Escuela Superior
de Ingeniería y Tecnología
Universidad de La Laguna

Trabajo de Fin de Grado

Desarrollo multiplataforma

Multiplatform development

Richard Morales Luis

La Laguna, 5 de julio de 2020

Dña. **Jezabel Miriam Molina Gil**, con N.I.F. 78.507.682-B profesora Ayudante Doctor adscrita al Departamento de Ingeniería Informática y de Sistemas de la Universidad de La Laguna, como tutora.

,

Dña. **Yanira González González**, con N.I.F. 78.555.933-P Doctora en Informática, contratada como Técnico de Grado Medio en el Servicio de Sistemas de Información del Complejo Hospitalario Universitario de Canarias, como cotutora.

C E R T I F I C A (N)

Que la presente memoria titulada:

”*Desarrollo multiplataforma*”

ha sido realizada bajo su dirección por D. **Richard Morales Luis**, con N.I.F. 79.073.156-F.

Y para que así conste, en cumplimiento de la legislación vigente y a los efectos oportunos firman la presente en La Laguna a 5 de julio de 2020

Agradecimientos

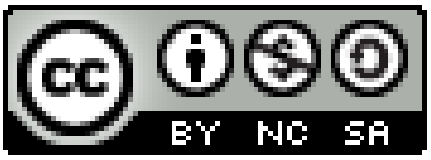
Quiero agradecer en primer lugar a mis dos tutoras Jezabel Miriam Molina Gil y a Yanira González González por todo el apoyo y facilidades que me han brindado para desarrollar este trabajo. De la misma manera quiero agradecer a la Comunidad de Regantes Alto los Baldíos por su compromiso con el proyecto y toda la información que han aportado para que este trabajo se llevara acabo.

Por otro lado, quiero agradecer tanto a mis compañeros de carrera como a muchos profesores por haberme enseñado tanto y haberme acompañado en estos años hasta convertirme en un profesional.

Agradecer por supuesto a mi familia y amigos por el apoyo durante todos estos años.

Finalmente agradecer a mis compañeros de la empresa Nologis, por haberme dado la libertad de poder compaginar la realización de este trabajo con mi desempeño laboral cada día.

Licencia



Resumen

Las comunidades de regantes tienen un papel fundamental en el suministro de agua para los agricultores. Su labor es garantizar a estos un servicio económico, cuyo fin es eliminar la responsabilidad de gestión del agua y que puedan únicamente dedicar su tiempo al cultivo de sus campos.

Este trabajo consistirá en la realización de un proyecto, para La Comunidad de Regantes Alto los Baldíos, que tiene como objetivo proponer y desarrollar una solución para mejorar la recogida y gestión de datos, de cada uno de los contadores que tienen los socios (comuneros) que forman parte de dicha comunidad.

Palabras clave: Comunidad de Regadío, comunero, canalero, contador, válvula.

Abstract

Irrigation communities have always played a key role in supplying water for farmers. Its labor is to guarantee the latter, an economic service whose purpose is to eliminate the responsibility of water management, so that they can dedicate their time to cultivate their fields.

This work will consist in the development of a project for the Comunidad de Regantes Alto los Baldíos, the purpose is to propose and develop a solution to improve the gathering and management data from each meter, the partners have.

Keywords: Irrigation Community, gutter, water meter, valve, community member

Índice general

Índice de Figuras

Índice de Tablas

Capítulo 1

Introducción

1.1. La agricultura en Canarias

Hasta hace unas décadas, Canarias fue una sociedad fundamentalmente agraria, y aunque en la actualidad la agricultura solo cubre una parte de las necesidades alimenticias de la región, algunos cultivos son reconocidos por su prestigio y juegan un papel muy importante en el sector. También es cierto, que la agricultura del archipiélago está situada en una continua inestabilidad, tanto estructural como comercial. Esto es así, porque existen otros sectores que actualmente se consideran más atractivos, lo que ha provocado que queden zonas de campo desatentidas.

A pesar de lo nombrado anteriormente, la agricultura sigue siendo un campo sumamente importante en el desarrollo económico del archipiélago. Para ello, los agricultores trabajan cada día en sus cultivos, a fin de generar un producto de calidad que de valor a los cultivos locales.

Para que las labores de los agricultores tengan un buen resultado, estos deben contar con diferentes sistemas de riego, que les permita abastecer sus cultivos. Muchos de estos agricultores pertenecen a las conocidas comunidades de regantes. Estas se pueden definir, como una agrupación de todos los propietarios de una zona regable, que se unen por Ley, para la administración autónoma y común de las aguas públicas. Actualmente, juegan un papel fundamental en las gestión de este recurso, puesto que distribuyen más del 70 % del agua en al superficie del país.

Cuando hablamos en términos de comunidades de regantes surgen diferentes conceptos y figuras:

- **Comunero**: Agricultor asociado a la comunidad de regantes.

- **Canalero**: Persona contratada por la comunidad que se encarga de registrar los datos mensuales del gasto de agua de cada comunero/agricultor.
- **Contador**: Dispositivo mediante el cual, se realiza la medida de agua utilizada por cada comunero/agricultor.

1.2. Comunidad de Regantes Alto los Baldíos

En la actualidad, existen diferentes comunidades de regantes que ayudan a los agricultores y ganaderos, gestionando el agua que consumen y necesitan en el día a día. En Tenerife concretamente existen más de 50 comunidades situadas alrededor de toda la isla.

Este trabajo, se centra en una de estas comunidades en concreto, la *Comunidad de Regantes Alto los Baldíos*, situada en el municipio de San Cristóbal de la Laguna. Esta comunidad fue fundaba en el año 2017 por lo que cuenta con 3 años de antigüedad. Actualmente la comunidad se encarga de suministrar agua a sus socios, distribuidos en 3 municipios diferentes: el municipio de Santa Cruz de Tenerife, el municipio de San Cristóbal La Laguna y el municipio de El Rosario.

Por otro lado, cuentan con un número total de 340 socios (*comuneros*) con sus respectivos 340 contadores. Además, esta comunidad en concreto cuenta con 4 válvulas que se encargan de gestionar y medir la cantidad de agua que pasa por las diferentes zonas en las que se encuentran. Estas son utilizadas con el fin de gestionar de manera más precisa, el agua que pasa por los contadores asociados a estas válvulas. Por ejemplo, si queremos cortar el agua de una zona concreta bastaría con cerrar esa válvula y por ello, los contadores que esten relaciados a esta última dejarían de recibir agua hasta que se volviera abrir.

Las comunidades de regadíos como comentamos en el punto anterior, se encargan de suministrar agua a sus socios de una manera sencilla y con un coste económico adecuado a las necesidades de cada uno de estos. Para ello, cada mes se realiza una medición completa de todos y cada uno de los contadores de los comuneros que pertenecen a la comunidad.

1.3. Herramientas de gestión actuales

1.3.1. Medida de los contadores

La labor de los canaleros es, probablemente, una de las clave de que la comunidad mantenga su consistencia, pues los datos que recogen son la base para tomar decisiones de cara al futuro de la misma. Actualmente existen diferentes aplicaciones que intentan solventar este problema:

- **Vi2Water:** Vi2Water es una aplicación desarrollada por la empresa Vi2WEB dedicada al desarrollo de software a medida, especializada en el diseño e implementación de herramientas de gestión de la información con componentes geográficos. Dicha aplicación se encarga, una vez introducidos los datos del suministrador y los contadores, de poder realizar una medición de los mismos así como localizarlos.[1]
- **Contadores con tecnología de ultrasonidos:** La tecnología tambien ha llegado al sector de la agricultura. Un ejemplo de ello, son los contadores de la empresa W-Sonic Technology la cual ha creado los llamados **Ultrimis W**, un conjunto de contadores capaces de determinar la cantidad de agua que pasa por ellos utilizando tecnologías de ultrasonidos. Toda esta información es registrada y puede ser visualizada por ejemplo en una aplicación de visualización de datos. [2]

1.3.2. Representación de los datos

Para la representación de los datos obtenidos en las medidas no existe una aplicación referente como tal en el mercado actualmente. Esto se debe a que cada comunidad o cada empresa realiza desarrollos a medida para los casos específicos de cada comunidad. Pueden existir aplicaciones de escritorio como aplicaciones web que se encargan de la gestión tanto de facturación como de control de contadores, así como solamente aplicaciones diferentes para cada caso. Para este trabajo será una aplicación de apoyo al software de facturación que la comunidad utiliza.

1.4. Objetivos del Trabajo

1.4.1. Proceso actual de facturación

Como se ha nombrado anteriormente este trabajo está basado en resolver una problemática real que tiene la Comunidad de Regantes Alto

los Baldíos. Actualmente, la comunidad cuenta con un único programa de facturación el cual se encarga de generar mensualmente, las facturas para cada uno de sus comuneros, en base al consumo de agua mensual que hayan tenido y a la cantidad que tengan contratada.

Para ello, a principio de cada mes los canaleros se encargan de ir contador por contador, apuntando en una cuadrícula de excel impresa, los datos medidos en ese instante. Una vez se tienen los datos en papel este es entregado a la responsable de administración, cuya tarea es introducir los datos en el programa de facturación y generar el recibo correspondiente.

1.4.2. Problemas encontrados

Tras una análisis de la mano con la responsable de la administración de la comunidad se han detectado diferentes problemas que se intentarán resolver en este trabajo:

- **Localización de los contadores:** La continúa variación del personal encargado de realizar las medidas de los contadores, supone un problema pues no tienen que ser personas relacionadas con el sector o con la zona. Por ello, saber donde se encuentra cada contador es uno de los grandes problemas que existen, y el cual actualmente se resuelve el primer día de trabajo de cada nuevo canalero, donde el personal de administración realiza la primera ruta con el nuevo trabajador a fin de mostrarle la posición exacta de cada contador.
- **Erratas en la lectura de agua:** Al realizarse la recogida de datos de usando papel, se dan casos donde el canalero puede confundirse a la hora de medir, estos errores se suele solventar tachando la medida y ensuciando el documento de registro, que la administración luego tiene que leer. En la misma línea, al ser tomada por personas diferentes la manera de escribir puede suponer un problema debido a malas interpretaciones, por ejemplo, en los datos ya que no se diferencia un numero de otro y por ello la medida puede llevar a confusión.
- **Perdida de referencia en la facturación:** Una vez la administración posee los datos, debe pasarlos al software de facturación donde suelen producirse errores. Por ejemplo, al tener que cambiar la vista del papel al programa se pierde la referencia de la lectura que se estaba introduciendo y retrasa mucho esta tarea.

- **La información no es consistente:** Como toda esta labor de medidas se realiza en papel, puede producirse la pérdida de información a largo plazo al poder extraviarse, o puede degradarse y dejar de entenderse que datos se habían tomado.

La idea principal de este trabajo reside, a grandes rasgos, en digitalizar toda la información que gestiona la comunidad de regantes para evitar los problemas descritos anteriormente.

1.5. Estructura de la memoria

El presente documento se organiza en los siguientes capítulos:

- Capítulo 1: Introducción.
  En este capítulo, se presetna una introducción al problema a solucionar.
- Capítulo 2: Resolución del problema.
  Se describirá la propuesta que se ha realizado para intentar resolver los problemas descritos en el capítulo 1. Además, se realizará un estudio de las tecnolgías que se podrían utilizar para resolverlo y la elección de cual se ha utilizado.
- Capítulo 3: Desarrollo del trabajo.
  En este capítulo, se expondrá la manera en la cual se ha desarrollado el trabajo y cuál ha sido el resultado final de ambas aplicaciones.
- Capítulo 4: Presupuesto.
  Se incluyen dos tablas con la estimación de costes del proyecto y las horas invertidas en cada actividad.
- Capítulo 5: Conclusiones y líenas futuras.
- Capítulo 6: Conclusiones y líenas futuras.

Capítulo 2

Resolución del problema

En el capítulo anterior se hizo una introducción del escenario sobre el cual se va a desarollar este trabajo. En este capítulo se describirá la propuesta que se ha realizado para intentar resolver los problemas descritos anteriormente.

2.1. Propuesta.

En vista de la propuesta principal del trabajo cuyo cometido consistirá en digitalizar toda la información de la comunidad de regantes se han realizado dos propuestas para abordarlo. Por un lado la recogida de datos de los contadores y por otro, la representación de los mismos antes de pasarlos al programa de facturación.

2.1.1. Recogida de información

Para abordar la problemática de la recogida de información tras varios estudios se ha decidido crear una **aplicación móvil** la cual sea capaz de:

- Localizar los contadores en su posición exacta.
- Permitir tomar medidas de estos contadores.
- Saber de manera rápida qué contadores han sido medidos y cuáles faltan por medir.
- Permitir localizar nuevos contadores o válvulas en caso de que sea necesario.

El objetivo de que sea una aplicación móvil reside en que se necesita que el usuario pueda moverse con libertad a la hora de recoger dicha información de los contadores. Además, deberá contar con un mapa que sea capaz de localizar todos los contadores para que aunque el usuario no sepa donde están, pueda encontrarlos. Por otro lado, al realizar una aplicación móvil esta podría adaptarse a dispositivos un poco más grandes como puede ser una *tablet* a fin de poder visualizar de una mejor manera la interfaz.

El punto fuerte de la aplicación debe residir en la sencillez. Esto viene dado porque se ha detectado que el *target* de quien utilizará la aplicación, son personas que no están muy familiarizadas con las nuevas tecnologías y por ello esta debe representar una interfaz simple y concisa, mateniendo siempre las funcionalidades básicas para la que se va a desarrollar.

2.1.2. Representación y gestión de la información

Por otro lado, se ha querido abordar el problema de la visualización de estos datos digitalizándolos y mostrándolos a través de una **aplicación web** que sea capaz de:

- Mostrar de manera sencilla la información de cada comunero con su respectivas medidas.
- Gestionar la llegada de nuevos comuneros y salida de los mismos, a fin de poder llevar una similitud con el software de facturación.
- Visualizar la información y los dispositivos de medición.
- Gestionar la información de los dispositivos de medición.
- Gestionar los canaleros de la comunidad.

Esta aplicación web servirá de apoyo al programa de facturación que utiliza ya la comunidad. Por ello, debe ser complementaría a la misma y no un sustitutiva. Al igual que ocurre con la aplicación móvil, debe ser sencilla de utilizar y donde el usuario pueda realizar todas las tareas de una manera simple, evitando todos aquellos problemas que se describieron en el capítulo anterior.

2.2. Impacto del COVID-19

Este trabajo ha sido desarrollado bajo unas circunstancias especiales ocasionadas por la pandemia mundial debido al Coronavirus. Este conjunto de circunstancias han hecho que el desarrollo de ciertas partes del proyecto se hayan visto afectadas.

- **Falta de Feedback por parte de la comunidad**: Los trabajadores de la comunidad de regantes no están muy familiarizados con las nuevas tecnologías. Por ello, la situación producida por el estado de alarma debido al COVID-19, ha difilcutado la comunicación con la Comunidad de Regantes sobre todo a la hora de poder mostrar los avances del trabajo y recibir un *feedback* directo por su parte.
- **Imposibilidad de pruebas en entorno real**: Por otro lado, la situación ha impedido que se pudieran realizar pruebas en un entorno real. En estas pruebas se hubiera testado la aplicación con datos propios de la comunidad, realizando alguna ruta simulando las que debería hacer el canalero dónde podríamos detectar posibles problemas de conexión y ver como se comportaría la aplicación.

Capítulo 3

Tecnologías utilizadas

En el capítulo anterior se planteo el problema que se afronta en este trabajo.

En este capítulo se describirán las tecnologías que se han utilizado para el desarrollo de ambas aplicaciones así como el por qué de su elección. Han sido agrupadas en los siguientes bloques:

1. Front-End
2. Back-End
3. Editores
4. Control de versiones

3.1. Front-End

Para desarrollar el **Front-End** se ha realizado un pequeño estudio de las principales tecnologías con el fin de poder decidir la mejor tecnología para tanto la aplicación web como la aplicaciom móvil.

Puesto que ambas aplicaciones son bastante diferentes y las tecnologías que se utilizan para desarrollar una u otra son tambien muy diferentes, en esta sección vamos a nombrar las principales tecnologías estudiadas para cada caso y la elección de cual se utilizará.

3.1.1. Aplicación Móvil

Puesto que parte del objetivo de este trabajo consiste en realizar apliaciones multiplataforma, la aplicación móvil debería poder usarse tanto en un dispositivo *Android* como en un dispositivo *iOS*. Además la aplicación solamente se encargará de la recogida de datos de los contadores y no necesitará ninguna funcionalidad específica, por lo cual fuera necesario desarrollarla en los entornos nativos de cada dispositivo.

A vista de lo descrito anteriormente, se ha decidido investigar sobre tecnologías multiplataforma de manera que se codifique de forma común y luego se pueda compilar para la plataforma que fuera necesaria. Para esta labor, existen diferentes variantes que actualmente se encuentran en pleno auge:

- **Flutter**: ”Flutter es un SDK creado por **Google** en 2015 cuyo fin reside en el desarrollo de interfaces de usuario para móviles Android o iOS. La primera versión de Flutter fue conocida como *Sky* y fue usada en un dispositivo Android” [7].

  El Framerwork está divido en varios componentes:

  - **Flutter Engine**: Esta escrito en C++ y proporciona el soporte necesario para el renderizado a bajo-nivel que uitliza *Google Sikia*.
  - **Foundation Library**: Está escrita en Dart y proporciona el conjunto de librerías base que podemos utilizar con Flutter para desarrollar nuetras apliaciones.
  - **Design-Specific Widgets**: Flutter cuenta con dos sistemas de diseño. Por un lado cuenta con el conocido Material Design de Google y por otro lado el sistema de Widgets Cupertino que intenta imitar la apariencia de Apple.

- **Ionic**: Al igual que ocurre con Flutter, Ionic es un SDK con el cual podemos desarrollar aplicaciones hibrídas tanto para Android como para iOS partiendo desde un mismo código. ”Fue creada por Max Lynch, Ben Sperry y Adam Bradley y publicada en 2013. En primer instancia el Framework fue desarrollado solamente para utiliazr Angular JS y Cordova como principales tecnologías que luego transcribía a código nativo” [8].
  Actualmente el lenguaje ha sido re-escrito y cuenta con soporte para las tecnologías de *Web Components* y los Frameworks de desarrollo web Angular, Vue y React. Ionic tiene una similitudes con el Framework *Electron* el cual nos permite lo mismo pero para crear aplicaciones de escritorio multiplataforma.

- **React Native**: ”React Native es una librería creada por Facebook y lanzada en el año 2013 para la creación de aplicaciones móviles multiplataforma. En su core utiliza la tecnología de React (librería para la creación de interfaces web) que luego transcribe a código nativo para generar las aplicaciones. La principal diferencia con React es que esta librería no trabaja con el *DOM* y el *Virutal DOM*”. [9] En cambio, lo que hace es correr en un proceso de segundo plano que interpreta el código de Javascript directamente en el terminal final y que se comunica con la plataforma nativa mediante un *Bridge* que el sistema nativo puede entender.

3.1.2. Aplicación Web

Para la aplicación web tambien se ha realiado un estudio de cuál podría ser la mejor tecnología posible para desarrollarla. A diferencia de las aplicaciones móviles en plataformas web existen muchos Frameworks y librerías para poder desarrollar aplicaciones web y cada día surgen muchas más.

Una cosa que debemos destacar entre todas ellas es que comparten el **Modelo de Componentes** cuyo principal objetivo es la creación de widgets con el fin de poder reutilizarlos a lo largo de todas nuestras páginas o aplicaciones web y de esta manera eliminar la duplicidad de código y agilizar mucho más el desarrollo. Este modelo ha sido adoptado ya por *W3C.*

Sin embargo, el papel principal actualmentel lo tienen tres tecnologías:

- **AngularJS**: ”AngularJS es un framework para aplicaciones web desarrollado en *TypeScript*, de código abierto, mantenido por Google, que se utiliza para crear y mantener las conocidas SPA (*Simple Page Aplication*). Su objetivo es aumentar las aplicaciones basadas en navegador con capacidad de Modelo Vista Controlador (MVC), en un esfuerzo para hacer que el desarrollo y las pruebas sean más fáciles. Fue lanzado en Marzo de 2015” [4].

- **VueJS**: ”Vue es un Framework de código abierto basado en el paradigma MVVM (*Model-View-ViewModel*) y Javascript para crear SPA. Fue creado por Evan You y mantenido por el mismo hasta que compañías como Netflix y Netguru entraron a darle soporte. Fue presentado en Febrero de 2014 y está diseñado exlusivamente para el renderizado de componentes y creación de interfacez web.” [5]

- **React**: ”React (también llamada React.js o ReactJS) es una librería Javascript de código abierto diseñada para crear interfaces de usuario con el objetivo de facilitar el desarrollo de las SPA. Es mantenido por Facebook y la comunidad de software libre. Han participado en el proyecto más de mil desarrolladores diferentes y cada día se suman más. Fue presentado en Mayo de 2013.” [6]

  La diferencia entre React y las dos tecnologías descritas anteriormente reside en que React es una librería y no un Framework como los demás. Esto hace que podamos utilizarlo en cualquier proyecto para crear nuestras vistas combinandolo con cualquier otro lenguaje de desarrollo puesto que finalmente lo que hará será transcribirse a un código *Vanilla* de HTML, CSS y Javascript.

3.1.3. Elección de la tecnonlogía

Finalmente se ha decicido utilizar tanto React Native para el Front de la aplicación móvil como React para la aplicación web. La principal razón de esta elección reside en que ambas comparten el mismo *core*, la misma estructura de desarrollo y donde único difieren es en las nomeclatura de ciertas sentencias. Esto nos ayudará a mantener la misma estrucutra de desarrollo tanto para la aplicación móvil como para la aplicaciónweb, haciendo que se acelere mucho la primera toma de decisiones sobre como se estructurarán los proyectos y como se abordarán los casos más básicos que ambas aplicaciones tendrán.

3.2. Back-End

Desde un principio se ha decido utilizar un mismo Back-End del cual consuman ambas aplicaciones a través de peticiones REST. En un comienzo se barajaron dos opciones sobre que tecnologías utilizar para desarrollarlo.

3.2.1. Node + Express

”Node.js es un entorno en tiempo de ejecución multiplataforma, de código abierto, para la capa del servidor (pero no limitándose a ello) basado en el lenguaje de programación JavaScript, asíncrono, con E/S de datos en una arquitectura orientada a eventos y basado

en el motor V8 de Google” [10]. Fue creado con el propósito de construir programas de altamente escalables como por ejemplo servidores web, API’s REST, etc.... Node.js fue presentado en Mayo de 2009 por Ryan Dahi, su creador.

Actualmente Noode.js es el entorno base para cualquier desarrollo web y cuenta con el apoyo de casi toda la comunidad.

Entre una de sus librerías más importantes podemos encontrar la de **Express.js** [11]. Esta librería nos permite crear de una manera muy rápida y sencilla una API REST con total funcionamiento utilizando el motor del entorno Node.js por debajo y el lenguaje Javascript.

3.2.2. Spring

”Spring es un framework para el desarrollo de aplicaciones y contenedor de inversión de control, de código abierto para la plataforma Java. Si bien las características fundamentales de Spring Framework pueden ser usadas en cualquier aplicación desarrollada en Java, existen variadas extensiones para la construcción de aplicaciones web. A pesar de que no impone ningún modelo de programación en particular, este framework se ha vuelto popular en la comunidad al ser considerado una alternativa, sustituto, e incluso un complemento al modelo EJB (*Enterprise JavaBean*).La primera versión fue escrita por Rod Johnson y publicada en Octubre del año 2002.” [12]

”**Spring Boot** es una infraestructura ligera que elimina la mayor parte del trabajo de configurar las aplicaciones basadas en Spring. El objetivo de Spring Boot es proporcionar un conjunto de herramientas para construir rápidamente aplicaciones de Spring que sean fáciles de configurar.” [13]

3.2.3. Elección de la tecnología

Con todos estos datos estudiados la tecnología que se ha escogido realizar el trabajo ha sido **Spring-Bot**. Aunque es verdad que con la primera variante hubiera sido mucho más rapido el desarrollo para este trabajo, en concreto, se ha preferido mantener un control sobre el tipado y estrucutras de datos que se utilizaban. Esto con la primera tecnología se puede hacer tambien, en caso de que incorporemos la librería de tipado para Javascript llamada Typescript [15], pero aun así esta librería proporciona un tipado mucho más débil que el de Java, pues solo se aplica a nivel de codificación. Mientras que en Java se aplica en todos los niveles del desarrollo de nuestra aplicación.

3.2.4. Firebase

Para desarrollar este proyecto también se han utilizado diferentes funcionalidades de las ofrecidas por Firebase, la plataforma de desarrollo en la nube de Google. Proporciona diferentes herramientas para gestionar la autenticación de usuarios, el almacenamiento de datos en la nube y diferentes estadísticas de uso, entre otras funcionalidades, sin necesidad de gestionar servidores. La API de Firebase está disponible para proyectos

Android, iOS, Web, C++ y Unity, además de disponer una SDK de administrador para servidores. [14] Las herramientas de Firebase que se han utilizado son:

- **Authentication**: Facilita la gestión de usuarios de forma simple y segura. Ofrece diferentes métodos de autenticación para los usuarios mediante e-mail y contraseña o inclusive gracias a la utilización de otros proveedores como Google, Facebook o Twitter.
- **Cloud Firestore**: Es una base de datos noSQL alojada en la nube. Permite mantener los datos sincronizados en tiempo real mediante listeners y ofrece soporte offline. De esta manera todas las aplicaciones que estén consumiendo datos de esta, pueden seguir en funcionamiento independientemente de la latencia de la conexión a internet.

  La principal diferencia entre Cloud Firestore y Realtime Database (las dos bases de datos que ofrece Firebase) es que con Cloud Firestore es posible realizar consultas más complejas, como pueden ser operaciones de filtrado u ordenación. Por otro lado, mediante el uso de Cloud Firestore se pueden leer datos de una sola vez, sin la necesidad de mantener un listener escuchando todo el tiempo. Se integra fácilmente con otros productos como Cloud Functions, sin necesidad de disponer servidores que actúen como intermediarios.

3.3. Editor

Para desarrollar el proyecto se ha decidido utilizar dos IDE de la suite de JetBrains:

3.3.1. Webstorm

Webstorm es uno de los IDE de la suite de JetBrains enfocado única y exclusivamente para desarrollo web. Este cuenta con numerosos plugins y snippets ya incorporados que permiten acelerar mucho el desarrollo y el manejo dentro del mismo.

Este se ha utilizado tanto para el desarrollo de la apliación web como de la aplicación móvil (ver Figura 3.1).

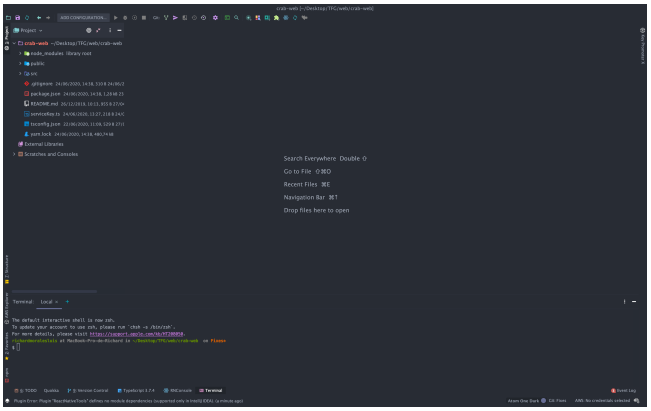


Figura 3.1: Webstorm

3.3.2. Intellij Idea

Intellij Idea al igual que Webstorm es otro de los IDE de la suite de JetBrains. La diferencia reside en que este IDE en concreto esta creado para la programación de aplicaciones android usando Java y/o Kotlin así como la creación de aplicaciones en Java. Cuenta con una gran cantidad de recursos e integraciones con gestores de dependencias de Java como Maven o Gradle que ayudan mucho a la hora del crear aplicaciones puesto que los gestionan automáticamente (ver Figura 3.2).

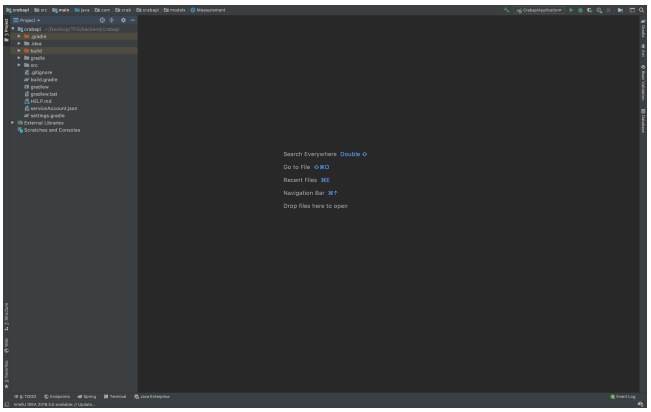


Figura 3.2: Intelijj Idea

3.4. Control de Versiones

El sistema de control de versiones elegido para registrar los cambios tanto en la aplicación móvil como en la aplicación web y el back-end es Git.

Además para alojar el código se ha utilizado **Github** (La popular plataforma de alojamiento del código de proyectos en remoto que utiliza Git.). En el cual se han creado 3 repositorios diferentes uno para el back-end y otro para cada una de las aplicaciones.

Capítulo 4

Desarrollo del trabajo

En los capítulos anteriores se han descrito detenidamente cual es las problemáticas que se abordarán en este trabajo así como las tecnologías que se utilizarán para ello.

En este capítulo, se expondrá la manera en la cual se ha desarrollado el trabajo y cuál ha sido el resultado final de ambas aplicaciones.

4.1. Desarrollo de la aplicación móvil

Como se comentaba en el primer capítulo la aplicación móvil se encargará de facilitar la tarea de recogida de las medidas mensuales de cada comuneros. Para ello la aplicación debía cumplir unos requisitos:

- Localizar los contadores.
- Permitir tomar medidas de los contadores y las válvulas.
- Permitir localizar nuevos contadores.
- Mostrar de manera sencilla el estado (medido, error y no medido) de los contadores ó válvulas.

Antes de plantearse todo el desarrollo, se analizaron los requisitos necesarios que debía cumplir dicha aplicación y se creó un primer diagrama de flujo en el cual se intentó plasmar las funcionalidad básica de la aplicación, que fue validado por las comunidad de rregantes antes del paso a la implementación (ver Figura 4.1).

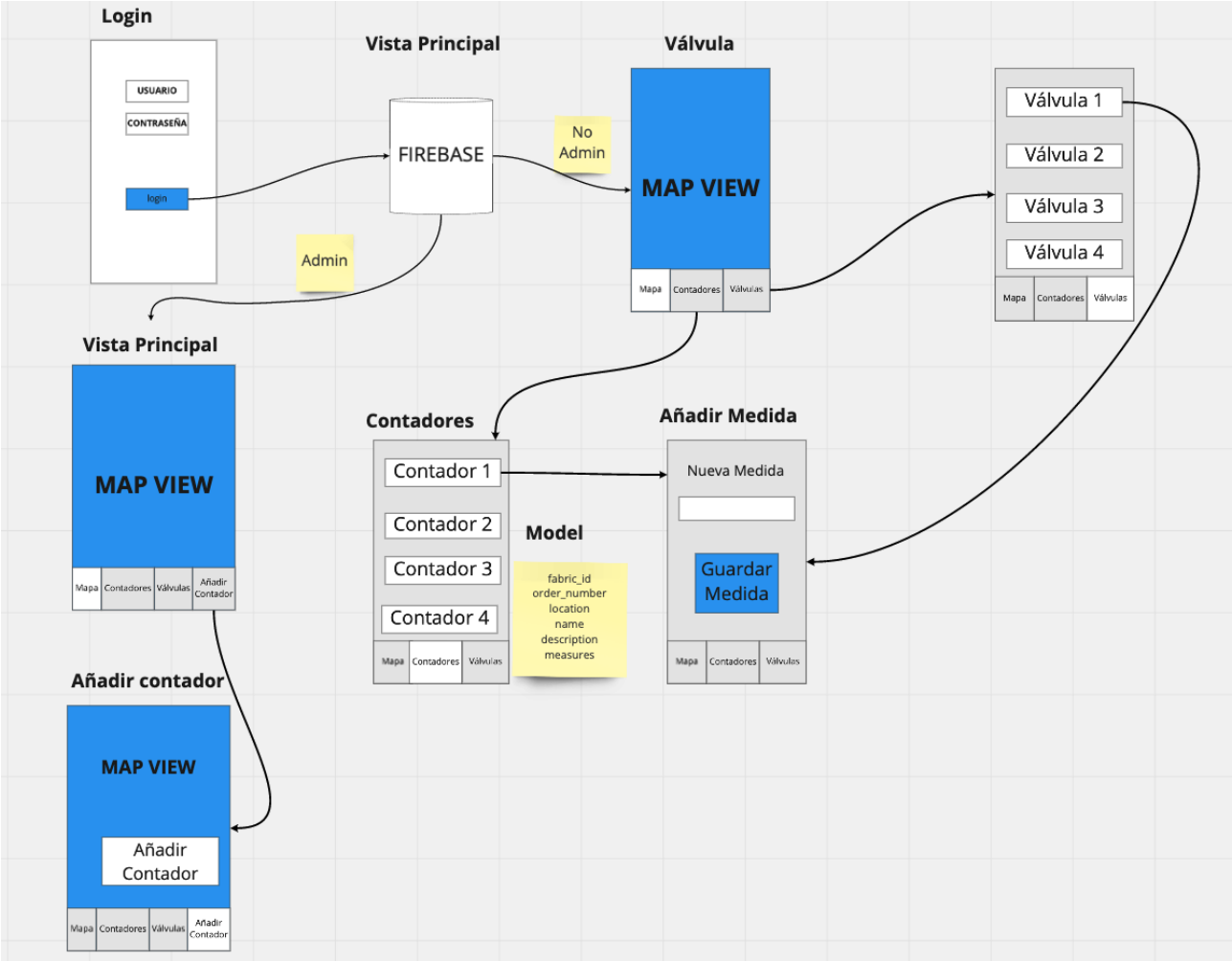


Figura 4.1: Diagrama de Flujo Móvil

Cuando arrancamos la aplicación tenemos que iniciar sesión en ella con el correo y contraseña que se haya falcilitado por parte de la persona que ha creado el perfil nuevo de canalero. Es importante destacar que existen dos perfiles que pueden usar la aplicación un perfil **administrador** y un perfil **no administrador**. Esto es así porque no se permite a los usuarios que no son administradores añadir localizaciones de los nuevos dispositivos (ver Figura 4.2).

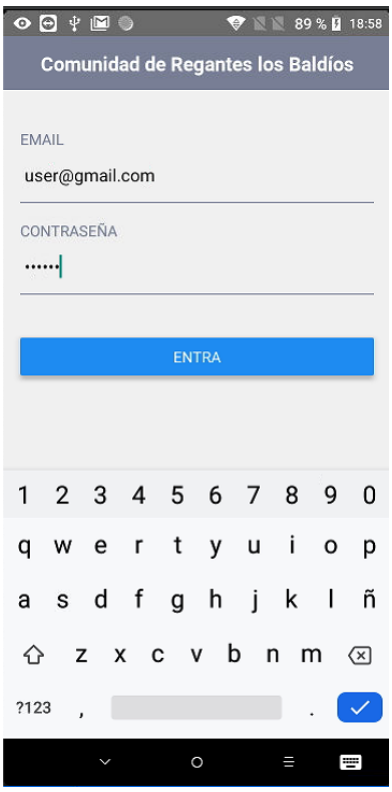


Figura 4.2: Login

Una vez entramos en la aplicación se muestra el mapa de la zona en la cual se encuntre el usuario y el conjunto de contadores que se hay en la misma. En la parte de abajo del a pantalla se puede ver las diferentes secciones que tiene la aplicación.

4.1.1. Mapa

En esta vista el usuario podrá ver los dispositivos que tiene a su alrededor localizados así como el estado en el que se encuentra cada uno de ellos. Para ello se ha utilizado la librería de *react-native-maps* [16] la cual ofrece un componente de mapa que puede ser interpretado tanto por iOS como por Android.

Aparte del mapa, esta librería proporciona un conjunto de componentes como pueden ser los *Markers* que permiten representar puntos de interés en el mapa y tambien *Callouts* que se utilizan para dar información extra a de los *Markers*.

La manera de implementarlo es muy sencilla, simplemente deberemos instalar la librería en nuestro proyecto usando algún de gestor de paquetes para javascript como puede ser *Yarn* ó *NPM*. Al usar la última versión de React Native (*0.62*) al compilar las aplicaciones se realizarán los conocidos como *auto-links*, que permiten convertir las librerías de javascript importadas en código nativo. Una vez instalado tendremos que realizar dos pasos de configuración para poder utilizarla:

- **Configuración para iOS**: Para el caso de iOS debremos instalar las dependencias desde cocoapods (uno de los gestores de dependencias más conocidos para el desarrollo de iOS). Para ello simplemente deberemos ejecutar (ver Figura 4.3):

Figura 4.3: Install Pods

Una vez instalado los pods, el último paso, sería añadir la *APIKEY* de Google para poder usar el mapa de Google en nuestra aplicación. Para ello, bastaría con editar nuestro código del AppDelegate.m escrito en Objetive C y añadir la siguiente linea (ver Figura 4.4):

Figura 4.4: Google API KEY iOS

- **Configuración Android**: En el caso de android simplemente habría que añadir la *APIKEY* de google en el fichero Manifest.xml (ver Figura 4.5).

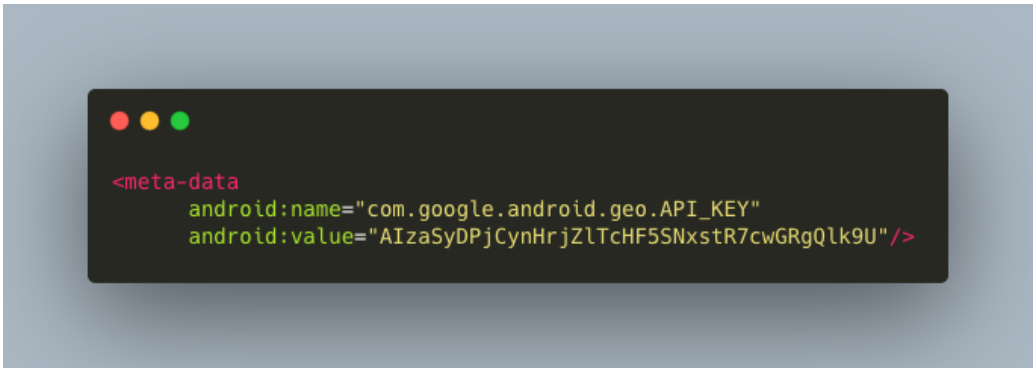


Figura 4.5: Google API KEY Android

Finalmente, simplemente habría que importar en la vista que se usará y configurarla con las propiedades (*props* de React) que mas convengan. En este caso la configuración del mapa y de sus markers quedaría de esta forma (ver Figura 4.6):

```
import MapView, { Marker } from 'react-native-maps';
<MapView
        key={mapKey}
        style={style.map}
        region={region}
        minZoomLevel={19}
        maxZoomLevel={20}
        mapType="satellite"
        showsPointsOfInterest={false}
        showsTraffic={false}
        showsUserLocation={true}
        followsUserLocation={true}
        cacheEnabled={false}>
        <Marker coordinate={{ latitude: 28.470076, longitude: -16.329254 }} title="Oficina Central
Comunidad de Regadíos" pinColor={BLUE}/>
```

Figura 4.6: Componente Mapa

El mapa se encarga de mostrar todos los contadores que pertenecen a los socios de la comunidad. Entre ellos se pueden diferenciar tres colores que represnetan el estado en el que se encuentran los dispositivos (leído, pendiente, leído con error) (ver Figura 4.7):

- Amarillo: Estado pendiente del contador o la válvula. Esto es, el dispositivo aún no ha sido medido.
- Rojo: Estado error del contador o válvula. Se activa cuando el contador ha sido medido con una anotación lo cual implica un error.
- Verde: Estado de éxito. El dispositivo ha sido medido con éxito.

Figura 4.7: Mapa Principal

4.1.2. Contadores

En esta pantalla se puede ver la lista de contadores ordenados según el criterio que se describe a continuación.

Primero se mostrarán aquellos dispositivos que no hayan sido medidos durante ese mes y posteriormente los que ya han sido medidos. A su vez los contadores deben estar ordenados segun un número de orden que se indica cuando se crea el mismo.

Cada contador de la lista podrá ser localizado en el mapa (el mapa se centrará en el contador seleccionado) ó medir donde se podrá añadir una nueva medida o anotación. A esta vista se puede llegar tambien desde el propio mapa. Si se hace click encima de algún dispositivo el mapa se centra encima de este y nos muestra información acerca del dispositivo seleccionado. Si se pulsa encima de dicha información, podemos medir dicho dispositivo (ver Figura 4.8).

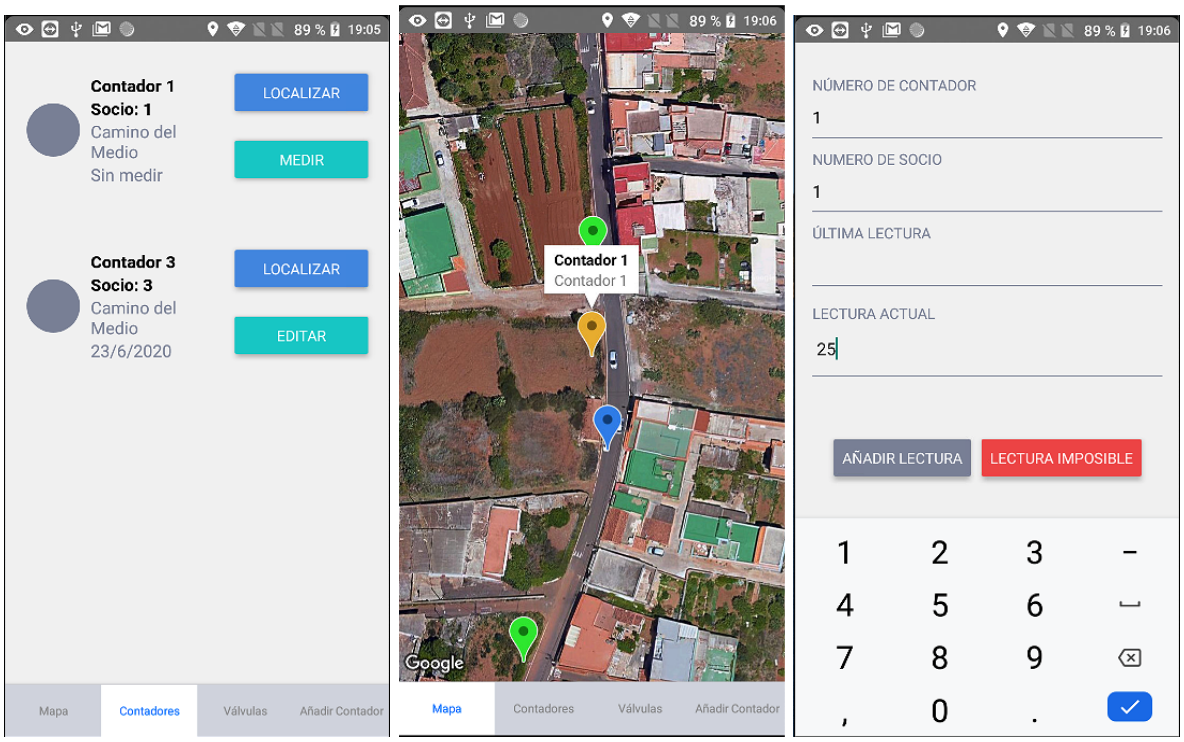


Figura 4.8: Medir un contador

4.1.3. Válvulas

Al igual que ocurre con la pantalla de contadores se muestra una lista de las válvulas ordenadas las cuales podremos medir o localizar (ver Figura 4.9).

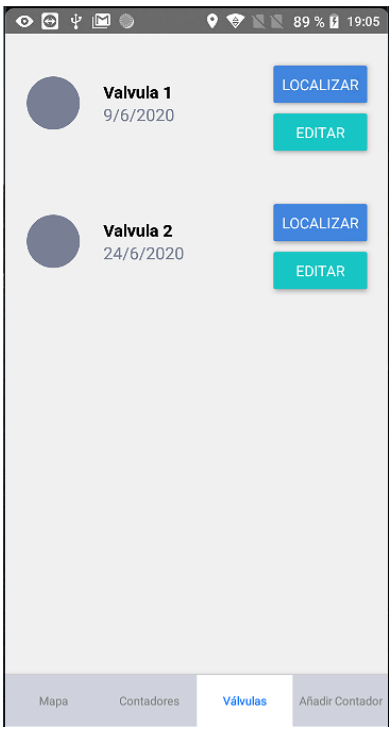


Figura 4.9: Listado de Válvulas

4.1.4. Añadir Contador

Esta vista solamente esta disponible para aquellos usuarios que entren con el rol de administrador, que se les podrá asignar o quitar desde la aplicación web. En ella, se mostrará una lista de aquellos dispositivos sin localizar (han sido creados pero no han silo colocados en una posición en el mapa) y permitirá colocarlos en el punto donde se quiera.

Una vez añadidos aperecerán en el mapa principal y en a lista de contadores ó válvulas, dependiendo del dispositivo que se añada, y podremos proceder a medirlo (ver Figura 4.10).

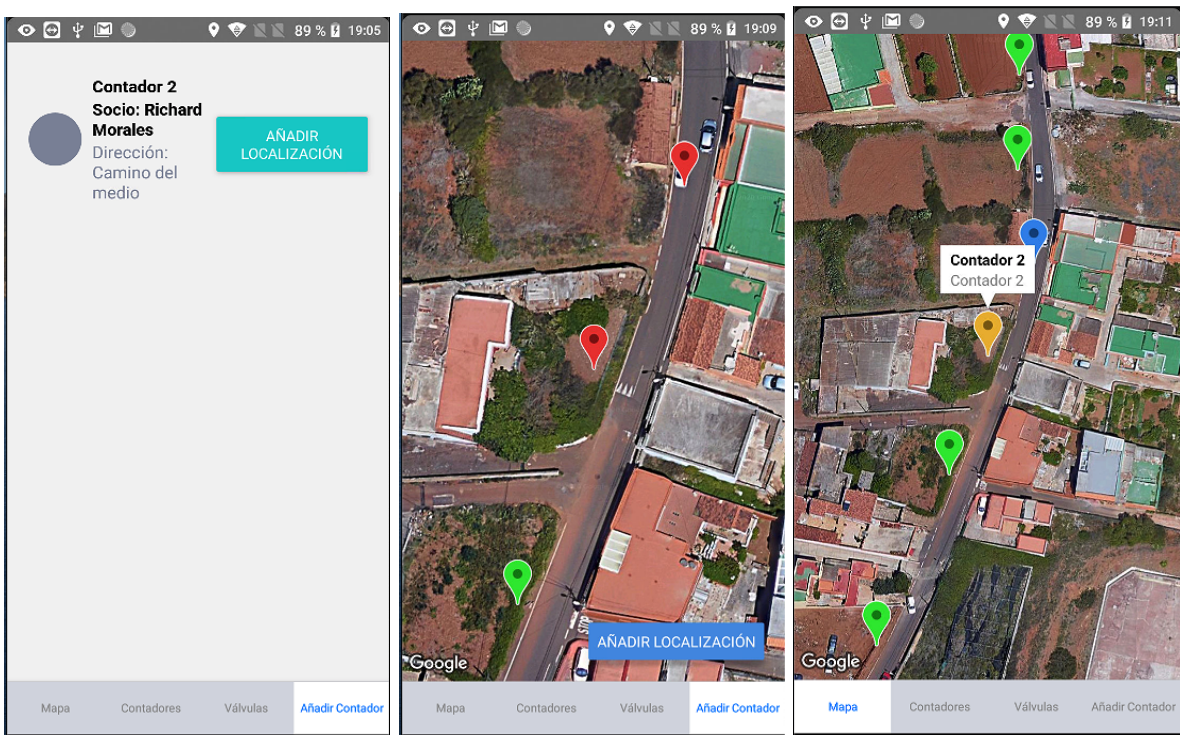


Figura 4.10: Añadir nuevo dispositivo

4.2. Desarrollo de la aplicación web

Al igual que en la aplicación móvil y como comentabamos en el capítulo uno, la aplicación web se creará para ayudar al usuario a revisar los datos que se registren durante el periodo de medición y facilitar la tarea de facturación que realiza la administración de la comunidad.

Por ello esta debería ser capaz de:

- Mostrar de manera sencilla la información de cada comunero con su respectivas medidas.
- Visualizar la información y de los dispositivos de medición.
- Gestionar los comuneros.
- Gestionar la información de los dispositivos de medición.
- Gestionar los canaleros de la comunidad.

Todo esto debería hacer desde una interfaz sencilla haciendo que el personal de administración de la comunidad no se equivoque cuando quiera realizar una acción concreta. Al igual que con la aplicación móvil en una primera instancia se desarrollo un diagrama de flujo que posteriormente fue validado por la comunidad antes de pasar a la fase de desarrollo (ver Figura 4.11).

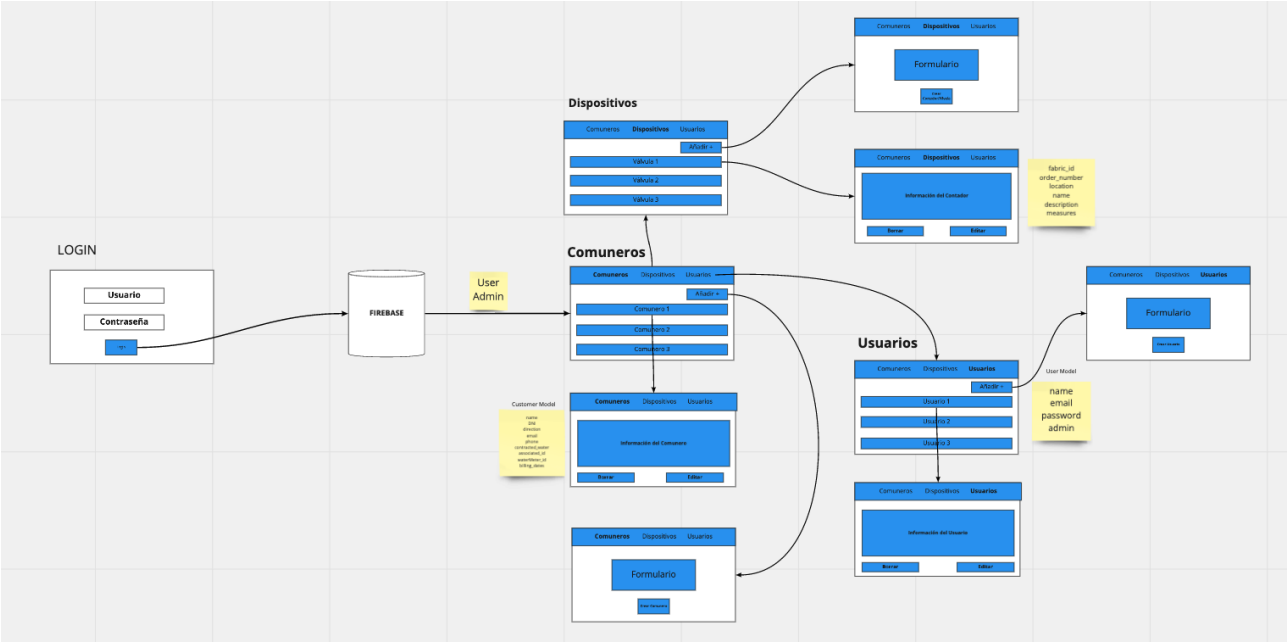


Figura 4.11: Diagrama de Flujo Web

Cuando se entra en la web, al igual que en la aplicación para tomar medidas, se muestra un login (Figura 4.12) a través del cual solo se puede entrar si se tiene el rol administrador. Esto es porque la aplicación maneja datos de los socios de la comunidad y algunos sensibles como pueden ser DNI, correo electrónicos, lugar de residencia o número de teléfono, entre otros. Por ello se ha decidido que solamente el personal de administración sea quien pueden entrar aunque la web sea visible para todo el mundo.

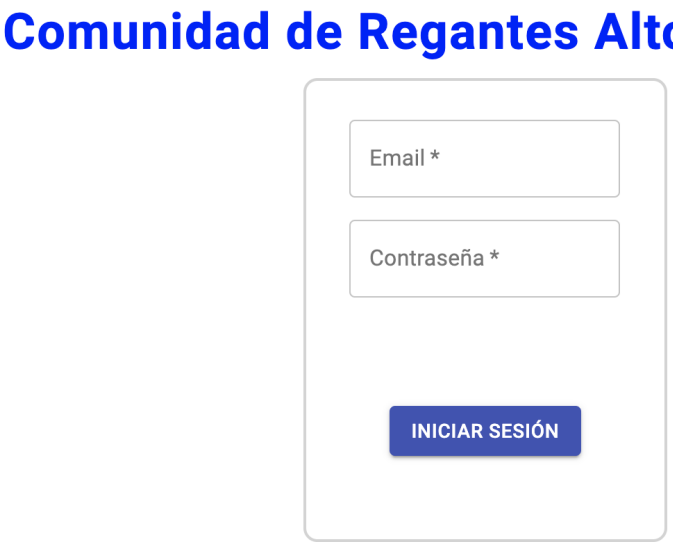


Figura 4.12: Login Web

Una vez realizado el login, se puede observar como la web tiene tres pestañas, cada una de ellas se centra únicamente en una sola sección que queramos manejar.

4.2.1. Comuneros

Es la pestaña principal y en ella se visualiza una tabla con toda la información que necesita el personal administrativo para poder llevar a cabo la emición de facturas mensual de cada comunero junto al software de facturació (ver Figura 4.13).

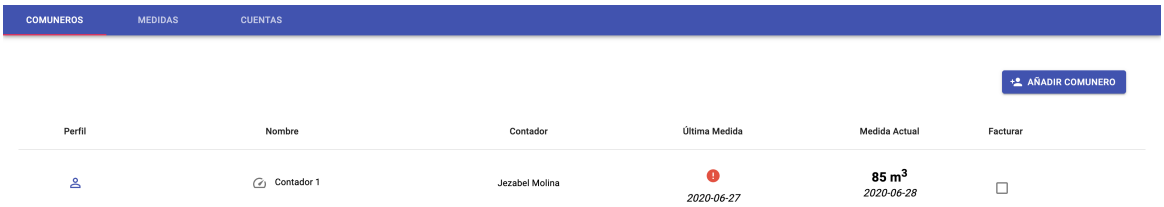


Figura 4.13: Vista de comuneros

A través de esta tabla el usuario puede checkear cuando una factura ya ha sido creada y automátiacmente esta fila pasará al final de la tabla colocandose en primero lugar aquellos comuneros que todavía no se les haya emitido una factura. Esto favorece la agilidad a la hora de saber con qué comunero se está en cada momento y se evita la pérdida de referencia que se nombraba como uno de los problemas en el primer capítulo.

Por otro lado, en la tabla se puede observar los datos de la medida del mes actual así como la del mes anterior, de manea que el administrador pueda verificar que no hay errores de un mes a otro. De la misma forma si existe un error a la hora de medir, se le notificará al usuario mediante un simbolo de error que al pulsar mostrará la anotación que haya hecho el canalero.

Además de visualizar la información, se pueden realizar diferentes acciones:

Crear nuevo comunero

En la parte derecha de la pantalla tenemos un botón, con el se permite crear un nuevo comunero. Al apretar sobre él se abrirá un formulario, el cual hay que rellenar con la información del comunero. Por una parte, se encuentra la información principal del comunero y en segundo lugar, la información básica sobre el contrato que maneja la aplicación. En este apartado el administrador insertará la cantidad de agua contratada por el comunero, el número de socio que se ha asignado y finalmente elegirá el contador que le pertenece. Si el contador del nuevo comunero no esta creado, se puede dejar en blanco y asignarlo más adelante editando la información del mismo (ver Figura 4.14).

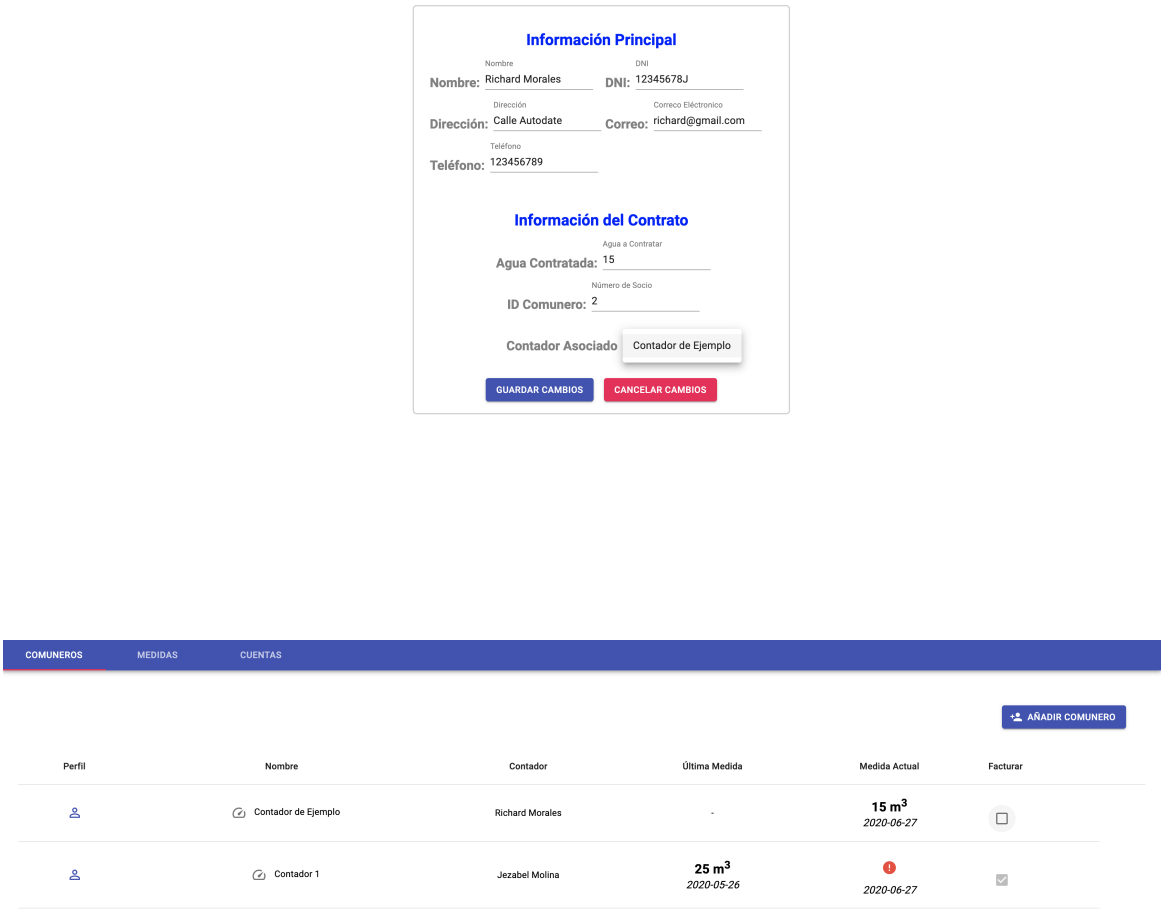


Figura 4.14: Añadir nuevo comunero

Ver/Editar información de un comunero

A través del icono de perfil de la parte izquierda de la tabla, se puede acceder a la información del usuario con el fin de, visualizarla, editarla o borrar el comunero en cuestión. En esta vista se puede reasignar un contador al comunero seleccionado y automáticamente el contador que tenia seleccionado pasará a quedar libre, sin perder su histórico de medidas. En caso de que se borre al comunero se comporta de la misma manera (ver Figuras 4.15 y 4.16).

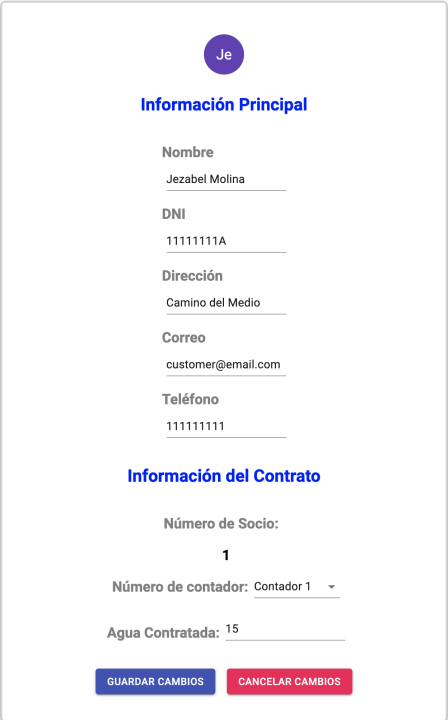


Figura 4.15: Información del comunero

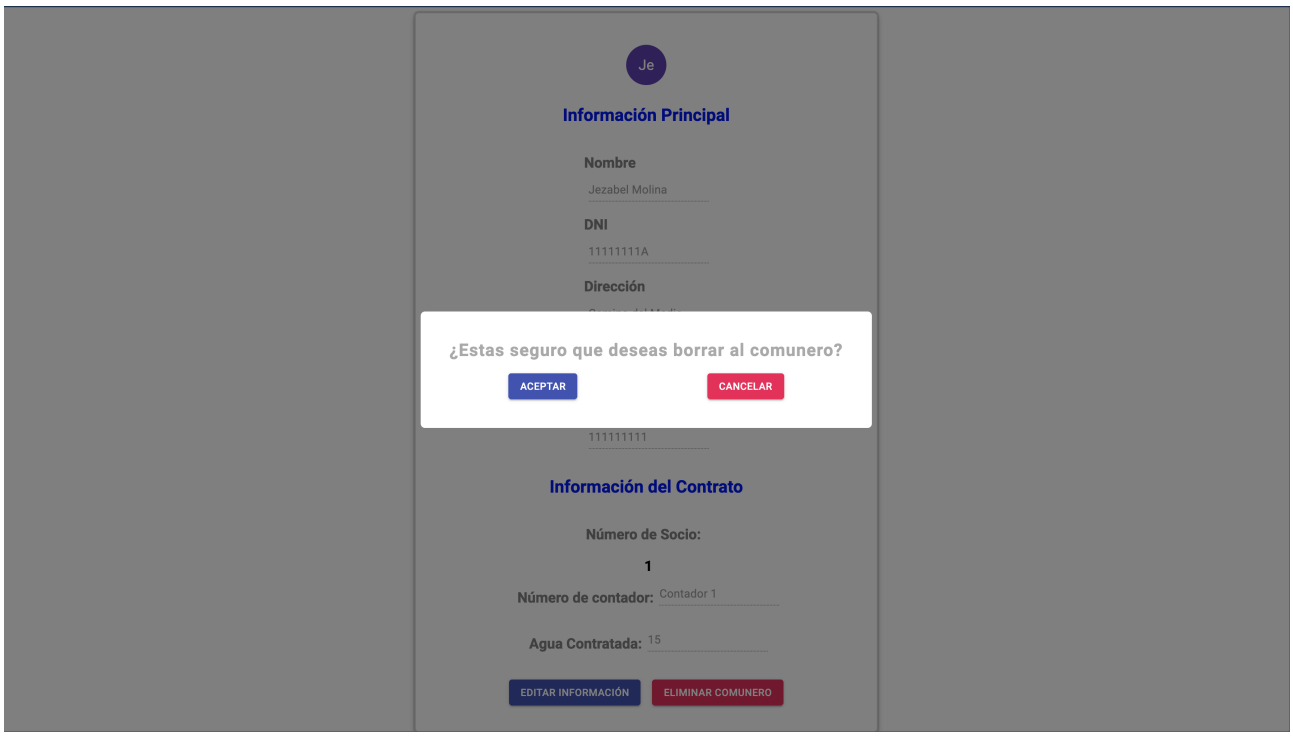


Figura 4.16: Borrar un comunero

4.2.2. Medidas

Válvulas y Contadores

En esta vista se despliega una lista con todas las válvulas que actualmente utiliza la comunidad. Cada válvula muestra la información del nombre y la medida que ha

registrado el canalero desde la propia aplicación. Una peculiaridad de estas es que cuando un contador asociado a la misma ha sido medido con un error, esta se marcará en color rojo a fin de notificar al usuario que la medida de alguno de sus contadores tiene un error. Al pulsar sobre una de ellas se comporta como un desplegable que abre una tabla con todos los contadores que tiene asociado. En dicha tabla se muestra la información de la última medida registrada de cada contador. Tambien se puede desplegar un histórico con las medidas de los últimos 12 meses (ver Figura 4.17).

Figura 4.17: Vista de medidas desplegada

En esta tabla se puede visualizar todas las medidas del contador hasta el momento así como borrar un contador. Cuando se borra uno de ellos automáticamente es desasignado del comunero que lo tenía (ver Figura 4.18).

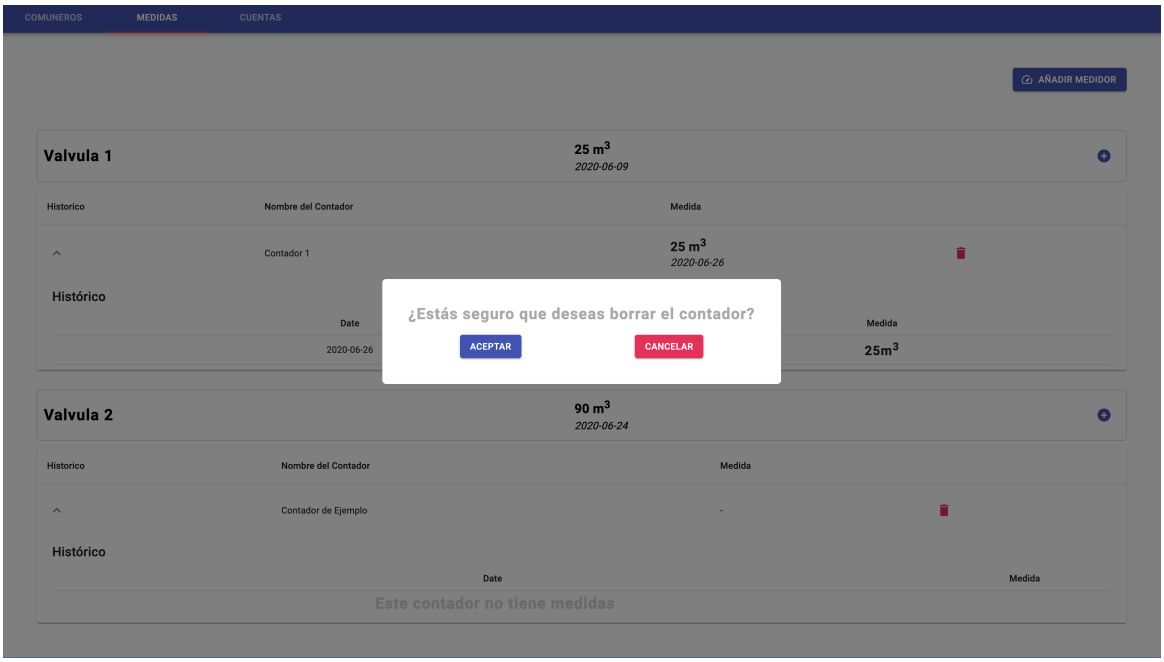


Figura 4.18: Borrar un contador

Crear nuevo dispositivo

A la derecha de la página existe un botón que permite crear un dispositivo nuevo, se entiende dispositivo como una válvula o un contador. En el formulario podremos elegir si lo que se va a añadir es una válvula o un contador, en cada caso el formulario se adapta a los datos que sean necesarios para cada dispositivo (ver Figuras 4.19 y 4.20).

Figura 4.19: Crear nuevo contador

Figura 4.20: Crear nueva válvula

Añadir un contador a una válvula

Cuando se crea un contador nuevo, este último debe ser asignado a una válvula pues debe quedar registrado en el total de metros cúbicos que la válvula registra cada mes. Por ello en cada válvula se ha creado un botón de añadir con el cual se puede referenciar a un contador dentro de una válvula (ver Figuras 4.21 y 4.22).

Figura 4.21: Añadir contador a una válvula

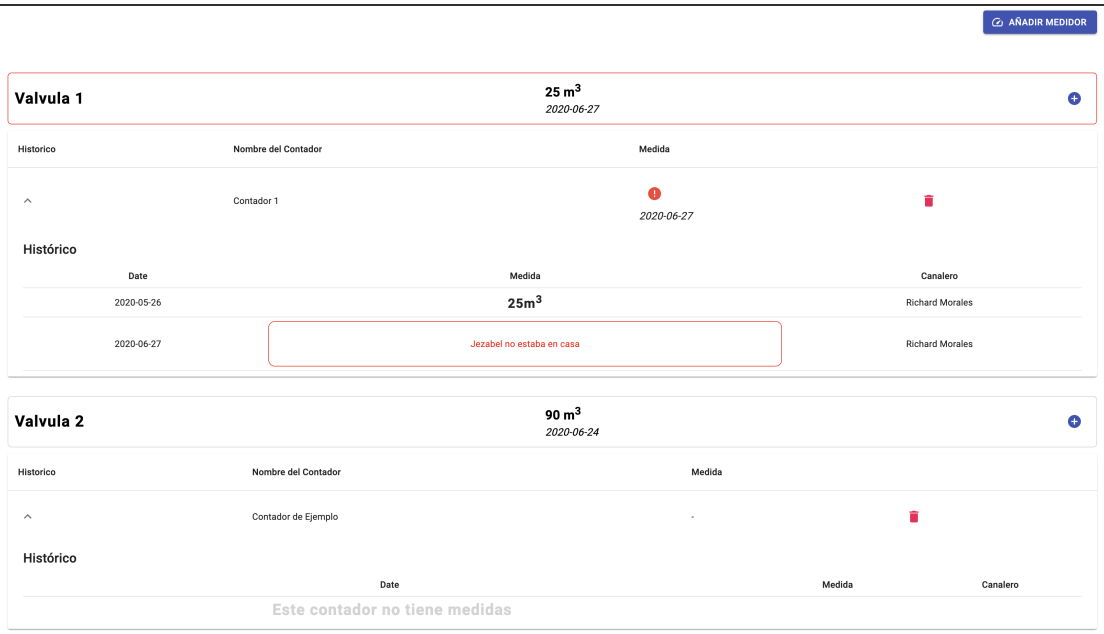


Figura 4.22: Contador nuevo añadido

4.2.3. Usuarios

En esta última pestaña se muestra una tabla con todo el personal de la comunidad (canaleros y personal de administración) con su información y el rol que tienen en cada momento (ver Figura 4.23)(administrador o no administrador).

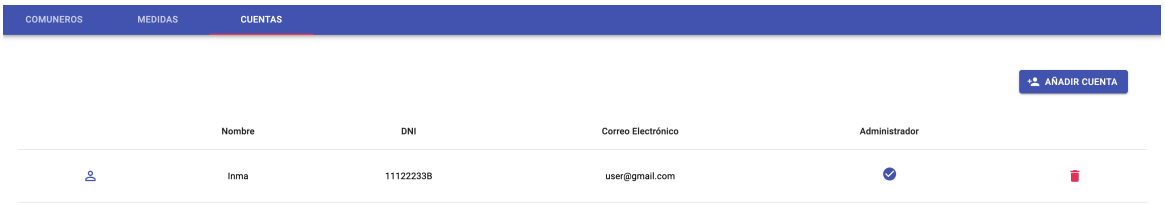


Figura 4.23: Vista de cuentas

A través de esta vista podemos realizar diferentes acciones:

Crear una cuenta nueva

Si se selecciona el apartado de **Crear Nueva Cuenta** se abre un formulario en el cual se introduce la información del nuevo usuario, así como su correo y contraseña con el cual podrá acceder a la aplicación móvil y finalmente se seleccionará el rol que tendrá (ver Figura 4.24).

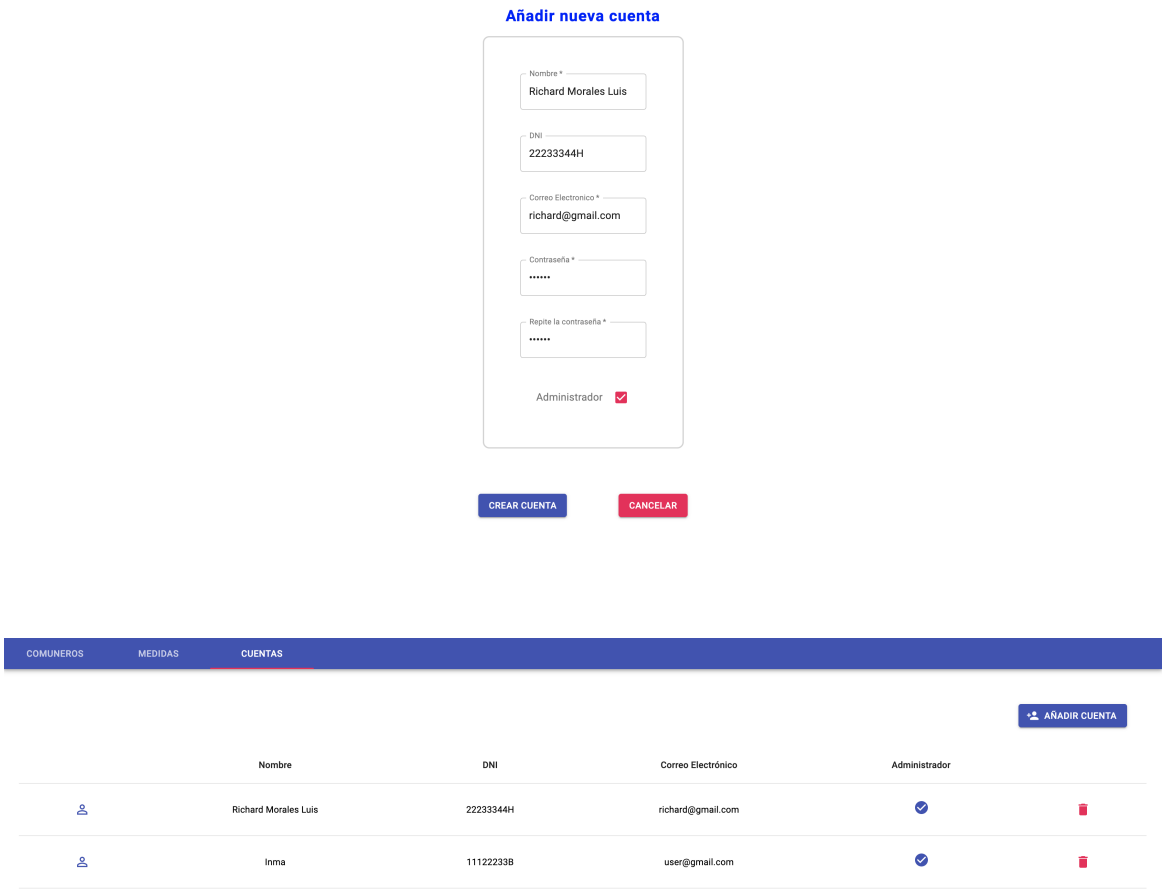


Figura 4.24: Añadir nueva cuenta

Editar un usuario

Al principio de la tabla se muestra un icono con el cual se abre una vista donde se puede visualizar toda la información del usuario seleccionado así como editarla para actualizarla (ver Figura 4.25).

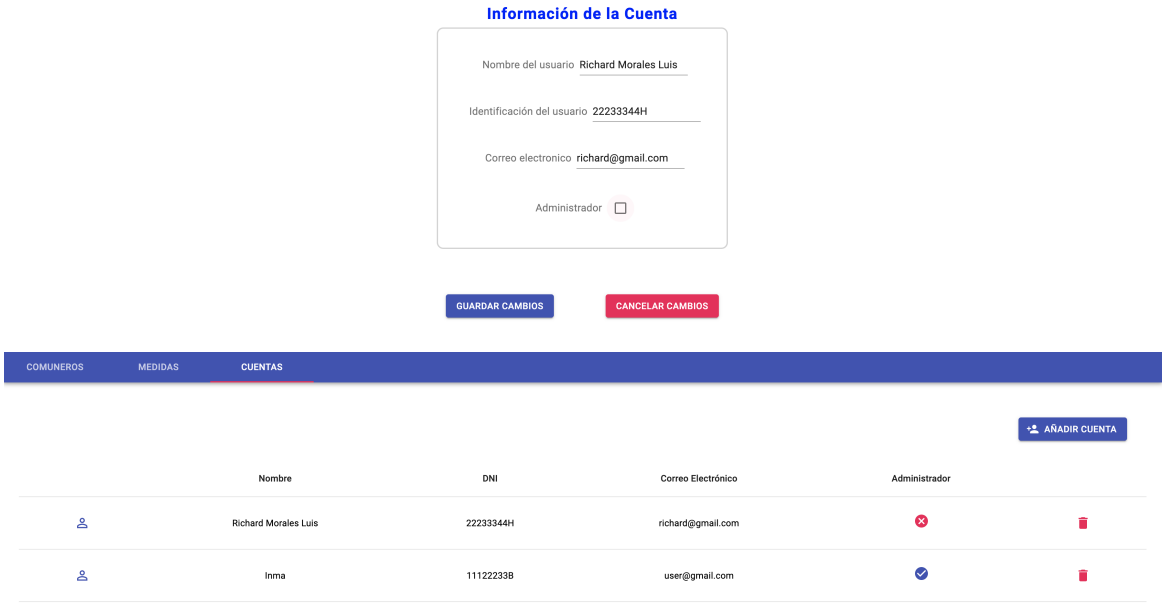


Figura 4.25: Editar información de una cuenta

Borrar un usuario

Como se ve al final de la tabla se encuentra un botón con el cual se puede borrar un ususario. Este automáticamente será borrado como usuario del sistema y no podrá acceder a la aplicación para realizar ninguna tarea. Antes de borrar al usuario se mostrará un mensaje para confirmar la acción (ver Figura 4.26).

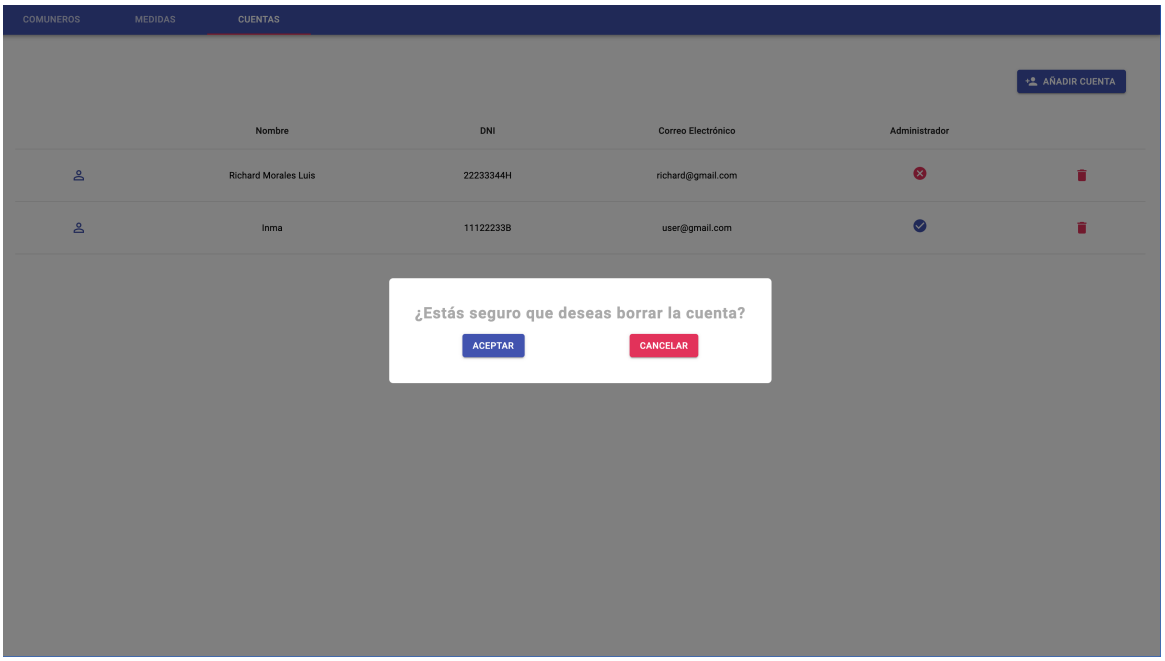


Figura 4.26: Borrar cuenta

4.3. Desarrollo del Back-End

Como se comentó en el capítulo dos el backend de la aplicación se ha hecho utilizando el Framework Spring-Boot basado en Java y los servicios de *Cloud Firestone* de almacenamiento para la base de datos y *Authentication* para el registro de usuarios que provee Firebase.

Conexión con Firebase

Desde un principio se ha intentado establecer un modelo de datos sencillo y que no se convirtiera en un problema a la hora de mantenerlo o escalarlo en un futuro. Se debe tener en cuenta que **Cloud Firestore** es una base de datos no relacional como ya nombramos basada en *Colecciones* y *Documentos*. Para poder utilizar los servicios de Firebase el back-end debe establecer una conexi ón con el Framework. La manera de configurarlo es bastante sencilla. En primer lugar deberemos ir a la consola del proyecto creado en Firebase y descargar desde ajustes el fichero de configuración ***serviceAccount.json*** el cual contiene toda la información y parámetros que necesita el back-end para establecer dicha conexión.

Una vez tenemos este fichero, el siguiente paso es instalar (en este caso usando Gradle como gestor de dependencias) la SDK de Firebase para Java. Para ello bastaría con añadir la siguiente línea en el fichero **build.gradle**:

compile 'com.google.firebase:firebase-admin:6.11.0'

Una vez se ha importado la dependencia de Firebase el siguiente paso es crear una clase que permita realizar la conexión entre la Api y Firebase una vez se inicie el controlador de Spring-Bot (ver Figura 4.27).

```java
package com.crab.crabapi.service;

import com.google.auth.oauth2.GoogleCredentials;
import com.google.firebase.FirebaseApp;
import com.google.firebase.FirebaseOptions;
import org.springframework.stereotype.Service;

import javax.annotation.PostConstruct;
import java.io.FileInputStream;
import java.io.IOException;

@Service
public class FirebaseInitialize {

    @PostConstruct
    public void initialize() throws IOException {
        FileInputStream serviceAccount = new FileInputStream("./serviceAccount.json");

        FirebaseOptions options = new FirebaseOptions.Builder()
                .setCredentials(GoogleCredentials.fromStream(serviceAccount))
                .setDatabaseUrl("https://crab-api.firebaseio.com")
                .build();

        FirebaseApp.initializeApp(options);
    }
}
```

Figura 4.27: Configuración inicial Firebase

La función initialize se ejecutará justo después de que el controlador de Spring-Bot se inicie. Esta función tiene como objetivo construir la conexión entre Firebase y Spring-Bot. A partir de este punto se puede utiliar el paquete Firestore y FireStoreClient para poder establecer la conexión con la base de datos que indicamos en el *FirebaseInitialize* y que permiten conexión con el servicio FireStore con el fin de poder acceder a los datos que este tiene almacenados (ver Figura 4.28).

```java
import com.google.cloud.firestore.*;
import com.google.firebase.cloud.FirestoreClient;

 Firestore db = FirestoreClient.getFirestore();
```

Figura 4.28: Instacia de Firestore

Modelo de datos

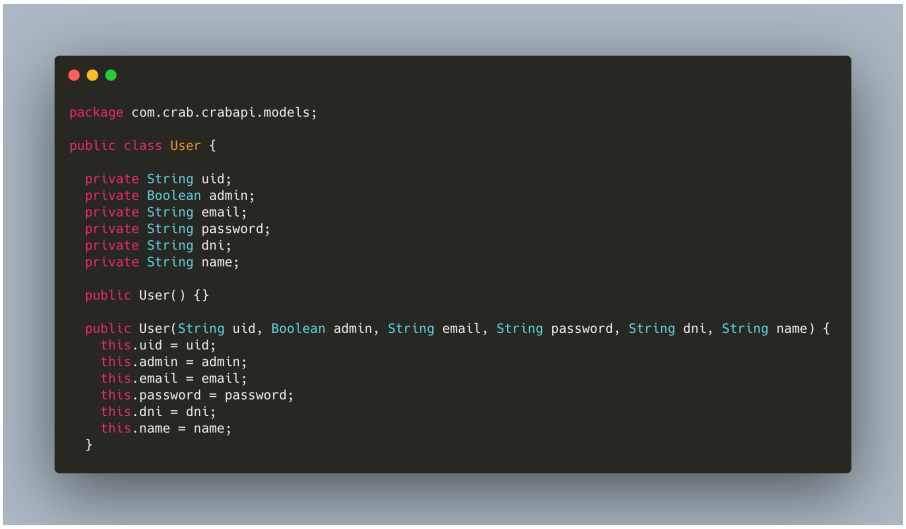
FireStore es una base de datos no relacional que se organiza usando Colecciones y Documentos como ya se nombró en el capitulo 2. En este caso particular se han creado 4 colecciones diferentes que se relacionan entre sí mediante unos identificadores que el back-end controla. Cada una de ellas utiliza un modelo específico para cada una así como un controlador que actúa a modo de CRUD para cada colección.

- **Customers**: En esta colección se registra los datos de los comuneros y se establece la relación con los contadores de cada uno de ellos. Esto se hace a través de un identificador único que se obtiene al crear cada contador nuevo. Cada comunero se identifica unequívocamente por su DNI (ver Figura 4.29).

```java
package com.crab.crabapi.models;

import java.util.ArrayList;

public class Customer {

  private String name;
  private Integer phone;
  private Integer contracted_water;
  private String email;
  private String dni;
  private String direction;
  private Integer associated_id;
  private String waterMeter_id;
  private ArrayList<CustomDate> billing_dates;

  public Customer() {
  }
```

Figura 4.29: Modelo de Comueros

- **Users**: La colección de usuarios trabaja de la mano con el servicio de *Authentication* de Firebase. Esta se encarga de resgistrar todos los usuarios dados de alta a través de la web. Al iniciar sesión el servicio de Firebase se comunica con dicha colección para devolver al usuario concreto que crea a través de un ID único que el mismo crea (ver Figura 4.30).



```java
package com.crab.crabapi.models;

public class User {

  private String uid;
  private Boolean admin;
  private String email;
  private String password;
  private String dni;
  private String name;

  public User() {}

  public User(String uid, Boolean admin, String email, String password, String dni, String name) {
    this.uid = uid;
    this.admin = admin;
    this.email = email;
    this.password = password;
    this.dni = dni;
    this.name = name;
  }
```

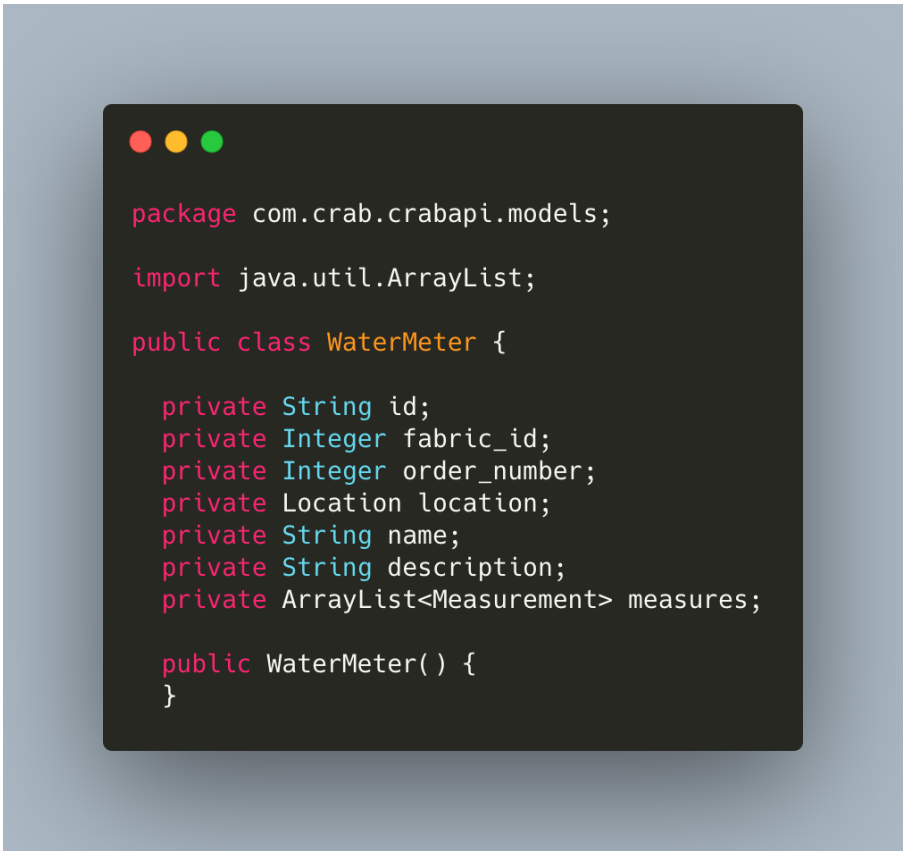
Figura 4.30: Modelo de Usuarios

- **Valves**: Guarda la información de las válvulas así como registra la relación de los contadores que pertenecen a cada una y guardar la localización de cada una de ellas (ver Figura 4.31).

```java
package com.crab.crabapi.models;

import java.util.ArrayList;

public class Valve {

  private String id;
  private String name;
  private ArrayList<String> waterMeters_associated;
  private Location location;
  private ArrayList<Measurement> measures;

  public Valve() {
  }
```

Figura 4.31: Modelo de Válvulas

- **WaterMeters**: Almacena la información de los contadores. En uno de sus campos contiene una lista con todas las medidas de dicho contador. Además registra la información de donde está localizado con una latitud y longitud específicas que se recogen desde la aplicación móvil al igual que ocurre con las válvulas (ver Figura 4.32).



```java
package com.crab.crabapi.models;

import java.util.ArrayList;

public class WaterMeter {

  private String id;
  private Integer fabric_id;
  private Integer order_number;
  private Location location;
  private String name;
  private String description;
  private ArrayList<Measurement> measures;

  public WaterMeter() {
  }
```

Figura 4.32: Modelo de Contadores

Capítulo 5

Presupuesto

En este capítulo se desglosa brevemente un posible presupuesto para el proyecto.

5.1. Coste Humano

Descripción	Horas	Preacio/Hora(€/h)	Total (€)
Planificación del Proyecto	10	6	60
Análisis de requisitos	8	6	48
Estudio y análisis de la tecnología	6	6	36
Creación de prototipo	20	15	300
Desarrollo de la aplicación web	40	15	600
Desarrollo de la aplicación móvil	80	15	1200
Desarrollo del Back-End	100	15	1500
Desarrollo de la memoria	36	10	360
Total	300	-	4014

Tabla 5.1: Presupuesto Coste Humano

5.2. Coste de Herramientas e Infraestructura

Descripción	Coste(€)
Licencia Apple Developer	89
Licencia Android Developer	22
Webstorm	129
Intelij Idea	499
Despliegue Servidor AWS	365
Servicio de Firebase	-
Total	1104

Tabla 5.2: Presupuesto Infraestructura y Herramientas

Capítulo 6

Conclusiones y líneas futuras

Desde un principio, el objetivo principal de este trabajo, ha sido facilitar la experiencia laboral de los trabajadores de la Comunidad de Regantes Alto los Baldíos. Con la creación de estas dos aplicaciones, este objetivo se ha cumplido y se podrá ahorrar mucho más tiempo a la hora de realizar la facturación de los comuneros. Esto se traduce en que se podrá dedicar mas tiempo a otras labores, como mejorar el servicio de riego de la comunidad, atraer más comuneros, expandirse hacia otros municipios, etc...

Al tratarse de un proyecto que tendrá una implicación real, ha contado con bastante colaboración por parte de la comunidad así como, de las profesoras las cuales han dado ideas que han facilitado y mejorado mucho la experiencia final del trabajo.

En mi experiencia personal, he de destacar que nunca me había enfrentado a un proyecto donde tuviera que crear 2 aplicaciones tan diferentes y un back-end del cual tuvieran que alimentarse ambas. Ha sido todo un reto y he aprendido muchísimo, sobre todo, en la parte de backend donde nunca había trabajado con Spring-Boot. Por la parte de React y React Native si que la había trabajado antes pero, ha servido mucho para reforzar esos conocimientos que ya tenía. A día de hoy, me siento bastante ilusionado con la salida de este producto a un entorno real, donde ayudará a las personas en su día a día.

Este trabajo no termina aquí. Por una parte, será presentado en los próximos meses a la comunidad de regantes, con el fin de poder afinar un poco más todo el flujo y poder pulir los detalles que necesitan para terminar sacando una versión final a producción, en un periodo corto de tiempo.

Por otro lado, ambas aplicaciones sufrirán diferentes mejoras como son las siguientes:

- Mejorar la calidad del mapa de la aplicación móvil. Para ello se buscará una nueva librería o inclusive se podría plantear el desarrollo de los componentes nativos e importados a través de un bridge que se cree entre React Native y Android/iOS.
- Mejoras en la seguridad de ambas aplicaciones. Actualmente el personal de administración puede conocer cual es la contraseña que usan los trabajadores por lo cual, la mejora sería utilizar un sistema de registro de usuarios donde se les envíe un correo electrónico y estos tengan que cambiar su contraseña inicial generada de manera automática.
- Tener un histórico global de las medidas. La idea principal sería añadir una nueva

pestaña en la página web donde el usuario pueda filtrar por contador y fechas un conjunto de medidas.

- Mejoras en la interfaz y flujo. Mejorar la interfaz a nivel de UX/UI así como de accesibilidad para que personas con discapacidades tambien puedan usarla sin ningún problema.
- Mejorar la seguridad del backend.
- Introducir testing en las aplicacines con librerías como *@react-testing-library*.
- Al ser un producto real, debería intentar aplicarse la metodología Devops y crear entornos de *stage* y producción haciendo uso de sistemas de integración y despliegue contínuo (*CI/CD*).

Capítulo 7

Summary and Conclusions

From the beginning, the main objective of this work has been to improve the work experience of the workers in *Comunidad de Regadío Alto los Baldíos*. Te creation of these two applications, has allow to reach the targets and save time billing issues, for the community. This means that more time can be used to other tasks like improve the irrigation service, get new members, expanding to other municipalities, etc.

As it project will have a real implication, it has had a lot of collaboration from the community as well as the teachers, who have given a different ideas that have eased and improve the final work experience.

I have never been working in a project with two applications and a back-end. It was a challenge for me, I have learned a lot about Spring-Boot because a I had never worked with this Framework before. On the other hand, I had worked with React and React Native but with this project I have improved my skills. I am very hopeful with the release of this product because it could be a product that it will help many people in their work.

This is not the end of this project. First at all, it will be presented to the community to make the last changes and improve the details of project. Then, the project will be release in a short period of time.

On the other hand, these applications will experience different improvements:

- Improve the Map of the mobile application: For this objective, we will investigate for new libraries to work with maps in React Native or maybe, other idea could be create a native view with a link to React Native using a native bridge.
- Improve the security of the applications: Now, the management staff can see all the passwords of the users. The solution is create a register system that allow the users create accounts with self-generated password and send an email to change it.
- Have a global history of the measures: The main idea is add a new tab on the web application where the user can filter by water meter and by dates and then show the results of this sets of measures.
- Improve the interface and flow of the applications: Improve the UX/UI and improve the accessibility for people with disabilities.
- Improve the Back-End.

- Add testing with libraries like @react-testing-library.
- As it is a real product, DevOps methodology should be applied and stage and production environments should be created, using integration systems and continuous deployment (CI/CD).

Bibliografía

[1] Vi2Water https://www.vi2web.es/project/vi2water/ (Junio 2020)

[2] Ultrimis http://www.apator.com/en/offer/water-and-heat-metering/water-meters(Junio 2020)

[3] Comunidad de Regadío Alto los Baldíos https://app.einforma.com/servlet/app/portal/EME/id_sess/00111258706000181182190000070776/prod/ETIQUETA_EMPRESA/nif/E38401873 (25 de Junio 2020)

[4] AngularJS https://en.wikipedia.org/wiki/AngularJS (Marzo 2020)

[5] VueJS https://en.wikipedia.org/wiki/Vue.js (Marzo de 2020)(Marzo 2020)

[6] ReactJS https://en.wikipedia.org/wiki/React_(web_framework)(Marzo 2020)

[7] Flutter https://es.wikipedia.org/wiki/Flutter_(software)(Marzo 2020)

[8] Ionic https://en.wikipedia.org/wiki/Ionic_(mobile_app_framework)(Marzo 2020)

[9] React Native https://en.wikipedia.org/wiki/React_Native(Marzo 2020)

[10] NodeJS https://nodejs.org/es/(Marzo 2020)

[11] ExpressJS https://expressjs.com/es/(Marzo 2020)

[12] Spring https://spring.io/(Marzo 2020)

[13] Spring-Boot https://spring.io/projects/spring-boot(Marzo 2020)

[14] Firebase https://firebase.google.com/?hl=es(Marzo 2020)

[15] TypeScript https://www.typescriptlang.org/(Abril 2020)

[16] react-native-map https://github.com/react-native-community/react-native-maps/(Abril 2020)