService.login(this.loginUser.email,this.loginUser.password);

  if(valido){
    //navegar tabs
    this.navController.navigateRoot('/main/tabs/tab1',{animated:true});
  }else{
    //mostrar alerta de usuario y contraseña no correctos
    this.alertaInformativa('Usuario o contraseña no son correctos.');
  }

}
```

Figura 59 Método login() de LoginPage

El otro método principal de la clase es registro, éste funciona del mismo modo que *login()* pero en lugar de llamar a éste método de la clase *usuarioService*, llama al método *registro()* y le enviar el objeto *registerUser*.

```
async registro(fRegistro:NgForm){

  if(fRegistro.invalid){return;}
  const valido = await this.usuarioService.registro(this.registerUser);

  if(valido){
    //navegar tabs
    this.navController.navigateRoot('/main/tabs/tab1',{animated:true});
  }else{
    //mostrar alerta de email ya existente
    this.alertaInformativa('Ese correo electrónico ya existe.');
  }

}
```

Figura 60 Método registro() de LoginPage

4.3.3.2 Tab1Page

El método *activarGPS()* de la página *Inicio (Tab1Page)* es el método en el que se basa la idea original de este proyecto, éste se encarga de obtener los datos de la posición del usuario en todo momento y de decidir cuándo cambiar el estado del cliente y llamar al método encargado de mediante una petición al servidor, enviar un mensaje a través de Whatsapp al cliente.

Mediante una suscripción al valor del *this.backgroundGeolocation.on()* se obtiene de forma actualizada las coordenadas del usuario, éstas son guardadas en las variables temporales de la aplicación y en el Storage. Cuando las coordenadas son se recorre la lista de clientes y se llama al método *calcularDistancia()*, el cual se encarga de obtener la distancia entre las coordenadas del cliente y las del usuario. Cuando las coordenadas de un cliente se encuentran a una distancia menor que la distancia de alerta, se modifica el estado de e éste y llama al método de *enviarMensajeCliente()*. Con cada iteración de la lista se reordena la misma en base a la distancia con el usuario.

```
activarGPS(){

  this.backgroundGeolocation.configure(this.config).then(() => {
    this.backgroundGeolocation.on(BackgroundGeolocationEvents.location).subscribe((location: BackgroundGeolocationResponse) => {
      //Mantiene actualizada la información de this.ultimas coordenadas al nuevo valor en el html
      this.ngZone.run(() => {
        this.ultimasCoords= `${location.latitude},${location.longitude}`;
        this.storage.set('ultimasCoords',this.ultimasCoords);

        this.guardados.forEach(cLient => {
          if (client.estado===2){
            let latlonYo=this.ultimasCoords.split(',');
            let latlonClien=client.coordenadas.split(',');
            client.dist=this.calcularDistancia(latlonYo[0],latlonYo[1],latlonClien[0],latlonClien[1]);

              if(Number(client.dist)<this.distanciaAlerta){
                client.estado=3;
                this.enviarMensajeCliente(client);
                client.estado=4;
                this.storage.set('guardados',this.guardados);

              }
          }
        });
        this.guardados.sort(function (a, b) {
          if (a.dist > b.dist) {
            return 1;
          }
          if (a.dist < b.dist) {
            return -1;
          }
          return 0;
        });
      });
      //Impide que la app de IOS se cierre de forma inesperada
      this.backgroundGeolocation.finish();
    },(error)=>{console.log('error in backgroundGeolocation '+error);});
  });
}
```

Figura 61 Método activarGPS() de Tab1Page

El método de *calcularDistancia()* aplicar la *Fórmula de Haversine* para realizar el cálculo en kilómetros de la distancia entre dos pares de coordenadas geográficas.

R = *radio de la Tierra*

$$\Delta lat = lat2 - lat1$$

$$\Delta long = long2 - long1$$

$$a = \sin^2(\Delta lat/2) + \cos(lat1) \cdot \cos(lat2) \cdot \sin^2(\Delta long/2)$$

$$c = 2 \cdot atan2(\sqrt{a}, \sqrt{(1-a)})$$

$$d = R \cdot c$$

Figura 62 Fórmula de Haversine

```
calcularDistancia(lat1,lon1,lat2,lon2):number{
    var rad = function(x) {return x*Math.PI/180;}
    var R = 6378.137; //Radio de la tierra en km
    var dLat = rad( lat2 - lat1 );
    var dLong = rad( lon2 - lon1 );
    var a = Math.sin(dLat/2) * Math.sin(dLat/2) + Math.cos(rad(lat1)) * Math.cos(rad(lat2)) * Math.sin(dLong/2) * Math.sin(dLong/2);
    var c = 2 * Math.atan2(Math.sqrt(a), Math.sqrt(1-a));
    var d = R * c;
    return Number(d.toFixed(3)); //Retorna tres decimales
}
```

Figura 63 Método calcularDistancia() de Tab1Page

El método de *cambioActivacion()* cambia el estado del cliente recibido por parámetro al siguiente y le asigna una distancia determinada para mantener siempre ordenada la lista de clientes siendo los cancelados y completados, los que siempre ocupen las últimas posiciones de la lista.

```
cambioActivacion(registro:Cliente){
  var indice = this.guardados.indexOf(registro);
   if( this.guardados[indice].estado===0){
    this.guardados[indice].estado=1;
    this.guardados[indice].dist=100;
   }else if( this.guardados[indice].estado===1){
    this.guardados[indice].estado=2;
    this.guardados[indice].dist=100;
   }else if( this.guardados[indice].estado===2||this.guardados[indice].estado===3||this.guardados[indice].estado===4){
    this.guardados[indice].estado=5;
    this.guardados[indice].dist=101;
   }else{
    this.guardados[indice].estado=0;
    this.guardados[indice].dist=102;
   }
   this.storage.set('guardados', this.guardados);
 }
```

Figura 64 Método cambioActivacion() de Tab1Page

El método *presentAlertPrompt()* es el encargado de mostrar la ventana de editar cliente y añadir cliente, cambiando los valores de los campos de texto en función de la opción. Una vez completo el formulario los clientes son establecidos con el resto de las propiedades por defecto y añadidos en la lista como nuevos clientes o sustituyendo su versión anterior.

```
async presentAlertPrompt(option:number, index?:number) {
  if(option){
    var titulo='Editar Cliente';
    var valNom=this.guardados[index].nombre;
    var valDir=this.guardados[index].direccion;
    var valTel=this.guardados[index].telefono;
    var valEst=this.guardados[index].estado;
    var valInf=this.guardados[index].info;
  }

  const alert = await this.alertController.create({

    header: titulo ||'Añadir Cliente',
    inputs: [
      {
        name: 'nombre',
        type: 'text',
        value:valNom||'',
        placeholder: 'Nombre'
      },
      {
        name: 'direccion',
        type: 'text',
        value:valDir||'',
        placeholder: 'Dirección'
      },
      {
        name: 'telefono',
        type: 'text',
        value:valTel||'',
        placeholder: 'Teléfono'
      },
      {
        name: 'info',
        type: 'text',
        value:valInf||'',
        placeholder: 'Añadir información'
      }
    ],
    buttons: [
      {
        text: 'Cancelar',
        role: 'cancel',
        cssClass: 'secondary',
        handler: () => {

        }
      }, {
        text: 'Ok',
        handler: async (data:any) => {
          var coordenadasCliente:string= await this.metodosService.

          let cliente:Cliente;
          cliente ={nombre:data.nombre,
            direccion:data.direccion,
            telefono:data.telefono,
            estado:1,
            coordenadas:coordenadasCliente,
            dist:100,
            info:data.info,
            selected:false
          };

          if(option){
            cliente.estado=valEst;
            this.guardados[index]=cliente;
          }else{
            this.guardados.push(cliente);
          }

          this.storage.set('guardados',this.guardados);

          if(this.textoBuscar!=''){
            location.reload();
          }

        }
      }
    ]
  });
```

Figura 65 Método presentAlertPrompt() de Tab1Page

4.3.3.3 EditarPage

La lógica principal de la página *EditarPage* se encuentra en el menú de opciones que ésta tiene.

El método *mostrarPop()* es el encargado de mostrar este menú de opciones, en función del valor que éste devuelva realiza una función u otra. Si el valor que devuelve es el correspondiente con la primera opción se recorre la lista de clientes y se establecen todos con el valor seleccionado a no ser que previamente estuviera todos seleccionados (en ese caso los establece con el valor deseleccionado). En el caso de que el valor que devuelve es el de la segunda opción se comprueba si el valor de estado de todos los clientes es diferente, inactivos o activos, se recorre la lista y se establece el estado de los clientes seleccionados en el siguiente correspondiente.

```
async mostrarPop(ev:any){

  const popover= await this.popoverController.create({
    component:PopeditarComponent,
    event: ev,
    translucent: true,
  });
  await popover.present();
  const {data} =await popover.onWillDismiss();

  if(data){

   if(data.item==="seltodo"){
     //Selecciona todos los clientes o los deselecciona
     if(this.todoSel){
      this.todoSel=false;
      this.guardados.forEach(registro=>{
        registro.selected=false;
      });
     }else{
      this.todoSel=true;
      this.guardados.forEach(registro=>{
        registro.selected=true;
      });
     }
```

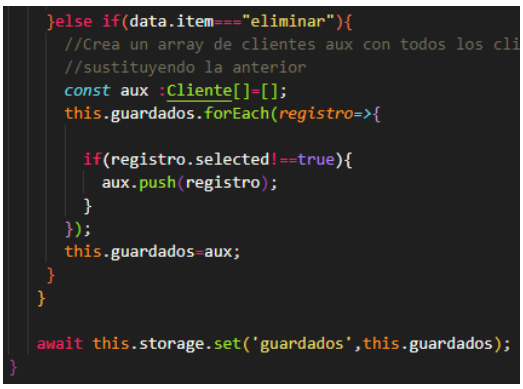
```
}else if(data.item==="acticancel"){
  //Recorre la lista de clientes guardados
  if(this.todoAct===0){
    this.todoAct=1;
    this.guardados.forEach(registro=>{
      if(registro.selected===true){
        registro.estado=1;
        registro.dist=100;
      }
    });
  }else if(this.todoAct===1){
    this.todoAct=2;
    this.guardados.forEach(registro=>{
      if(registro.selected===true){
        registro.estado=2;
      }
    });
  }else if(this.todoAct===2){
    this.todoAct=5;
    this.guardados.forEach(registro=>{
      if(registro.selected===true){
        registro.estado=5;
        registro.dist=101;
      }
    });
  }else{
    this.todoAct=0;
    this.guardados.forEach(registro=>{

      if(registro.selected===true){
        registro.estado=0;
        registro.dist=102;
      }

    });
  }
```

Figura 66 Método mostrarPop() de EditarPage parte 1

La última opción elimina los clientes seleccionados creando una lista de clientes auxiliar en la que no se encuentran los clientes seleccionados y se sustituye el valor original de ésta.



```
  }else if(data.item==="eliminar"){
    //Crea un array de clientes aux con todos los cli
    //sustituyendo la anterior
    const aux :Cliente[]=[];
    this.guardados.forEach(registro=>{

      if(registro.selected!==true){
        aux.push(registro);
      }
    });
    this.guardados=aux;
  }
 }

 await this.storage.set('guardados',this.guardados);
}
```

Figura 67 Método mostrarPop() de EditarPage parte 2

4.3.3.4 AjustesPage

En el caso de *AjustesPage*, todos los métodos realizan llamadas a los métodos de la clase *UsuarioService*, la cual se verá mas adelante en el apartado *4.3.3.7 UsuarioService*.

El método *eliminarUsuario()* de esta clase previo a la llamada al método correspondiente de la clase *UsuarioService* muestra una ventana de confirmación al usuario para prevenir una eliminación errónea del usuario.

```
async eliminarUsuario(){
  const alert = await this.alertController.create({
    cssClass: 'my-custom-class',
    header: 'Confirmar Eliminación',
    message: '¿Estás seguro de que quieres eliminar el usuario?',
    buttons: [
      {
        text: 'Cancelar',
        role: 'cancel',
        cssClass: 'secondary',
        handler: (blah) => {

        }
      }, {
        text: 'Confirmar',
        handler: () => {
          this.usuarioService.eliminarUsuario();
        }
      }
    ]
  });

  await alert.present();
}
```

Figura 68 Método eliminarUsuario() de AjustesPage

4.3.3.5 MapaPage

Los métodos más importantes de la clase *MapaPage* son *addMarker()*, *renderMarkers()* y *loadMap()*, los cuales se encargan de crear los marcadores, añadirlos al mapa y cargar el mapa de Google Maps.

El método *addMarker()* crea un string de contenido para que se muestra al hacer click en el marcador, esta información es añadida en el mapa y un nuevo objeto *mark* es creado con los valores recibidos dentro del objeto recibido por parámetro.

```
addMarker(marker: Marker) {

  const contentString =
  '<div id="bodyContent">' +
  '<p style="color:#000000">Nombre: ' + marker.nombre+'</p>'+
  '<p style="color:#000000">Direccion: ' + marker.direccion+'</p>'+
  '</div>';

  const infowindow = new google.maps.InfoWindow({
    content: contentString
  });
  const mark =new google.maps.Marker({
    position: marker.position,
    map: this.map,
    title: marker.nombre,
    icon:marker.icon
  });

  mark.addListener("click", () => {
    infowindow.open(this.map, mark);
  });

}
```

Figura 69 Método addMarker() de MapaPage

El método *renderMarkers()* es el encargado de llamar al método *addMarker()* con todos los clientes, para ello recorre la lista de clientes y creo un objeto *marker* por cada cliente que envía al método *addMarker()* antes de pasar al siguiente cliente. Un último objeto *marker* es enviado, pero con las coordenadas del usuario y un icono personalizado para identificar el punto en el que se encuentra éste a la hora de abrir el mapa.

```
renderMarkers(){
  this.guardados.forEach(registro=>{

    const latlon:String[]= registro.coordenadas.split(',');
    const marker = {
      //+ Permite parsear de String a number
      position:{
        lat: +latlon[0],
        lng: +latlon[1]
      },
      nombre: registro.nombre,
      direccion:registro.direccion

    }

    this.addMarker(marker);
  });

  const coordsCenterMe: String []= this.ultimasCoords.split(',');
  const coordsCenterMeNumbered: number []= [+coordsCenterMe[0],+coordsCenterMe[1]];
  const centerMarker = {
    //+ Permite parsear de String a number
    position:{
      lat: coordsCenterMeNumbered[0],
      lng: coordsCenterMeNumbered[1]
    },
    nombre: 'Yo',
    direccion:''+coordsCenterMe,
    icon: 'https://www.google.com/intl/en_us/mapfiles/ms/micons/truck.png'
  }
  this.addMarker(centerMarker);
}
```

Figura 70 Método renderMarkers() de MapaPage

El método *loadMap()* crea un nuevo elemento de tipo HTMLElement con el identificador *map* el cual contiene el mapa de Google con la configuración de lugar de centrado (las últimas coordenadas obtenidas del usuario) y zoom establecido. Tras esto al llamar al elemento *map* en el HTML el mapa puede ser visualizado.

```
async loadMap() {
  this.ngZone.run(async() => {
  this.ultimasCoords =  await this.storage.get('ultimasCoords')|| '40.4167,-3.70325';
  });
      //Crea un nuevo mapa de tipo HTMLElement
    const mapEle: HTMLElement = document.getElementById('map');

    var coordsCenterMe: String []= this.ultimasCoords.toString().split(',');
    const coordsCenterMeNumbered: number []= [+coordsCenterMe[0],+coordsCenterMe[1]];
    //Objeto con la lat y lng de ultimasCoords dividido por "," y convertidas en number
    const myLatLng = {lat: coordsCenterMeNumbered[0], lng: coordsCenterMeNumbered[1]};
    // Crea un nuevo mapa centrado en las ultimasCoords
    this.map = new google.maps.Map(mapEle, {
      center: myLatLng,
      zoom: 16
    });

    //Tras crear el mapa añade los marcadores y lo muestra
    google.maps.event.addListenerOnce(this.map, 'idle', () => {
      this.renderMarkers();
        mapEle.classList.add('show-map');
    });

}
```

Figura 71 Método loadMap() de MapaPage

4.3.3.6 MetodosService

La clase *MetodosService* contiene los métodos comunes que son llamados en métodos de las clases anteriores como son el de *buscar()*, *coordenadasDesdeDireccion()* y *cargarModoOscuro()*.

El método *buscar()* se encarga de crear una variable que utiliza el pipe para filtrar la lista con el texto escrito en la barra de búsqueda. El encargado de establecer el tema (claro u oscuro) de la aplicación es el método *cargarModoOscuro().*

Es el método *coordenadasDesdeDireccion()* el que contiene la lógica más interesante de esta clase ya que mediante una dirección obtenida por parámetro en formato string y haciendo uso de la librería *nativeGeocoder*, es capaz de obtener el par de coordenadas geográficas asociados a la dirección proporcionada en un objeto JSON. El objeto JSON es leído y convertido en un string ya que éste nos facilita su uso dentro de la lógica de la aplicación.

```
//Cambiar el valor de textoBuscar que es obtenido por el pipe en funcion del texto del evento proporcionado
buscar(event):string{
  this.textoBuscar=event.detail.value;
  return this.textoBuscar;
}

//Obtiene las coordenadas de la direccion proporcionada, esta es obtenida en Json por lo que es formateada
//como string lat,lng
async coordenadasDesdeDireccion(direccion:string){
  var coordendasDir='';
  let options: NativeGeocoderOptions = {
    useLocale: true,
    maxResults: 1
  };
  await this.nativeGeocoder.forwardGeocode(direccion, options).then((result) => {
    coordendasDir=JSON.stringify(result[0].latitude)+','+JSON.stringify(result[0].longitude);
  }).catch(err=>{coordendasDir='Error coordenadas'});
  let re= /\"/gi;
  return coordendasDir.replace(re,'');
}

//Cambia el tema de la app en funcion del valor darkOn proporcionado
cargarModoOscuro(darkOn:boolean){
  if(darkOn){
    document.body.setAttribute('color-theme','dark');
  }else{
    document.body.setAttribute('color-theme','');
  }
}
```

Figura 72 Métodos de MetodosService

4.3.3.7 UsuarioService

La clase *UsuarioService* es la encargada de realizar todas las conexiones http con el servidor, es la clase más grande de todo la parte de la aplicación móvil del sistema.

En este caso se verán por completo todos los métodos de la case, estos son: *login(), logout(), registro(), getUsuario(), guardarToken(), cargarToken(), validaToken(), eliminarUsuario(), difusion(), mensaje(), extraerData y exportarArchivo().*

Los métodos *login()* y *registro()* reciben por parámetro los elementos del formulario rellenado en la página *Login*, realizan una petición http al servidor, éste devuelve el token del usuario que ha accedido a la aplicación y lo envía al método *guardarToken()*. Por último, resuelven un booleano true en caso de conseguir realizar esta acción y un false en caso de no conseguirlo.

El método *logout()* elimina el valor de las variables locales correspondientes con el token y el usuario, vacía el storage y desplaza al usuario a la página *Login.*

```
login(email: string, password: string) {
  const data = { email, password };
  return new Promise(resolve => {
    this.http.post(`${URL}/user/login`, data).subscribe(async resp => {
      if (resp['ok']) {
        await this.guardarToken(resp['token']);
        resolve(true);
      } else {
        this.token = null;
        this.storage.clear();
        resolve(false);
      }
    });

  });

}

//Restablece el token, usuario cargado y el storage y navega hasta la página
logout() {
  this.token = null;
  this.usuario = null;
  this.storage.clear();
  this.navController.navigateRoot('/login', { animated: true });
}


//Hace una petición http al servidor con el usuario, este comprueba si el usu
//devolviendo un booleano
registro(usuario: Usuario) {

  return new Promise(resolve => {
    this.http.post(`${URL}/user/create`, usuario).subscribe(async resp => {
      if (resp['ok']) {
        await this.guardarToken(resp['token']);
        resolve(true);
      } else {
        this.token = null;
        this.storage.clear();
        resolve(false);
      }
    });
  });

}
```

Figura 73 Métodos login(), logout() y registro() de UsuarioService

El método *getUsuario()* tan solo llama al método *validaToken()* si el usuario no se encuentra cargado en la variable local *usuario.* Una vez hecha la comprobación y la llamada en caso de ser necesario devuelve la variable *usuario*.

El método *guardarToken()* recibe por parámetro un token en formato string, lo guarda en su correspondiente variable local y en el storage y tras esto llama al método *validaToken()*.

El método *cargarToken()* se encarga de obtener y guardar el variable local *token* su valor del storage o en caso de no ser encontrado, un null.

```
getUsuario() {
  if (!this.usuario._id) {
    this.validaToken();
  }
  return { ...this.usuario };
}

//Guarda en this.token el token proporcionado, lo guarda
//obtiene el usuario
async guardarToken(token: string) {
  this.token = token;
  await this.storage.set('token', token);
  await this.validaToken();
}

//Carga el token del storage en this.token
async cargarToken() {
  this.token = await this.storage.get('token') || null;
}
```

Figura 74 Métodos getUsuario(), guardarToken() y cargarToken() de UsuarioService

El método *validaToken()* en primer lugar llama al método *cargarToken()*, tras esto si la variable local *token* es null desplaza al usuario a la página *Login* y devuelve un valor booleano falso. En el caso de que el valor del token no sea null se continua con la ejecución del método, éste realiza

una petición http al servidor enviando la variable *token* para en caso de completarse correctamente obtener y guardar localmente el objeto usuario, devolviendo un true, o en caso contrario desplazar al usuario a la página *Login* y devolver un false.

El método *eliminarUsuario()* del mismo modo que el método anterior envía una petición http con el token al servidor, el cual si es completado muestra un mensaje por consola y llama al método *logout()*. En caso de no ser completada la petición se muestra un mensaje de error por consola.

```
async validaToken(): Promise<boolean> {

  await this.cargarToken();

  if (!this.token) {
    this.navController.navigateRoot('/login');
    return Promise.resolve(false);
  }

  return new Promise<boolean>(resolve => {

    const headers = new HttpHeaders({
      'x-token': this.token
    });

    this.http.get(`${URL}/user/`, { headers }).subscribe(resp => {
      if (resp['ok']) {
        this.usuario = resp['usuario'];
        resolve(true);
      } else {
        this.navController.navigateRoot('/login');
        resolve(false);
      }
    });
  });

}

//Hace una petición http al servidor con el token del usuario para eliminar
eliminarUsuario(){
  const headers = new HttpHeaders({
    'x-token': this.token
  });
  this.http.post(`${URL}/user/eliminar`, {},{ headers }).subscribe(() => {
    console.log('Usuario eliminado');
    this.logout();
   }, () => {
    console.log('Usuario NO eliminado');
  });
}
```

Figura 75 Métodos validaToken() y eliminarUsuario() de UsuarioService

Los métodos *difusion() y mensaje()* funcionan realizan una petición http al servidor con el token (y con el teléfono del cliente el método *mensaje()*), si este es completado muestra un mensaje por consola y si no muestra un mensaje de error por consola.

```
difusion(){

  const headers = new HttpHeaders({
    'x-token': this.token
  });
  this.http.post(`${URL}/user/difusion`, {},{ headers }).subscribe(() => {
    console.log('Difusion enviado');
   }, () => {
    console.log('Difusion no enviado');
  });

}

//Hace una petición http al servidor con el token del usuario y el teléfono proporc
//de whatsapp al cliente con el teléfono proporcionado
mensaje(telefono:string){
  const headers = new HttpHeaders({
    'x-token': this.token
  });
  this.http.post(`${URL}/user/mensaje`,{telefono} ,{ headers }).subscribe(() => {
    console.log('Mensaje enviado');
   }, () => {
    console.log('Mensaje no enviado');
  });

}
```

Figura 76 Métodos difusion() y mensaje() de UsuarioService

El método *getLista()* realiza una petición http al servidor, enviando el token del usuario, y envía los datos obtenidos de respuesta al método *extraerData()*.

El método *extraerData()* recibe una respuesta por parámetro y las guarda en una variable temporal *csvData*, ésta es enviada al método *parse()* de la librería *PapaParse* , el cual transforma la respuesta (que debe de ser un archivo csv) en un string llamado *csvData*. Es inicializada como un array vacío la variable *guardados* y se recorren las líneas de *csvData,* creando un objeto cliente con el contenido de cada línea, para después sen incluido en la variable *guardados*. Por último, *guardados* es guardado en el storage.

```
getLista() {

 const headers = new HttpHeaders({
   'x-token': this.token
 });

  this.http.get(URL + '/user/lista/',{headers,responseType: 'text'}).subscribe(
    data => this.extraerData(data),
    err => console.log('error: ', err)
  );
}

//Extrae los datos de archivos .csv y los guarda en la lista de clientes guardados y esta en el Storage
extraerData(res) {
  let csvData = res || '';
   this.papa.parse(csvData, {
    complete: parsedData => {
      this.csvData = parsedData.data;
      this.guardados = [];
      this.csvData.forEach(async registro => {
        let cliente: Cliente;

        var coordenadasCliente: string = await this.metodosService.coordenadasDesdeDireccion(registro[1]) || 'Direccion no valida';

        var disRegistro:number=100;
        if(registro[3]==='5'){
          disRegistro=101;
        }else if(registro[3]==='0'){
          disRegistro=102;
        }

        cliente = {
          nombre: registro[0],
          direccion: registro[1],
          telefono: registro[2],
          estado: Number(registro[3]),
          coordenadas: coordenadasCliente,
          dist: disRegistro,
          info: registro[4],
          selected: false
        };
        this.guardados.push(cliente);

        this.storage.set('guardados', this.guardados);
      });
    }
  });

}
```

Figura 77 Métodos getLista() y extraerData() de UsuarioService

El método *exportarArchivo()* recibe por parámetro un string con el nombre de *opcion.* En primer lugar, el método carga el storage, tras esto se crea una lista *exportedData* que se llena con los datos de los clientes de la lista *guardados*, que es recorrida. La lista *exportedData* es enviada al método *unparse()* de la librería *PapaParse* y guardada en una variable llamada *csv*. Una variable con el nombre de la lista (compuesto por el id del usuario, seguido de la extensión *.csv*) es creada. Se llama al método *writeFile()* de la librería *File*, con la variable dataDirectory del objeto *file*, el nombre de la lista, la variable *csv* y una opción para sobrescribir en caso de existir el archivo.

En el caso de que la opción recibida por parámetro sea *servidor*, se crea un objeto con los elementos que se enviaran al servidor (token y lista) y otro objeto *fileTransfer* del tipo

FileTransferObject, obtenido de la librería *FileTransfer*. Es llamado el método *upload()* del objeto *fileTransfer,* con los elementos mencionados anteriormente, y éste realiza la petición http al servidor que sube la lista.

Cuando la opción recibida por parámetro no es servidor, la lista escrita por el método *writeFile()* es enviado localmente a través de las aplicaciones instaladas del dispositivo gracias a la librería *SocialSharing*.

```
async exportarArchivo(opcion: string) {
  this.guardados = await this.storage.get('guardados') || [];

  let exportedData = [];
  this.guardados.forEach(registro => {
    exportedData.push([registro.nombre, registro.direccion, registro.telefono,registro.estado,registro.info]);
  });
  let csv = this.papa.unparse({
    data: exportedData
  });

  let nombreFile: string = this.usuario._id + '.csv';

    this.file.writeFile(this.file.dataDirectory, nombreFile, csv, { replace: true }).then(res => {

      if (opcion === 'servidor') {
        //La opcion del if es subir el archivo al servidor
        const options: FileUploadOptions = {

          fileKey: 'lista',

          headers: {
            'x-token': this.token
          },
          mimeType: "text/csv",
        };

        const fileTransfer: FileTransferObject = this.fileTransfer.create();

        fileTransfer.upload(res.nativeURL, URL + '/user/upload/', options)
          .then(data => {
            console.log(data);
          }).catch(err => {
            console.log('error en carga', err);
          });


      } else if (opcion === 'local') {
        //La opcion del else es exportar el archivo localmente
        this.socialSharing.share(null, null, res.nativeURL, null);
      }

    });
}
```

Figura 78 Método exportarArchivo() de UsuarioService

4.3.4 Despliegue

El despliegue de la aplicación móvil sería posible tanto en la tienda de aplicaciones oficial de Android, como en la de IOS.

Debido a que este proyecto forma parte de un TFG y no se pretende, a priori, obtener rentabilidad de ello, se ha decido no alojar la aplicación para dispositivos móviles del sistema en ninguna de estas tiendas. Para poder llevar esto a cabo sería necesaria la creación de una cuenta de desarrollador de estas plataformas, siendo los precios de las cuentas de desarrollador en Android de 20 dólares (pago único) y en el caso de IOS de 100 dólares (pago anual).

Otro de los motivos por los que no se ha subido públicamente la aplicación a ninguna de las tiendas mencionadas anteriormente es debido a que se puede superar el límite de uso gratuito

de APIS de pago en su versión gratuita, de servicios de hosting para la parte del sistema correspondiente con la API del servidor y de hosting de bases de dato.

Para la distribución de la aplicación debido a esta situación para android se ha generado un paquete instalable .apk desde el cual cualquier usuario con este archivo puede instalar la aplicación. Para IOS en cambio es necesario instalarla desde un ordenador MacOS que contenga el proyecto y el entorno de desarrollo Xcode.

En caso de asociar este software a una empresa que pudiera beneficiarse económicamente con su uso, se alojarían las aplicaciones en sus respectivas tiendas.

4.4 Aplicación del servidor

La aplicación del servidor es la parte del sistema que se encuentra alojada en un servicio de hosting y con la que a través de peticiones http interactúa la aplicación móvil. Mediante estas peticiones, por parte de la aplicación móvil, la aplicación del servidor interactuar con la base de datos de usuarios, alojar copias de las listas de clientes de cada usuario y llevar a cabo las funcionalidades relacionadas con la comunicación mediante Whatsapp con los clientes.

4.4.1 Patrón de diseño

El patrón de diseño utilizado para la aplicación del servidor se basa en una estructura dividida en 3 capas [29]. Esta división tiene el objetivo de alejar tratar de alejar la lógica del sistema de las rutas de la API y los accesos a la base de datos, para ellos se realiza la siguiente división:

- Rutas de Express: Contiene todas las rutas a las que se pueden realizar peticiones http en la API REST, recibe las peticiones y los datos o archivos asociados a ellas y los traslada hacia la lógica interna de la aplicación y la lógica de acceso a la base de datos.
- Servicios y clases: Contienen la funcionalidad principal del sistema, conteniendo la lógica interna que es llamada desde las rutas.
- Acceso a base de datos con Mongoose: Se encarga de gestionar las peticiones relacionadas con la creación, modificación y eliminación de registros Usuario de la base de datos.

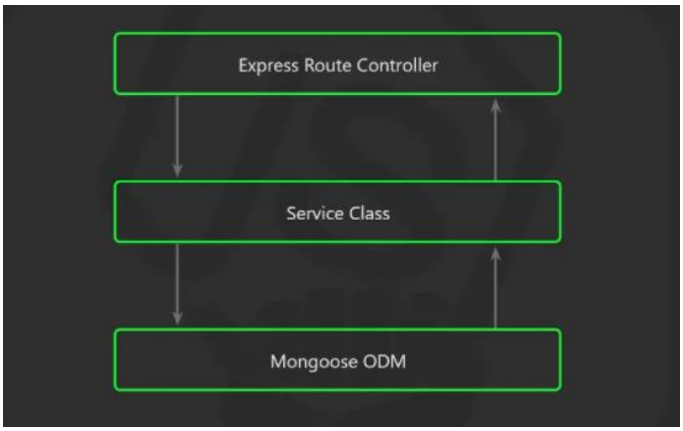


Figura 79 Esquema de capas API Node.js

La estructura de ficheros en la que se estructura la aplicación del servidor de este proyecto es la siguiente:

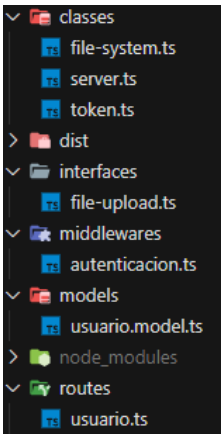


Figura 80 Estructura de ficheros de la aplicación del servidor

La capa relacionada con las rutas a las que se pueden acceder en la API REST mediante peticiones http es la correspondiente el archivo *usuario.ts* de la carpeta *routes* y la que contiene el acceso a la base de datos de usuario es la correspondiente con el archivo *usuario.model.ts* de la carpeta *models*. Por otro lado, las clases y servicios que contienen la lógica interna principal de la aplicación del servidor se encuentran en los archivos de las carpetas *classes, interfaces* y *middlewares.*

4.4.2 Diseño y explicación de uso de la interfaz

El objetivo de la parte del sistema correspondiente con la aplicación del servidor es ser un complemento a la aplicación móvil. Debido a esto la aplicación del servidor carece de una interfaz de usuario como tal, la única forma de interactuar con ella es mediante el uso de la aplicación móvil o el de una herramienta de testeo de API REST como es Postman, de la cual ya hemos hablado anteriormente.

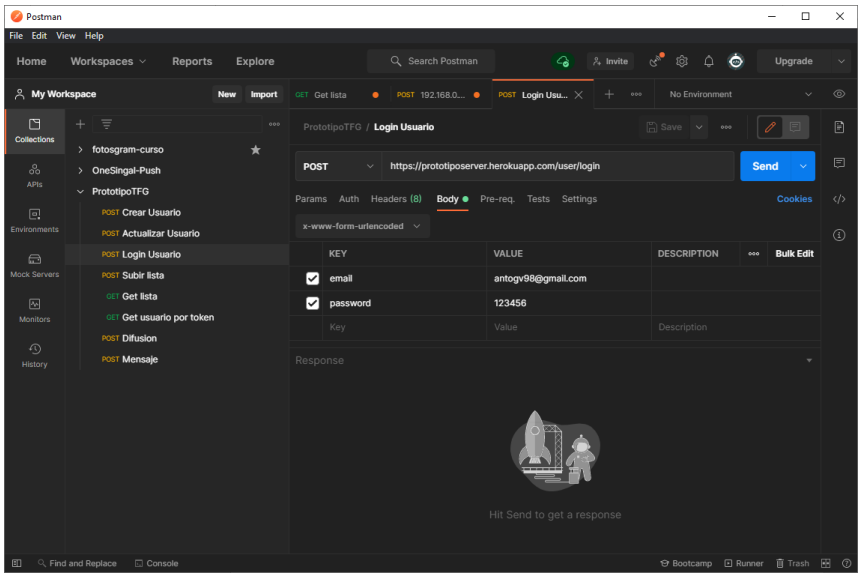


Figura 81 Ejemplo de interacción con la aplicación del servidor desde Postman

4.4.3 Aspectos destacados de la implementación

A continuación, se muestran los detalles relacionados con la implementación que otorga funcionalidad a la aplicación del servidor. La implementación se ve apoyada y reflejada en el siguiente diagrama de clases UML que traza claramente la estructura del sistema, al modelar sus clases, atributos y métodos, así como la relación entre las clases.

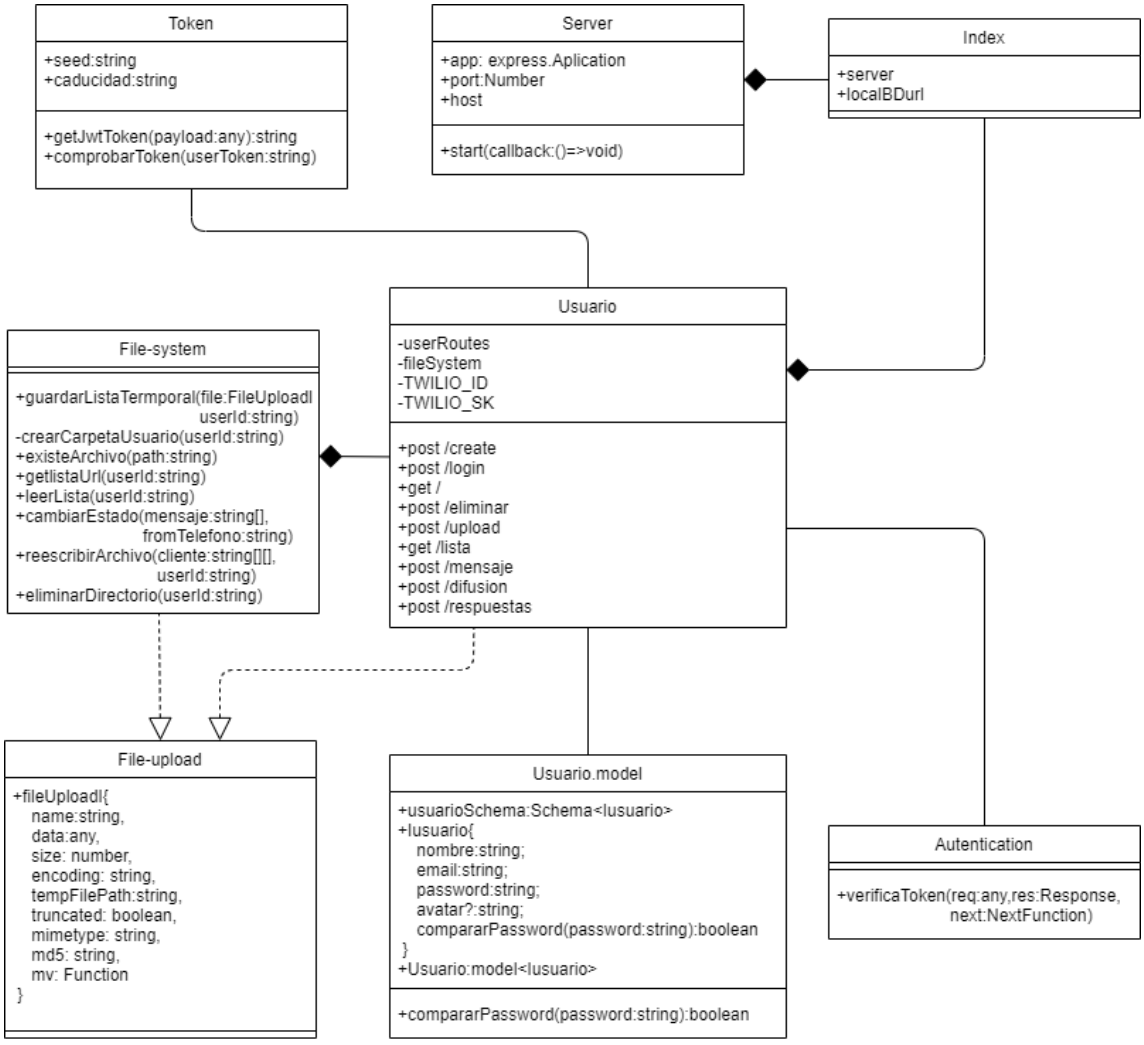


Figura 82 Diagrama de clases de la aplicación del servidor

Tras observar las distintas clases que componen la implementación de la parte del sistema correspondiente con la aplicación multiplataforma, se verá de forma más detallada como funciona internamente cada una de éstas.

4.4.3.1 Index

La clase *Index* es la clase ejecutada a la hora de levantar el servidor, ésta contiene los diferentes apartados, configuraciones y conexiones con la base de datos que realizará el servidor en el

momento que se encuentre corriendo. Como se puede observar se crea un objeto de la clase *Server* (que será vista más adelante), a la cual se le asignan una serie de propiedades que éste usará y una conexión con la base de datos mediante la librería *mongoose.*

```
const server = new Server();
//Body parser que procesa las peticiones y prepara el objeto para la D
server.app.use(bodyParser.urlencoded({extended:true}));
server.app.use(bodyParser.json());
//FileUpload
server.app.use(fileUpload());
//Configurar CORS para permitir peticiones cross domain
server.app.use(cors({origin: true,credentials:true}));
//Rutas de mi aplicación
server.app.use('/user',userRoutes);

//Conectar DB

//const localDBurl ='mongodb://localhost:27017/prototipoTFG';
const localDBurl ="mongodb+srv://root:root@cluster0.ydo1d.mongodb.net/

mongoose.connect(localDBurl,
                 {useNewUrlParser:true,useCreateIndex:true},(err)=>{
                     if (err) throw err;
                     console.log('Base de datos ONLINE');
                 });

//Levantar express
server.start(()=>{
    console.log('Servidor corriendo en puerto '+ server.port);
});
```

Figura 83 Clase Index

4.4.3.2 File-system

El método *guardarListaTemporal()* de encarga de colocar un archivo de tipo .csv con el nombre del id del usuario y en la carpeta correspondiente con el mismo id. Tanto el archivo como el id del usuario son recibidos por parámetro.

```
guardarListaTemporal(file: FileUploadI, userId: string ){

    return new Promise<void>((resolve,reject) =>{
                //Crear Carpetas
                const path = this.crearCarpetaUsuario(userId);
                const nombreArchivo =userId+'.csv';
                //Mover el archivo del Temp a nuestra carpeta
                file.mv(`${path}/${nombreArchivo}`, (err:any) =>{
                    if(err){
                        //no se pudo mover
                        reject(err);
                    }else{
                        //todo salio bien
                        resolve();
                    }
                });
    });

}
```

Figura 84 Método guardarListaTemporal() de File-system

Durante la ejecución del método anterior es llamado el método *crearCarpetaUsuario()*, el cual se encarga de obtener la ruta hasta la carpeta con el id del usuario y en caso de que no exista la crea y devuelve su ruta.

```
private crearCarpetaUsuario( userId: string){
    //Crea la ruta a la carpeta partiendo desde la estructura de direct
    const pathUser= path.resolve(__dirname, '../uploads/', userId);
    //Comprueba si existe la ruta
    const existe = fs.existsSync(pathUser);
    //En caso de que no exista es creada
    if(!existe){
        fs.mkdirSync(pathUser);
    }
    return pathUser;

}
```

Figura 85 Método crearCarpetaUsuario() de File-system

El método *existeArchivo()* se encarga de comprobar si la ruta recibida por parametro existe y en función de ello devuelve un booleano true o false. Este método es utilizado durante la ejecución del *getlistaUrl()*, el cual se encarga de a traves de un id de usuario recibido por parametro devolver la ruta hacia el archivo de lista de clientes asociado a él.

```
existeArchivo(path:string){
    //Comprueba si existe la ruta
    const existe = fs.existsSync(path);
    return existe;
}


//Devuelve el url de la lista que está asociada al userId proporcionado o en caso
getlistaUrl(userId:string){

    const nombreLista = userId+'.csv';
    //Obtiene la ruta del archivo partiendo desde la estructura de directorios de
    const pathLista= path.resolve(__dirname, '../uploads/', userId,nombreLista);
    const existe=this.existeArchivo(pathLista);

   if(!existe){
       return 'error';
   }

    return pathLista;

}
```

Figura 86 Métodos existeArchivo() y getlistaUrl() de File-system

El método *leerLista()* se encarga de mediante un id de usuario recibido por parametro, recorres las filas del archivo que contiene la lista de clientes asociada a éste y convertirlo en una matriz bidimensional en la que cada coordenada principal de la matriz contiene una lista de los datos de un cliente. La matriz bidimensional obtenida del archivo en este método es retornada.

```
leerLista(userId:string){
    const path = this.getlistaUrl(userId);
    //Lee el archivos pasandole la ruta y el formato del texto
    const leer=fs.readFileSync(path,'utf8');

    const rows=leer.split("\n");
    var clientes: string[][]=[];
    rows.forEach(row=>{
        clientes.push(row.replace("\r","").split(","));
    });
    return clientes;
}
```

Figura 87 Método leerLista() de File-system

El método *cambiarEstado()* se encarga de en función del valor obtenido en el mensaje recibido por parámetro, leer el archivo y crear la matriz bidimensional de la lista de clientes. Durante el recorrido al cliente con el mismo teléfono al recibido por parámetro se le cambia su estado. Una vez leída y modificada la lista se reescribe el archivo con el nuevo estado del cliente.

```
cambiarEstado(mensaje:string[],fromTelefono:string){
    const userId=mensaje[1];
    const path = this.getlistaUrl(userId);
    //Lee y escribe el archivos pasandole la ruta y el formato del t
    const leer=fs.readFileSync(path,'utf8');
    const rows:string[]=leer.split("\n");
    var clientes: string[][]=[];
    //Reccorre el array de rows y se guarda en el array de dos dimer
    //telefono coincide con el proporcionado es modificado
    rows.forEach(row=>{
        const replaceRow:string[]=row.replace("\r","").split(",");
        if(replaceRow[2]===fromTelefono){
            if(mensaje[0]==='CANCELAR'){
                replaceRow[3]="0";
            }else if(mensaje[0]==='REACTIVAR'){
                replaceRow[3]="1";
            }
        }
        clientes.push(replaceRow);
    });
    //Reescribe el archivo correspondiente con el userId con su nuev
    this.reescribirArchivo(clientes,userId);

}
```

Figura 88 Método cambiarEstado() de File-system

El método *reescribirArchivo() mediante una matriz bidimensional y el id del usuario, recibidos por parámetro, da formato obtiene la ruta al directorio del archivo asociado al usuario y genera un string con la información que más tarde es escrita en el archivo con la ruta obtenida.*

```
reescribirArchivo(clientes:string[][],userId:string){
    const path = this.getlistaUrl(userId);
    var nuevoTexto='';
    var indice=0;
    clientes.forEach(registro=>{
        //Guarda el toString del array de la primera dim
        nuevoTexto+=registro.toString();
        if(indice<(clientes.length-1)){
            nuevoTexto+='\n';
        }
        indice++;
    });
    //Crea o modifica un archivo existente en la ruta pr
    fs.writeFileSync(path,nuevoTexto,{encoding:'utf8'});
}
```

Figura 89 Método reescribirArchivo() de File-system

El método *eliminarDirectorio()* obtiene la ruta del archivo correspondiente con el id del usuario, obtenido por parametro, y si esta ruta existe la elimina.

```
eliminarDirectorio(userId:string){
    const path = this.getlistaUrl(userId);
    if(this.existeArchivo(path)){
        fs.unlinkSync(path);
    }

}
```

Figura 90 Método eliminarDirectorio() de File-system

4.4.3.3 Token

La clase *Token* solo contiene los métodos *getJwtToken()* y *comprobarToken()*. El primero de los métodos se encarga de crear un toke con una caducidad determinada y mediante un payload recibido por parámetro, una semilla. El segundo método recibe un token y mediante un proceso inverso con este token y la semilla comprueba si es válido o no y en caso de serlo devuelve la decodificación.

```
static getJwtToken(payload:any):string{
    return jwt.sign({usuario:payload}, this.seed,{expiresIn:this.caducidad});
}

//Comprueba el token recibido con nuestra seed para comprobar si es valido
static comprobarToken (userToken:string){

    return new Promise((resolve,reject)=>{
        jwt.verify(userToken, this.seed, (err,decoded)=>{
                    if(err){
                        //no confiar
                        reject();
                    }else{
                        //token valido
                        resolve(decoded);
                    }
                });
    });


}
```

Figura 91 Métodos getJwtToken() y comprobarToken() de Token

4.4.3.4 Autenticacion

El único método de la clase *Autenticacion* es *verificaToken()* y se encarga de obtener el token que se encuentra en el header *x-token*, llamar al método *comprobarToken()* de la clase *Token* y en caso de recibir el token decodificado, devolverlo como el objeto usuario en la respuesta.

```
export const verificaToken = (req : any,res: Response,next: NextFunction) =>{

    //Obtiene la propiedad x-token enviada por el url
    const userToken = req.get('x-token') || '';
        //Decodifica el token y si obtiene un usuario permite continuar con el
        Token.comprobarToken(userToken)
        .then((decoded: any) =>{
            console.log('Decoded',decoded);
            req.usuario = decoded.usuario;
            next();
        }).catch(err =>{
            res.json({
                ok:false,
                mensaje:'Token no es correcto'
            })
        });

}
```

Figura 92 Método verificaToken() de Autenticacion

4.4.3.5 Usuario.model

El único método de la clase *Usuario.model* es *compararPassword()* y se encarga de realizar una comparación entre el pasword recibido por parámetro y el password global de la clase, mediante la librería de encripatación *bcrypt*. En función del resultado devuelve un booleano

```
usuarioSchema.method('compararPassword', function(password:string = ''):boolean{
    if(bcrypt.compareSync(password, this.password)){
        return true;
    }else{
        return false;
    }
});
```

Figura 93 Método compararPassword() de Usuario.model

4.4.3.6 Usuario

La ruta post */create* debe de recibir los parámetros nombre, email, password y avatar. Una referencia usuario es creada en la base de datos y a dicho usuario se le genera un token mediante el método *getJwtToken()* de la clase *Token.* Si la petición se completa correctamente se devuelve el token del usuario en la respuesta.

```
userRoutes.post('/create', (req: Request, res: Response)=>{

    const user = {
        nombre: req.body.nombre,
        email: req.body.email,
        password: bcrypt.hashSync(req.body.password,10) ,
        avatar:req.body.avatar
    };

    Usuario.create(user).then(userDB =>{
        const tokenUser = Token.getJwtToken({
            _id: userDB._id,
            nombre:userDB.nombre,
            email:userDB.email,
            avatar:userDB.avatar
        });

       res.json({
           ok:true,
           token:tokenUser
       }) ;
    }).catch(err=>{
        res.json({
            ok:false,
            err
        });
    });

});
```

Figura 94 Ruta post /create de Usuario

La ruta post */login* debe de recibir los parámetros email y password. Se comprueba si existe algún usuario en la base de datos con el email recibido y si es así se compara el password. En caso de pasar dichas comprobaciones se genera el token del usuario y se devuelve en la respuesta.

Figura 95 Ruta post /login de Usuario

La ruta get / recibe el token del usuario en la petición y éste devuelve el usuario correspondiente decodificado en el método *verificaToken()* de *middleware*.

Figura 96 Ruta get / de Usuario

La ruta post */eliminar* recibe en la petición el token del usuario y éste llama al método *eliminarDirectorio()* con el id del usuario con ese token, obtenido mediante *verificaToken()* de *middleware,* este método elimina el directorio de este usuario. Tras esto se elimina el usuario de la base de datos.

Figura 97 Ruta post /eliminar de Usuario

La ruta post */upload* obtiene el usuario correspondiente con el token recibido en la petición mediante *verificaToken* de *middleware*. Tras esto compruba si se recibió un archivo en la petición y en caso de ser así se comprueba su formato, si es correcto se llama al método *guardarListaTemporal() de* File-system. Por último se devuelve el tipo del archivo.

```
userRoutes.post('/upload', [verificaToken], async (req: any, res:Response) =>{

    if(!req.files){
        return res.status(400).json({
            ok:false,
            mensaje:'No se subió ningun archivo'
        });
    }

    const file:FileUploadI = req.files.lista;

    if(!file){
        return res.status(400).json({
            ok:false,
            mensaje:'No se subió ningun archivo - lista'
        });
    }

    if(!file.mimetype.includes('csv')){
        return res.status(400).json({
            ok:false,
            mensaje:'Lo que se subió no es un archivo csv'
        });
    }

    await fileSystem.guardarListaTemporal(file, req.usuario._id);

    res.json({
        ok:true,
        file: file.mimetype
    });


});
```

Figura 98 Ruta post /upload de Usuario

La ruta get */lista* obtiene el usuario correspondiente con el token recibido en la petición mediante *verificaToken* de *middleware*. Tras esto llama al método *getlistaUrl()* de *Fyle-system,* éste devuelve la ruta del archivo y éste es enviado en la respuesta.

```
userRoutes.get('/lista', [verificaToken],(req:any, res:Response)=>{

    const userId=req.usuario._id
    const pathLista = fileSystem.getlistaUrl(userId);
    res.sendFile(pathLista);

});
```

Figura 99 Ruta get /lista de Usuario

La ruta post */mensaje* obtiene el usuario correspondiente con el token recibido en la petición mediante *verificaToken* de *middleware*. Tras esto se compruba si se recibe en la petición un elemento *telefono*, si es así se llama al método *messages.create()* de la API de *Twilio,* el cual envía un mensaje por Whatsapp al número de teléfono recibido en la petición. Por ultimo se devuelve en la respuesta el usuario y el número de teléfono de vuelta.

```
userRoutes.post('/mensaje', [verificaToken], async (req: any, res:Response) =>{

    if(!req.body.telefono){
        return res.status(400).json({
            ok:false,
            mensaje:'No se envió ningun numero de telefono'
        });
    }

    const client = new Twilio(TWILIO_ID,TWILIO_SK);

    client.messages
    .create({
       from: 'whatsapp:+14155238886',
       body: 'Su pedido será entregado en menos de 1 hora',
       to: 'whatsapp:'+req.body.telefono
     })
    .then(message => console.log(message.sid));

    res.json({
            ok:true,
            mensaje: 'El mensaje se envió correctamente',
            usuario:  req.usuario,
            telefono: req.body.telefono
        });

});
```

Figura 100 Ruta post /mensaje de Usuario

La ruta post */difusion* obtiene el usuario correspondiente con el token recibido en la petición mediante *verificaToken* de *middleware*. Tras esto se lee la lista de clientes del usuario y se recorre, llamando al método *messages.create()* de la API de *Twilio,* el cual envía un mensaje por Whatsapp al número de teléfono del cliente. Por ultimo se devuelve en la respuesta el usuario.

```
userRoutes.post('/difusion', [verificaToken], async (req: any, res:Response) =>{


    const userId=req.usuario._id
    var lista = [];
    lista=fileSystem.leerLista(userId);
    console.log(lista);

    lista.forEach(cliente=>{

        const client = new Twilio(TWILIO_ID,TWILIO_SK);

        client.messages
        .create({
           from: 'whatsapp:+14155238886',
           body: 'Su pedido será entregado mañana entre las 8:00 y las 14:00 ID repartidor
           to: 'whatsapp:'+cliente[2]
         })
        .then(message => console.log(message.sid));
    });

    res.json({
            ok:true,
            mensaje: 'La difusión se envió correctamente',
            usuario:  req.usuario,
```

Figura 101 Ruta post /difusion de Usuario

La ruta post */respuestas* recibe la petición por parte de *Twilio* con la información del remitente y el mensaje que ha sido recibido, en este caso enviado por el cliente. Tras esto se obtiene el teléfono del remitente y el id del usuario que se encuentra en el cuerpo del mensaje. En función del cuerpo del mensaje recibido se envía un mensaje de respuesta al cliente u otro y se llama al método *cambiarEstado*() de *File-system* con un valor por parametro determinado. Por último se envia la respuesta de la petición con un mensaje informativo.

```
userRoutes.post('/respuestas', async (req: any, res:Response) =>{

    const mensaje=req.body.Body.split('-');
    const from = req.body.From.split(':');
    const fromTelefono=from[1];
    const path=fileSystem.getlistaUrl(mensaje[1]);
    const existelistaUsuario=fileSystem.existeArchivo(path);
    const client = new Twilio(TWILIO_ID,TWILIO_SK);

    if(mensaje[0]==='CANCELAR'&&existelistaUsuario){
        client.messages
        .create({
           from: 'whatsapp:+14155238886',
           body: 'Hemos cancelado la entrega asociada al número de teléfono '+req.body.F
           to: req.body.From
         })
        .then(message => console.log(message.sid));
        res.json({
            ok:true,
            mensaje: 'La respuesta se recibio correctamente'
        });
        fileSystem.cambiarEstado(mensaje,fromTelefono);
    }else if (mensaje[0]==='REACTIVAR'&&existelistaUsuario){
        client.messages
        .create({
           from: 'whatsapp:+14155238886',
           body: 'Hemos reactivado la entrega asociada al número de teléfono '+req.body.
           to: req.body.From
         })
        .then(message => console.log(message.sid));
        res.json({
            ok:true,
            mensaje: 'La respuesta se recibio correctamente'
        });
        fileSystem.cambiarEstado(mensaje,fromTelefono);
    }else{
        res.json({
            ok:false,
            mensaje: 'La respuesta no fue procesada'
        });
    }

});
```

Figura 102 Ruta post /respuesta de Usuario

4.4.4 Despliegue

Para el despliegue de la parte del sistema correspondiente con la aplicación del servidor y la base de datos se ha optado por servicios básicos con planes gratuitos. En el caso de que se comenzase a obtener beneficios gracias al uso del sistema se estudiarían otras posibilidades de pago con mejores características.

Para el despliegue online de la base de datos se ha optado por el servicio oficial de MongoDB Atlas, el cual proporciona 512 MB de almacenamiento gratuito y RAM y CPU compartida con otro usuario suscritos al plan gratuito de MongoDB Atlas.

El despliegue online de la aplicación junto con el almacenamiento de las listas de clientes corre a cargo de Heroku. El plan gratuito de Heroku ofrece 512MB de memoria RAM, sincronización con Git y un máximo de 2 procesos simultáneos. Cada 30 minutos de inactividad la aplicación entra en modo suspensión y la próxima llamada al servidor que lo despierte puede llegar a tardar unos segundos.

5. CONCLUSIONES Y LÍNEAS DE TRABAJO FUTURAS

5.1 Conclusiones

Con la finalización de este proyecto se ha conseguido desarrollar una aplicación híbrida para la gestión de los clientes de los repartidores, con funcionalidades útiles para la agilización de repartos e interconectada con un servidor para, mediante la gestión de usuarios, poder trabajar con múltiples dispositivos desplegados en una flota de repartidores.

Todos los objetivos que se relataron al inicio de este proyecto han sido alcanzados y con ello se han conseguido una serie de objetivos relacionados con los conocimientos y formación adquiridos sobre éstas que tecnologías son muy utilizadas en la actualidad y dan grandes resultados.

- Ser capaz de desarrollar un api REST con Express para dar soluciones en red a las peticiones de datos.
- Aprender las tecnologías de Ionic y Angular.
- Utilizar paquetes de Node.js como herramienta para crear funcionalidades en el servidor.
- Incrementar conocimientos en TypeScript tanto para el frontend como el backend.
- Tener un acercamiento al uso de bases de datos con MongoDB.
- Mejorar la capacidad de dar solución a problemas.
- Aumentar capacidades de redacción de proyectos.

Éstas entre muchos otros han sido los aprendizajes obtenidos durante la realización del proyecto.

5.2 Líneas de trabajo futuras

La finalización de este proyecto no significa la imposibilidad de continuar trabajando en él, dándole más funcionalidad y arreglar los problemas que vayan apareciendo con el tiempo.

En el caso de que se continuase con el proyecto algunas posibles nuevas funcionalidades y mejoras serían las siguientes:

- Creación de una interfaz administrativa del apartado del servidor.
- Mejora de la gestión de usuarios y aumento de los roles que estos pueden tener.
- Creación un sistema de cálculo de la mejor ruta a seguir, recorriendo la localización de todos los clientes.
- Mejora del manejo de excepciones y aparición más mensajes de aviso con los posibles errores de uso de la aplicación para ayudar a los usuarios.

Bibliografía

[1] C LOGÍSTICA, "La inversión en logística supera los 1.300 M€". Recuperado el de 20 de abril de 2021 de: https://logistica.cdecomunicacion.es/noticias/sectoriales/34817/la-inversion-en-logistica-supera-los-1-300-m

[2] MYSOLUTIONS DIGITAL FACTORY, "¿Qué es Ionic? Framework". Recuperado el 21 de abril de 2021 de: https://web.mysolutions.cl/blog/que-es-ionic/#:~:text=Las%20principales%20ventajas%20que%20ofrece%20son%3A&text=Una%20herramienta%20tan%20%E2%80%9Creciente%E2%80%9D%20como,EmberJS%20o%20KnockOut%20por%20ejemplo

[3] DRAUTA, "¿Qué es Node.js y para qué sirve?". Recuperado el 21 de abril de 2021 de: https://www.drauta.com/que-es-nodejs-y-para-que-sirve

[4] MDN WEB DOCS, "Introducción a Express/Node". Recuperado el 21 de abril de 2021 de: https://developer.mozilla.org/es/docs/Learn/Server-side/Express_Nodejs/Introduction

[5] GENBETA, "MongoDB: qué es, cómo funciona y cuándo podemos usarlo (o no)". Recuperado el 22 de abril de 2021 de: https://www.genbeta.com/desarrollo/mongodb-que-es-como-funciona-y-cuando-podemos-usarlo-o-no

[6] SENDCLOUD, "Empresas de transporte en España: ¿Correos, Correos Express, SEUR, DHL Express o UPS?". Recuperado el 24 de abril de 2021 de: https://www.sendcloud.es/empresas-de-transporte-cual-es-mejor/

[7] MARKETING 4 ECOMMERCE, "Historia de Seur: el decano de la mensajería en España cumple 75 años". Recuperado el 24 de abril de 2021 de: https://marketing4ecommerce.net/historia-de-seur-el-decano-de-la-mensajeria-en-espana-cumple-75-anos/

[8] MARKETING 4 ECOMMERCE, "Historia de MRW: 40 años de evolución en la logística española". Recuperado el 24 de abril de 2021 de: https://marketing4ecommerce.net/historia-de-mrw/

[9] MARKETING 4 ECOMMERCE, "Amazon ya es el mayor partner de paquetería de… Amazon: 3.500 millones de envíos repartidos en todo el mundo". Recuperado el 24 de abril de 2021 de: https://marketing4ecommerce.net/amazon-ya-es-el-mayor-partner-de-paqueteria-de-amazon-3-500-millones-de-envios-repartidos-en-todo-el-mundo/

[10] APACHE CORDOVA, "Documentación oficial". Recuperado el 25 de abril de 2021 de: https://cordova.apache.org/

[11] MDN WEB DOCS, "HTML5". Recuperado el 25 de abril de 2021 de: https://developer.mozilla.org/es/docs/Web/Guide/HTML/HTML5

[12] HOSTINGER TUTORIALES, "¿Qué es CSS?". Recuperado el 25 de abril de 2021 de: https://www.hostinger.es/tutoriales/que-es-css

[13] TYPESCRIPTLANG, "What is TypeScript?". Recuperado el 25 de abril de 2021 de: https://desarrolloweb.com/articulos/introduccion-a-typescript.html#:~:text=La%20caracter%C3%ADstica%20fundamental%20de%20TypeScript,su%20c%C3%B3digo%20a%20Javascript%20com%C3%BAn. https://developer.mozilla.org/es/docs/Web/JavaScript

[14] COMUNICAWEB, "¿Pagarás por usar la API de Google Maps?". Recuperado el 26 de abril de 2021 de: https://comunica-web.com/blog/marketing-digital/api-google-maps-coste/

[15] CAPTERRA, “Twilio”. Recuperado el 26 de abril de 2021 de: https://www.capterra.es/software/180158/twilio-communications-platform
[16] ITPRO, “Visual Studio Code ¿Qué es? y ¿Qué no es?”. Recuperado el 26 de abril de 2021 de: https://blogs.itpro.es/eduardocloud/2016/08/22/visual-studio-code-que-es-y-que-no-es/
[17] OPEN WEBINARS, “Qué es Postman y para qué sirve”. Recuperado el 26 de abril de 2021 de: https://openwebinars.net/blog/que-es-postman/
[18] OPEN WEBINARS, “Qué es GIT y para qué sirve”. Recuperado el 25 de abril de 2021 de: https://openwebinars.net/blog/que-es-git-y-para-que-sirve/
[19] XATAKA, “Qué es Github y qué es lo que le ofrece a los desarrolladores”. Recuperado el 26 de abril de 2021 de: https://www.xataka.com/basics/que-github-que-que-le-ofrece-a-desarrolladores
[20] DESARROLLOWEB, “Ionic CLI”. Recuperado el 26 de abril de 2021 de: https://desarrolloweb.com/articulos/ionic-cli.html#:~:text=Ionic%20CLI%20es%20el%20int%C3%A9rprete,de%20aplicaciones%20con%20Ionic%202.
[21] HIPERTEXTUAL, “Todo lo que puedes hacer con «inspeccionar elemento» en Google Chrome, Firefox y Safari”. Recuperado el 26 de abril de 2021 de: https://hipertextual.com/2018/07/inspeccionar-elemento-google-chrome-firefox-safari
[22] PLATZI, “¿Qué es Heroku y para qué me sirve?”. Recuperado el 26 de abril de 2021 de: https://platzi.com/blog/que-es-heroku/
[23] PARADIGMA DIGITAL, “Compass será a MongoDB como Toad a SQL.”. Recuperado el 26 de abril de 2021 de: https://www.paradigmadigital.com/dev/compass-sera-mongodb-toad-sql/
[24] IDA BLOG, “MongoDB Atlas: El salto a la nube”. Recuperado el 26 de abril de 2021 de: https://blog.ida.cl/desarrollo/mongodb-atlas-el-salto-a-la-nube/
[25] MONGOOSE, “Documentación oficial”. Recuperado el 6 de mayo de 2021 de: https://mongoosejs.com/docs/
[26] DEV, “Node.js Express Login example with MongoDB”. Recuperado el 7 de mayo de 2021 de: https://dev.to/tienbku/node-js-express-login-example-with-mongodb-2ofc
[27] LEOMARIS REYES, “Aplicando el patrón de diseño MVVM”. Recuperado el 8 de mayo de 2021 de: https://medium.com/@reyes.leomaris/aplicando-el-patr%C3%B3n-de-dise%C3%B1o-mvvm-d4156e51bbe5
[28] CURSO DE INTERACCIÓN PERSONA-ORDENADOR, “Prototipos de Papel (Paper Prototyping)”. Recuperado el 8 de mayo de 2021 de: https://mpiua.invid.udl.cat/prototipos-de-papel-paper-prototyping/
[29] SOFTWARE ON THE ROAD “Una arquitectura a prueba de balas para proyectos Node.js”. Recuperado el 9 de mayo de 2021 de: https://softwareontheroad.com/es/ideal-nodejs-project-structure/

UNIVERSIDAD DE ALMERÍA

Resumen/Abstract

Hoy en día los servicios de mensajería son cada vez más utilizados y por ello necesitan estar actualizados a las últimas tecnologías para facilitar su desempeño. Es de aquí donde nace la idea de este proyecto, la creación de una app multiplataforma de automatización de mensajes a través de GPS orientado a empresas de reparto. Esta herramienta facilita y hace más seguro el trabajo de los repartidores, permitiéndoles agilizar el proceso de las entregas sin necesidad de hacer uso de su dispositivo durante la conducción. Mediante el cálculo de distancias entre ellos y las entregas a través de GPS se enviarán mensajes a una determinada distancia de los clientes a través de "Whatsapp", todo esto sumado a una interfaz cómoda para la gestión de sus repartos y centralización de los datos con un servidor.

Today messaging services are increasingly used and therefore need to be updated to the latest technologies to facilitate their performance. This is where the idea of this project was born, the creation of a multiplatform app for automating messages through GPS aimed at delivery companies. This tool makes the work of delivery men easier and safer, allowing them to streamline the delivery process without having to use their device while driving. By calculating the distances between them and the deliveries through GPS, messages will be sent to a certain distance from the clients through "WhatsApp", all this added to a comfortable interface for managing their deliveries and centralizing the data with A server.

GRADO EN INGENIERÍA INFORMÁTICA, 2020/2021