Enrique Molina Giménez

DESARROLLO DEL FRONTEND DE APLICACIÓN MÓVIL DE ANÁLISIS DE ENERGÍA EN INSTALACIONES DE CARGADORES DE VEHÍCULOS ELÉCTRICOS

TRABAJO FINAL DE GRADO

Dirigido por Germán Galià Beltrán y Jordi Duch Gavaldà

Grado en Ingeniería Informática

etecnic
ENERGY AND MOBILITY

y

UNIVERSITAT ROVIRA I VIRGILI

Tarragona

2021

Agradecimientos

En estas primeras líneas, querría expresar mi más sincero agradecimiento a la empresa Etecnic por brindarme la oportunidad de poder realizar el trabajo final de grado en el marco de la empresa. Más particularmente me gustaría agradecer a Germán Galià Beltrán la tutorización y apoyo de este trabajo, aspectos que han hecho posible la realización de este proyecto.

Por último, un fuerte abrazo a mi familia, ya que vosotros sois realmente aquellos que con vuestro esfuerzo y apoyo habéis hecho posible poder graduarme como ingeniero en el mundo de la informática.

Resumen

La empresa Etecnic, dedicada a la ingeniería y la movilidad eléctrica, dispone de una aplicación web energysmartmanager.com que muestra datos de consumo sobre instalaciones eléctricas que alojan cargadores de vehículos eléctricos. La finalidad de esta aplicación es poder visualizar y controlar cual es el estado de estas instalaciones eléctricas, cuyos datos son extraídos de analizadores de red que las controlan. Así, podemos visualizar las propiedades del analizador de red, los históricos de consumo de energía y de potencia... etc.

El objetivo de este trabajo final de grado será el desarrollo del frontend de una aplicación móvil para este proyecto. Este proyecto cuenta con un backend web, y se dispone de API para realizar determinadas consultas al servidor. No obstante, se modifica y añade alguna funcionalidad al backend para poder realizar el frontend de la app móvil satisfactoriamente.

La aplicación móvil se creará utilizando el framework Ionic configurado para React. Este framework nos permite exportar el código fuente en forma de dos aplicaciones móviles, una para dispositivos Android y otra para iOS.

Resum

L'empresa Etecnic, que es dedica a l'enginyeria i la mobilitat elèctrica, disposa d'un aplicatiu web energysmartmanager.com que mostra dades de consum sobre instal·lacions elèctriques que allotjan carregadors de vehicles elèctrics. La finalitat d'aquesta aplicació es permet visualitzar i controlar quin es l'estat d'aquestes instal·lacions elèctriques; les dades són extrets d'analitzadors de xarxa que les controlin. Així, podem visualitzar les propietats del analitzador de xarxa, els històrics de consum d'energia i de potència...

L'objectiu d'aquest treball final de grau és el desenvolupament del frontend d'una aplicació mòbil para aquest projecte. Aquest projecte té un backend web, y es disposa d'API per fer determinades consultes al servidor. Tot i que potser cal modificar o afegir alguna funcionalitat al backend per poder treballar el frontend bé.

L'aplicació mòbil és crearà sent servir en entorn de desenvolupament Ionic per a React. Aquest entorn permet exportar codi font en forma de dues apps móbils, una per dispositius Android i una altra per iOS.

Abstract

Etecnic is one business dedicated to engineering and electric mobility. Etecnic has got one web app called energysmartmanager.com that show data about electric installations consumptions connected to electric vehicles charger points. The aim of this application is to visualize and control the status of these electric installation. Data is provided by network analyzers. We can show network analyzer properties, the historic of energy and power consumptions…

The purpose of this final degree job is to develop the frontend of one mobile app for this project. This project has got one backend web, and we can do queries to the server via API. It would be necessary modify or add one or several functionalities to the server.

The mobile app will be created using Ionic and React. This framework allows us to get two mobile apps (one for iOS and one for Android) writing code in the same language.

Índice

Listado de siglas y acrónimos

SI: Sistema Internacional

MVC: Modelo Vista Controlador

DOM: Document Object Model

HTML: HyperText Markup Language

CSS: Cascading Style Sheets

API: Application Programming Interface

NPM: Node Package Manager

JSON: JavaScript Object Notation

XML: Extensible Markup Language

SPL: Sistema de Protección de la Línea general de alimentación

CGP: Caja General de Protección

BaaS: Backend as a Service

i18n: internationalization

UI: User Interface

URL: Uniform Resource Locator

FCM: Firebase Cloud Messaging

1. Introducción

Este proyecto surge del acuerdo entre mi persona y la empresa Etecnic Movilidad Eléctrica de realizar un trabajo final de grado en el marco de la empresa. Etecnic, empresa en la cual había realizado anteriormente las prácticas curriculares en empresa, me dió la opción de hacer un desarrollo informático que podría ser presentado como proyecto final.

Este proyecto será el presente trabajo a explicar, una aplicación móvil cuyo objetivo es visualizar y tener presencia de datos proporcionados por medidores de consumo eléctrico sobre instalaciones eléctricas asociadas a puntos de carga de vehículos eléctricos. Así, podemos visualizar gráficas en las cuáles podemos observar el histórico en un rango temporal de parámetros de consumo tales como energía, potencia, voltaje, intensidad…

La finalidad de la aplicación móvil es que sea utilizada como una herramienta de consulta que sirva a los administradores de red eléctrica como un acceso rápido a la información más relevante de la instalación eléctrica de una manera muy accesible y cómoda, como es utilizar el smartphone que fácilmente llevaremos encima.

Como se tratará de una herramienta de consulta rápida, solamente los datos más relevantes de la instalación eléctrica serán los que aparezcan en la aplicación móvil. En la versión móvil, existe la limitante de que las gráficas a mostrar deben ser sencillas, ya que una gráfica compleja dificultaría su representación e interpretación debido a las pequeñas dimensiones de la pantalla del dispositivo móvil.

1.1. Contextualización

En los siguientes apartados vamos a poner en contexto acerca de cómo funcionan las mediciones de consumo en instalaciones eléctricas y qué elementos intervienen en ellas.

1.1.1. Analizador de red

Un analizador de red es un instrumento eléctrico utilizado para medir las diferentes propiedades de una red eléctrica. Muchos analizadores de red están preparados para ser colocados en cualquier tipo de red eléctrica, en nuestro caso será conectado a instalaciones eléctricas donde hay enchufados uno o varios punto de carga de vehículos eléctricos.

En la siguiente figura podemos ver un ejemplo de como sería gráficamente un analizador de red.

Figura 1. Ejemplo de analizador de red

Un analizador de red obtiene una cantidad de diferentes parámetros sobre la instalación, que enviará al backend de EvCharge Energy para operar con ellos, guardarlos correctamente en su base de datos y mostrarlos debidamente al usuario.

Los motivos de uso de un analizador de red pueden ser muy variados, entre los que más destacar podrían estar:

- Identificar y evitar los excesos de consumo
- Detección de ciertos problemas en la instalación eléctrica, con finalidad de solventarlos
- Medición de la potencia de la instalación en un cierto momento, evitando que se supere la máxima permitida

En definitiva, utilizar un analizador de red nos permite tener acceso a gran cantidad de información valiosa acerca de nuestra de red eléctrica, información que podemos utilizar con gran variedad de fines.

1.1.2. Sistema trifásico

Se trata de una manera de producir, conducir y consumir la corriente eléctrica que está formada con tres corrientes alternas de una sola fase, todas con la misma frecuencia y amplitud, que presentan una diferencia de fase de 120º entre cada una de ellas.

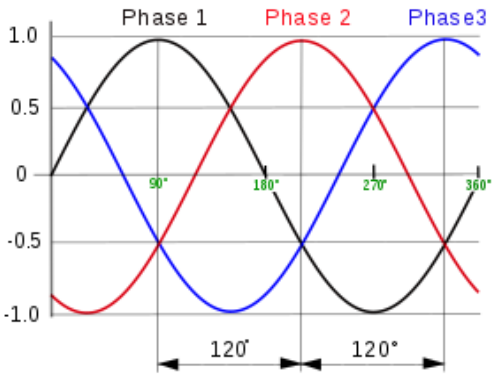

Figura 2. Esquema del sistema trifásico

Este sistema es bastante utilizado debido a que presenta diferentes ventajas significativas, entre las que destacan, un mayor rendimiento en el momento de la producción, una poder constante con respecto a la corriente monofásica y una mayor facilidad de transporte, traducida en un menor coste económico debido a que se puede transportar mayro cantidad de energía con la misma cantidad de material conductor.

1.1.3. Datos de interés sobre las instalaciones eléctricas

Un analizador de red nos permite obtener gran cantidad de parámetros sobre la instalación, los datos de interés mostrados en EvCharge Energy son:

1.1.3.1. Energía activa

Se trata de la energía utilizada por todo aparato eléctrico para funcionar. Es la que más fácilmente podremos encontrar en una factura y es la conocida como "energía útil". Su unidad de medida más común es el kWh.

1.1.3.2. Energía reactiva

La energía reactiva es un tipo de energía eléctrica que algunos equipos eléctricos "absorben" de la instalación, pero que no es consumida por ellos, sino que posteriormente se devuelve a la red siendo reaprovechable. Supone un costo extra porque aunque no sea consumida sí se tiene que generar y conducir hacia los dispositivos eléctricos. Es típicamente medida en KVArh.

1.1.3.3. Potencia activa

Corresponde con la potencia real que consume la instalación, es llamada también "potencia útil". Es comúnmente medida en W o kW.

1.1.3.4. Potencia reactiva

De manera análoga a como pasa con la energía, la potencia reactiva es aquella que tiene que ser suministrada a la red pero que no es aprovechable, es por ello que también la conocen como "potencia no útil". Medida en KVAr.

1.1.3.5. Voltaje

El voltaje o también llamado tensión eléctrica es la fuerza necesaria para empujar los electrones por un material conductor, en puntos entre los cuáles exista una diferencia de potencial eléctrico. El voltaje corresponde con esta diferencia de potencial eléctrico, y su unidad en el SI es el voltio (V).

1.1.3.6. Intensidad

La intensidad de corriente tiene que ver con la cantidad de carga eléctrica que va atravesando un conductor en el tiempo de un segundo. Su unidad en el SI es el amperio (A).

1.1.4. Explicación de la arquitectura de la instalación

Recordemos que la instalación eléctrica tiene conectada a ella uno o varios cargadores de vehículos eléctricos. Se puede decir entonces que los cargadores están incluidos dentro de la instalación eléctrica.

- El consumo en energía siempre será mayor en la instalación completa que en la subinstalación de cargadores.

- La potencia, del mismo modo, siempre será menor en el módulo de cargadores que en la instalación en general.

Así, ya podemos dar unas gráficas de ejemplo, y explicar brevemente como interpretar los datos de ellas.

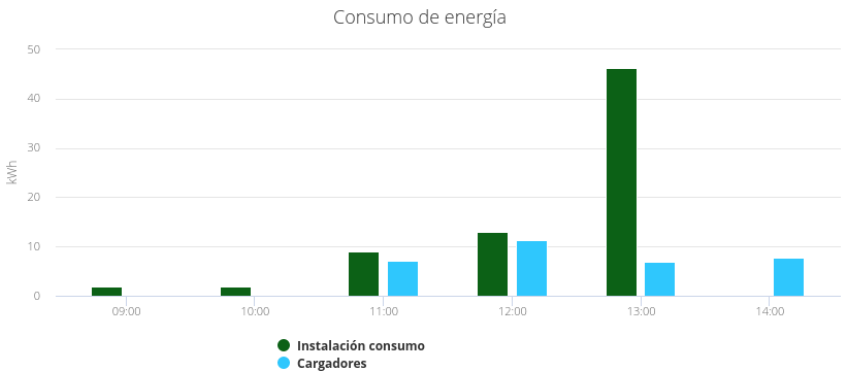

Figura 3. Gráfica de energía de EvCharge Energy

En la gráfica superior podemos observar como el consumo de energía de la instalación al completo siempre supera a la de los cargadores. De la gráfica de energía podemos sacar la siguiente información:

- Consumos reales por franjas horarias tanto de la instalación al completo como sólo de los cargadores.
- Diferencia de consumo entre los cargadores y el resto de la instalación; permite saber cuánto realmente de tu consumo es producido por los cargadores.

A continuación, mostraremos la gráfica de potencia y procederemos a explicar cómo interpretarla.

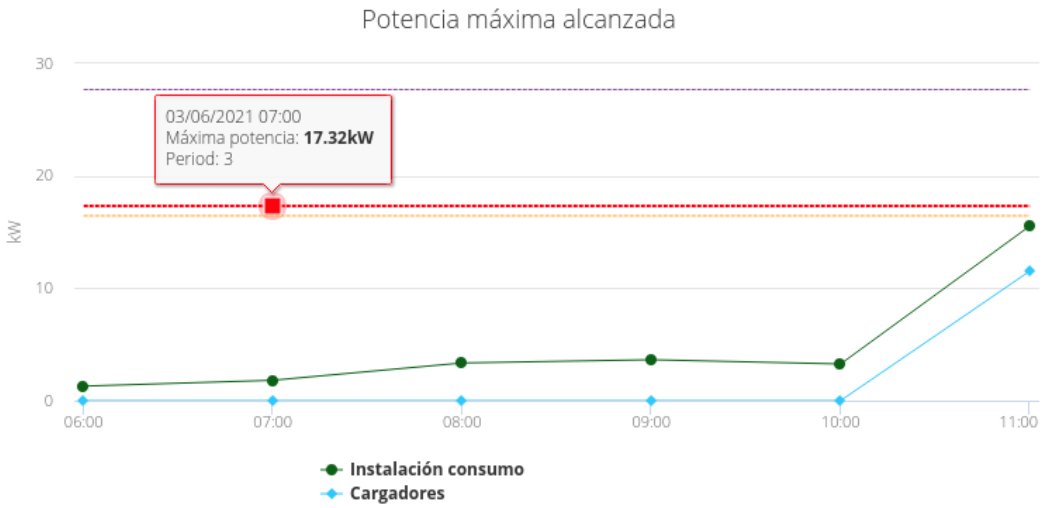

Figura 4. Gráfica de potencia de EvCharge Energy

Esta gráfica, que podemos observar que en lugar de ser de barras es de líneas, muestra el histórico en un rango de tiempo de consumo de potencias activas en unas determinadas horas.

Además de tener los consumos de potencias tanto de la instalación como los de los cargadores (recordemos que el primero siempre es superior al segundo), también tenemos 3 barras horizontales que nos dan información relevante sobre la potencia de nuestra instalación:

- Horizontal roja: se trata de la máxima potencia de la instalación, la cual, en caso de ser superada, se aplica un sobrecoste en la factura de electricidad debido a que es el límite contratado con la compañía. Con lo cual, es recomendable no superar nunca este límite.

- Horizontal amarilla: corresponde con el límite de potencia. Corresponde con la máxima potencia contratada si le restas el 5% de esta, el fin es poder controlar las subidas de potencia de manera que cuando se supere este límite se puede estar alerta de que se roza el límite de potencia contratada, y así se pueden tomar las acciones pertinentes.

- Horizontal violeta: es la máxima potencia admisible y jamás debe ser superada. Superar este límite de potencia podría suponer un daño y perjuicio fatal para la instalación eléctrica, dado que es el límite de potencia máximo soportado por la instalación. Superar esta potencia se traduciría en que se "caerían los plomos" de la instalación con el fin de protegerla de posibles daños.

La motivación principal de estas gráficas de potencia es poder controlar el consumo de potencia y poder saber si en la factura de la luz nos va a llegar un sobrecargo o si por el contrario no.

Cabe destacar que, si bien el portal de EvCharge Energy ofrece más gráficas acerca de otros parámetros y medidas, no las estudiaremos en la presente memoria debido a que tienen un menor nivel informativo que las anteriores. Al tratarse la app móvil a desarrollar de una herramienta simple y accesible de control de la instalación eléctrica, recordamos que se centrará en su simplicidad y facilidad de uso, no siendo su misión convertirse en un complejo panel de control.

1.2. Portal web de la aplicación

Esta aplicación consta de un sitio web desde el que podemos acceder a todas sus funcionalidades, cuya dirección web es energysmartmanager.com

Esta aplicación tiene como usuarios finales a administradores de redes que lo que necesitan es poder observar el estado de la instalación eléctrica, para controlar que todo vaya correctamente, o que algo está sucediendo fuera de lo normal y necesita de acciones correctoras. Esta aplicación tiene un fuerte carácter técnico sobre los datos que ofrece, así que los administradores de red serán personas con una formación técnica suficiente para poder comprender e interpretar los datos mostrados por esta aplicación.

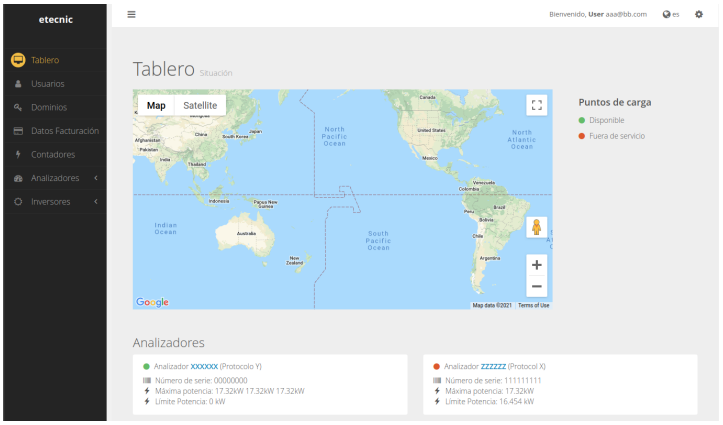

Figura 5. Pantalla de inicio de energysmartmanager.com

Este proyecto está desarrollado en el framework de desarrollo web Ruby On Rails, mediante el cual se puede desarrollar una aplicación web al completo mediante en patrón de desarrollo MVC, y nos permite distinguir entre:

- Frontend: define la disposición y vistas de la aplicación, cómo se presentan los datos al usuario, que elementos gráficos utilizará la aplicación, cuáles serán sus colores, iconos, acciones al pulsar… en definitiva, todas aquellas partes del código fuente que están más directamente relacionadas con aquello que el usuario puede visualizar.

- Backend: consiste en la configuración de todo aquello que queda oculto a la interfaz de usuario, pero que sigue siendo igual de necesario e importante para el funcionamiento de la aplicación. En este apartado entran el diseño y configuración de las bases de datos, cómo instalar y mantener el sitio web en los servidores, el desarrollo de APIs que darán ciertas funcionalidades y control a terceros sobre nuestros datos…

El proyecto existente hasta el momento de EvCharge Energy, dispone de una API documentada que permite realizar ciertas acciones al llamar a sus endpoints. Ciertas funcionalidades de esta API serán utilizadas para el desarrollo del proyecto que nos ocupa, o bien y como se detallará más adelante, se incorporarán alguna nueva funcionalidad con el fin de poder realizar el desarrollo del frontend de la aplicación móvil de manera satisfactoria.

1.2.1. Principales funcionalidades de la aplicación web

Profundizando en la funcionalidades más destacadas de la aplicación web, explicaremos en los siguientes puntos que tareas se pueden realizar con EvCharge Energy:

- En la pantalla de inicio, podemos observar una lista de analizadores de red, listados con sus datos más destacados; además observamos que existe un mapa con el cual podemos geolocalizar los analizadores de red espacialmente.

- Existe un apartado de usuarios donde cada usuario podrá realizar actualizaciones de su perfil si lo desea.

- Podemos acceder a cada uno de estos analizadores, y observar una gran cantidad de información sobre ellos.
    - Un apartado en el cual podemos ver atributos estáticos como número de serie, coordenadas, protocolo que utiliza, cargadores asociados…
    - Otro apartado en el que podemos observar gráficas tecnicas acerca de los siguientes parámetros de red:
        - Energía activa
        - Energía reactiva
        - Potencia activa
        - Potencia reactiva
        - Voltaje
        - Intensidad

      Por cada uno de estos parámetros anteriores, podemos observar gráficas en cada una de sus fases, es decir, podemos visualizar la combinación de todas estas posibilidades. Además, dado que puede haber más de un cargador conectado a estos analizadores de red, también se puede hacer un desglose de los parámetros de red por cargadores, viendo los consumos particulares de cada uno de ellos. Además, podemos seleccionar las fechas entre las cuáles deseamos mostrar estos datos.

      En resumen, podemos concluir que se muestran gráficas complejas y personalizables sobre diferentes datos de la red eléctrica.
    - También se muestran tablas que muestran las estadísticas de consumo, se pueden generar informes sobre los consumos…

1.3. Control de la potencia mediante SPL

El SPL será un sistema inteligente que prevendrá la sobrecarga de la CGP evitando así la apertura de los fusibles. El sistema debe ser capaz de medir la energía que se está consumiendo y en función de ello regular o pausar cargas de vehículos eléctricos.

Sabiendo la potencia consumida de la instalación, la potencia contratada y la admisible, enviamos la orden al cargador de que se ponga en fuera de servicio cuando la de la instalación supera la admisible y en el caso de que haya una carga en curso, si se supera la potencia contratada se envían órdenes de cambio de potencia al cargador para que la instalación no se pase de la contratada por culpa del cargador, llegando a parar la carga si es necesario.

El backend de EvCharge Energy es el encargado de realizar estas acciones; en casos como los anteriores, envía comandos a los cargadores que modifican su comportamiento. Esta

infraestructura está implementada en este caso concreto sobre cargadores, pero sería posible extrapolarla a otras muchas áreas o aparatos eléctricos.

En el momento en el que vuelva a haber disponibilidad de potencia suficiente se volvería a iniciar la carga de manera automática.

En resumidas cuentas, se trata de un sistema “inteligente” de gestión de la energía que nos permite aprovechar los recursos al máximo de la manera más eficiente y con un menor coste económico posible.

2. Tecnologías utilizadas

Para el desarrollo del frontend de la aplicación móvil que nos ocupa, hemos utilizado una cantidad de entornos, tecnologías y herramientas que nos facilitan y hacen posible el éxito del proyecto.

A continuación, vamos a proceder a introducir y explicar brevemente cuáles son cada una de estas tecnologías, y como y en qué medida nos han aportado valor a la hora de llevar a cabo el desarrollo del proyecto.

2.1. Ionic

Ionic es un framework de desarrollo de frontend multiplataforma que se está popularizando, y que nos permite crear aplicaciones móviles mediante típicos lenguajes de desarrollo web tales como HTML, CSS, Javascript…

Fue creado en 2013, y solo en unos pocos años se ha logrado convertir en un framework muy utilizado debido a su principal ventaja: el desarrollo multiplataforma. Ionic nos permite, mediante un mismo código fuente, generar instalables de aplicaciones para diferentes entornos móviles, entre los que por supuesto se encuentran Android e iOS.

Además, Ionic encuentra otra ventaja en que varios lenguajes pueden utilizarlo; es decir, el desarrollador de Ionic puede utilizar React, Angujar, Vue… como lenguajes para crear su aplicación dentro de Ionic.

La decisión tomada a la hora de comenzar el proyecto fue utilizar como lenguaje a React, dado que la tendencia última indica que React se está convirtiendo en uno de los lenguajes más utilizados (junto con Vue); así, desde la empresa fue recomendado escoger a React como lenguaje a utilizar y así fue hecho.

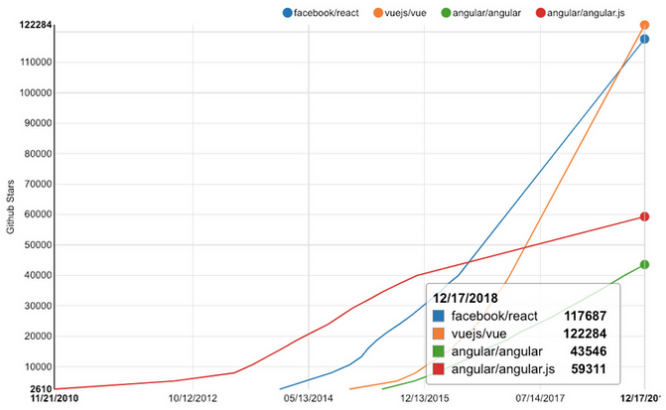

Figura 6. Tendencias Angular-React-Vue en los últimos años

Ionic nos da una gran facilidad a la hora de crear componentes de interfaz de usuario, ya que el framework nos proporciona una gran cantidad de elementos gráficos conocidos como Ionic Components con los que fácilmente (instanciación y personalización) podemos crear interfaces de usuario limpias y amigables.

2.2. React

Se trata del lenguaje escogido para Ionic en el desarrollo de este proyecto. Es una biblioteca Javascript, de código abierto, que permite crear interfaces de usuario fomentando el desarrollo de aplicaciones de una simple página. Facebook y desarrolladores libres son los encargados de dar mantenimiento a esta tecnología.

La tecnología comenzó a tener uso a partir del año 2014, y desde entonces ha tenido una gran aceptación y cuota de uso por parte de la comunidad tecnológica.

Son varias las características de React las que lo hacen destacar y le dan un valor añadido y enfoque a su utilización:

2.2.1. Virtual Dom

React hace uso de un virtual DOM (se trata de un DOM propio, de manera que no solamente se confiará en el DOM del navegador). Esto nos permite conocer las diferencias entre las versión antigua y la nueva, y así podremos saber como actualizar de manera óptima el DOM del navegador, aquel que el usuario puede apreciar. Es decir, React DOM compara el elemento y sus hijos con el elemento anterior, y solo aplica las actualizaciones del DOM que son necesarias para que el DOM esté en el estado deseado.

2.2.2. Componentes stateless y statefull

React nos permite definir componentes stateless y statefull. Los componentes stateless o sin estado son aquellos que no necesitan almacenar parámetros en su memoria, es decir, no dependen de parámetros porque su estado es siempre el mismo.

Pero también tenemos la posibilidad de utilizar componentes statefull o con estado, y son aquellos que sí almacenan datos, que determinan su “estado”, y generalmente son componentes más complejos que hacen uso de métodos y funciones dentro de ellos para determinar su comportamiento.

2.2.3. Ciclo de vida de los componentes

React nos da la posibilidad de determinar qué acciones se producirán en qué determinados momentos. De este modo, resulta sencillo definir qué tarea previa realizar tras renderizar un componente, o qué hacer cuando un elemento deja de estar en el DOM…

Estas funciones nos permiten diseñar una serie de tareas de manera sencilla y unívoca que gestione el flujo de nuestra aplicación de manera correcta.

2.2.4. Lenguaje JSX

Cuando renderizamos las vistas de nuestra aplicación mediante React, no se utiliza código HTML si bien su sintaxis nos recuerda directamente a él. Se trata del lenguaje JSX, que es una extensión del lenguaje Javascript, y nos permite realizar acciones extra que no podríamos con HTML común como almacenar el código JSX en variables o inyectar valores de variables dentro del código JSX.

```
const name = 'Josh Perez';
const element = <h1>Hello, {name}</h1>;
```

Código 1. Ejemplificación de JSX

2.2.5. Hooks y componentes funcionales

En versiones anteriores de React, los programadores se veían obligados a declarar una clase cuando lo que querían era definir un componente con estado o statefull.

En los últimos años existe una tendencia por parte de una cantidad de programadores de programar en la máxima medida de lo posible solamente mediante el uso de funciones, y ahí es donde tienen su cabida los componentes funcionales y los hooks (ganchos) de React.

Un componente funcional es aquel componente que es declarado como una función, y nos permite olvidarnos de las clases para crear componentes. Sin embargo, sin declaración de una clase, no podíamos tener acceso a ciertas funciones de librería. Hasta el momento, sin una clase, no podríamos almacenar el estado de un componente statefull.

Los hooks de React, este los cuáles el más destacado es useState, permiten generar el mismo comportamiento de una clase en un componente funcional y, además, tal y como después fue comprobado, los componentes funcionales entregan un mejor rendimiento en muchos casos con respecto a las clases. Es decir, los hooks de React nos permiten agregar las funcionalidades que hasta el momento eran particulares de las clases a los componentes funcionales, abriendo paso a un paradigma de programación donde las clases no aportan una ventaja frente al uso de funciones.

2.3. Capacitor

Capacitor es la herramienta que nos permite crear soluciones multiplataforma a partir de un proyecto programado bajo los estándares web típicos (HTML, CSS y Javascript).

Tiene un gran parecido al conocido Apache Cordova, y su principal utilización en nuestro proyecto será para, a partir del código fuente web, obtener los instalables tanto de Android como del sistema operativo iOS.

Esta es una solución de gran interés para los programadores ya que permite que desarrollando en una sola plataforma se puedan obtener diferentes aplicaciones para diferentes sistemas operativos, y proporciona abstracción al programador, que no se ha de preocupar de dominar el lenguaje nativo del entorno en el que se ejecutará la aplicación.

2.4. Visual Studio Code

Es el editor de código fuente escogido para desarrollar este proyecto. Es un entorno muy completo, flexible ya que se pueden instalar en él gran cantidad de plugins… Con resaltado inteligente del código, aviso de errores de sintaxis, de falta de importación de librerías, autocompletado de código... se convierte en un editor muy conveniente para aquellos programadores que quieran desarrollar en React.

2.5. NPM

"Node Package Manager", es el gestor de paquetes de Node, que es ampliamente utilizado por los desarrolladores de Javascript para instalar módulos y administrar sus dependencias. Es una herramienta muy potente, y frecuentemente utilizada durante el desarrollo del proyecto, que funciona de dos maneras:

- Es un repositorio ampliamente utilizado para la publicación de proyectos libres. Forma entonces una comunidad de desarrolladores donde unos contribuyen con otros, donde se puede publicar herramientas que otros programadores podrán instalar y utilizar.

- Además, se trata de una herramienta que se utiliza desde la línea de comandos y que nos ayuda a instalar y borrar módulos, gestionar las dependencias de ejecución, gestionar las versiones…

Más adelante, y una vez explicadas las necesidades y funcionalidades a implementar dentro de nuestro frontend, explicaremos en detalle cada uno de los módulos de NPM de los cuales nos hemos servido en este proyecto.

2.6. Peticiones vía API

Los datos que representará el frontend de nuestra aplicación son recibidos desde el backend de EvCharge Energy vía peticiones HTTP.

La API es la encargada de transferir los recursos pertinentes desde los servidores hacia la representación del estado en el lado del cliente del sistema. La representación del estado puede estar definida en diferentes lenguajes entre los cuáles destacan JSON y XML.

La tecnología usada en este proyecto es la arquitectura de API REST, que podemos explicar brevemente a continuación:

- Existe una separación clara entre el lado del cliente y el lado del servidor, siendo el frontend el encargado de pedir los datos que necesita y de representarlos, y el servidor queda encargado simplemente de satisfacer estas peticiones.

- Dentro de la arquitectura RESTful destacan cuatro operaciones básicas:

    - GET: utilizada para consultar acerca de datos en el servidor. No los modifica, simplemente envía los datos oportunos como respuesta a su petición.

    - POST: una petición de este tipo se encarga de crear algún recurso en el servidor de la aplicación.

    - PUT: este método también accede a recursos en el servidor pero en este caso se encargará de modificar alguno de ellos.

    - DELETE: petición encargada de eliminar algún recurso en el servidor.

- El servidor puede ser programado en una gran variedad de entornos (Django, Spring, Ruby On Rails…) que independientemente de ellos las respuestas a las peticiones siempre serán servidas en los mismos estándares (JSON, XML…).

2.7. Inspector de elemento del navegador web

En este caso particular fue utilizado el inspector de elemento de Mozilla Firefox, aunque muchos navegadores cuentan con esta opción muy utilizada a la hora de depurar el código.

Mediante esta herramienta podemos acceder a la consola de Javascript, en la cual podemos visualizar diferentes errores, imprimir mensajes estratégicos utilizados para hacer debugging, observar el DOM… pero además de estas ofrece otras muchas funcionalidades de gran utilidad en nuestro caso para probar la aplicación.

- Consola: nos permite ejecutar funciones de Javascript desde la misma línea de comandos, que no necesita compilar pues este se trata de un lenguaje interpretado. Resulta de gran utilidad cuando desde el código, en un fragmento de interés utilizas la sentencia console.log() para imprimir un texto informativo que te servirá de gran utilidad para, por ejemplo, saber si en un determinado momento una parte del código se ha ejecutado o no.

- Visualizar como dispositivo móvil: se trata de una opción que nos da nuestro navegador que nos permite visualizar el sitio web desde un PC como si de un sitio móvil se tratara. Como el destino final de la aplicación es ser instalada en móviles, esta será una opción que siempre tendremos activada para poder visualizar cómo se vería nuestra aplicación en un móvil real.

- Elementos: es el apartado que nos muestra en primer lugar el código HTML de nuestro sitio web. Nos sirve de gran utilidad para inspeccionar donde se ubica, que clases tiene… cada uno de nuestros elementos dentro del árbol DOM.

2.8. Firebase

Firebase es una plataforma digital (BaaS) que se utiliza para facilitar el desarrollo de aplicaciones web o móviles de una forma efectiva, rápida y sencilla, la cual es utilizada por sus diversas funciones para aumentar la base de usuarios y generar mayores beneficios económicos.

Su principal objetivo, es mejorar el rendimiento de las apps mediante la implementación de diversas funcionalidades que van a hacer de la aplicación en cuestión, mucho más manejable, segura y de fácil acceso para los usuarios.

Se trata de una herramienta que permite introducir funcionalidades complejas como una identificación de usuarios segura, envío de notificaciones push mediante Cloud Messaging, realizar complejos análisis de resultados mediante su herramienta Analytics...

En nuestro caso particular, esta herramienta será utilizada para el envío de notificaciones push a los dispositivos móviles que instalen la app, con la finalidad de poder recibir fácilmente diferentes alertas de su interés.

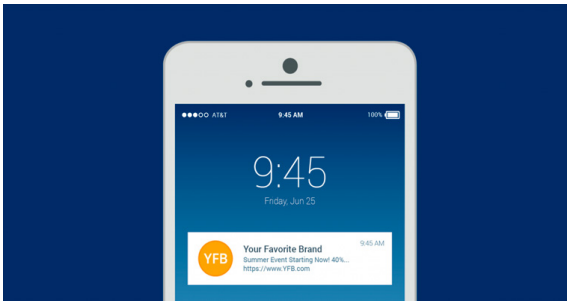

Figura 7. Ejemplo de notificación emergente

3. Propuesta de aplicación a desarrollar

Una vez puestos en situación acerca de qué tema central y finalidad tiene la aplicación, y qué tecnologías se van a utilizan para implementarla satisfactoriamente, vamos a profundizar en qué objetivos queremos conseguir que contenga nuestra aplicación.

Estas funcionalidades aparecen enumeradas a continuación:

3.1. Funcionalidades que incluirá la aplicación

- Una pantalla de inicio de sesión, que nos permita autenticar al usuario que quiere acceder al sistema. Es primordial que el usuario solamente pueda acceder al sistema con sus credenciales (usuario y contraseña) y que se encuentre con una negativa en caso de nos proporcionar estas credenciales correctamente.

  Esta pantalla de inicio de sesión solamente deberá aparecer en el primer inicio de sesión, es decir, si el usuario ha accedido al sistema anteriormente desde su dispositivo móvil, la sesión deberá permanecer hasta que sea el usuario el que decida cerrar su sesión. Este hecho tiene una razón de ser, y es que el usuario no tenga que estar introduciendo sus credenciales cada vez que inicie la aplicación, ya que desembocaría en una mala experiencia y fatiga para el usuario, que constantemente deberá invertir su tiempo en autenticarse.

- Opción de que el usuario pueda recuperar su contraseña. En ocasiones el usuario puede que haya olvidado su contraseña; de ser así no podrá iniciar sesión. Sin embargo, existe una solución ya que aparecerá una opción “recuperar contraseña” que permite al usuario que, tras una verificación de su identidad, pueda crear una nueva contraseña con la que podrá iniciar sesión.

- Una pantalla de inicio, en la que el usuario pueda ver la lista de analizadores que le corresponden. La lista de analizadores deberá ser sencilla y clara, mostrando sus datos más relevantes e identificadores.

- Un mapa, en el cual podamos geolocalizar y posicionar en el mapa los diferentes analizadores del usuario. Los analizadores serán señalados con un punto en el mapa que, además, nos dará información sobre el estado de cada uno de los cargadores mediante la utilización de un código de color.

- Una pantalla de perfil de usuario, que muestre los datos del usuario de la aplicación. Además, el usuario si así lo desea, deberá poder actualizar los datos del usuario en el momento que desee.

- Esta aplicación es propuesta para un público que puede tener diferentes idiomas, es por ello que la aplicación ha sido pensada para integrar i18n y dar soporte para diferentes lenguas. Si bien la lengua por defecto será obtenida del sistema operativo del usuario, el usuario podrá modificar a su conveniencia entre las diferentes lenguas soportadas por la aplicación.

- Se mostrarán datos relativos acerca de copyright, propiedad y legalidad al usuario.

- Se podrá ver el detalle de cada uno de los analizadores. El usuario, al hacer click sobre cada uno de los analizadores, podrá ver los diferentes parámetros más importantes de cada analizador. Además, podrá ver las gráficas de energía y de potencia medias (de las 3 fases) de cada uno de los analizadores y sus cargadores asociados; estas gráficas mostrarán las estadísticas de las seis últimas horas.

- Queremos que la aplicación sea capaz de recibir notificaciones de avisos relevantes acerca de posibles errores o datos de interés sobre cada uno de los analizadores. Para ello, se deben implementar notificaciones push que nos lleguen al móvil y nos avisen en cualquier momento.

- Por último, se deja al usuario la opción de cerrar sesión en la aplicación.

A partir de las especificaciones fijadas en las líneas anteriores, podemos extraer una propuesta de distribución de las diferentes páginas y componentes que conformarán nuestra aplicación.

Así, en el siguiente esquema, podemos observar representando mediante flujos la navegación que deseamos obtener a la hora de desarrollar la aplicación, de manera que contaremos con las siguientes pantallas:

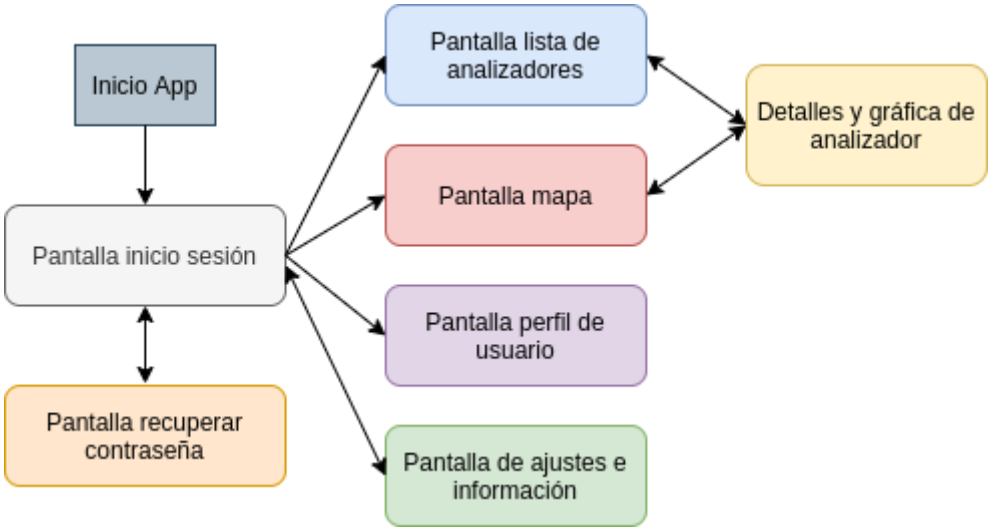

Figura 8. Flujo de navegación de la aplicación

Para ser más gráficos, podemos representar el gráfico anterior sustituyendo la frase explicativa por la pantalla que ocupará su lugar.

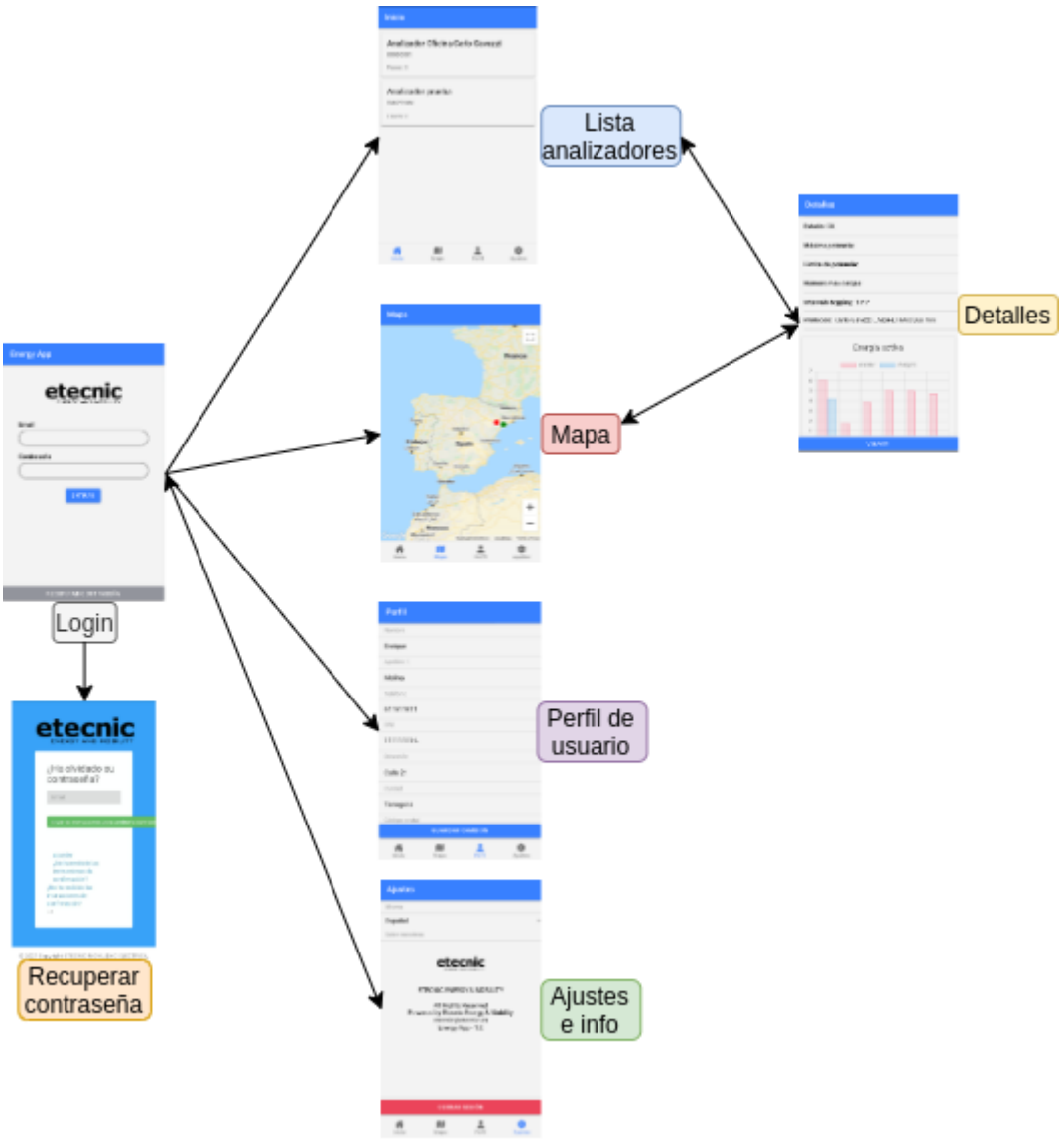

Figura 9. Navegación por la diferentes pantallas de la aplicación

Una vez analizados los requisitos y funcionalidades que queremos que presente nuestra aplicación, llega el momento de explicar su desarrollo e implementación.

4. Implementación del frontend

4.1. Configuración del entorno e inicialización

Lo primero de todo consiste en instalar todas aquellas herramientas indispensables para el desarrollo de aplicación móvil. En la siguiente lista encontramos aquellos que necesitaremos para arrancar con el proyecto.

- Visual Studio Code
- Android Studio
- NPM
- Ionic, React & Capacitor
- Git

Con esto, junto con el navegador web que podemos encontrar en cualquier sistema operativo (en mi caso fue utilizado Mozilla Firefox, browser por defecto de Ubuntu), será suficiente para comenzar.

El proyecto que vamos a crear nació a partir de una de las plantillas de inicio que ofrece Ionic como punto de partida ("Starter Templates") para todo aquel desarrollador que quiera comenzar un proyecto en su framework. La ejecución en línea de comandos de:

```
> ionic start EnergyApp tabs
> ionic init
```

Código 2. Comandos de inicialización del proyecto

nos genera un punto de partida para nuestro proyecto EnergyApp con una interfaz simple de usuario tabulada. Posteriormente, ya podemos iniciar nuestro servidor de desarrollo y acceder al portal web a través del navegador.

```
> ionic serve
```

Código 3. Comando de arranque del servidor web de desarrollo

En la siguiente imagen podemos observar gráficamente cual es la interfaz de nuestro punto de partida:

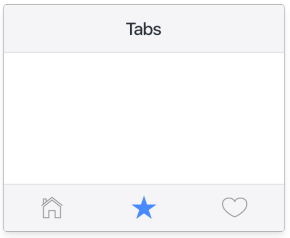

Figura 10. Plantilla de inicio de la aplicación

4.2. Configuración del los plugins necesarios

Nos damos cuenta de que, para el desarrollo de la aplicación, va a ser necesaria la utilización de diferentes plugins que den funcionalidades añadidas a nuestra aplicación. Nos vamos topando con necesidades a satisfacer durante el desarrollo del proyecto que nos requieren la utilización de estos plugins.

Comentamos en primer lugar los diferentes planteamientos y motivos que nos han dado lugar al uso de estas herramientas, para después explicar qué plugin utilizamos:

- "El usuario podrá iniciar sesión en su dispositivo y, una vez iniciada, la sesión se mantendrá abierta hasta que el usuario decida por cuenta propia cerrar su sesión".

  Este requerimiento necesita que la aplicación pueda almacenar datos de manera persistente, estos datos serán por ejemplo el identificador de usuario, de manera que si este identificador está guardado significa que la sesión está abierta y si no se encuentra en la base de datos significa que no hay ninguna sesión activa.

  Para lograr obtener el comportamiento de almacenamiento persistente en la aplicación utilizaremos el paquete capacitor-data-storage-sqlite, acompañado de su gancho o hook react-data-storage-sqlite-hook. Se trata de una librería de guardado permanente de estructuras de tipo clave-valor, que permite crear una manera de almacenar datos para aplicaciones sencillas; la utilización de este paquete nos resulta perfecta para la funcionalidad que deseamos.

  Podemos importar en el código el hook de sqlite:

```
import { useStorageSQLite } from 'react-data-storage-sqlite-hook'
```

Código 4. Importación del hook de la base de datos

  y hacer uso de sus muchas funciones, que entre las que más utilizamos destacan:

  - `getItem('key')` → Devuelve el valor almacenado en esa key
  - `setItem('key', 'value')` → Guarda la estructura clave-valor
  - `openStore()` → Crea una tabla donde guardar los datos

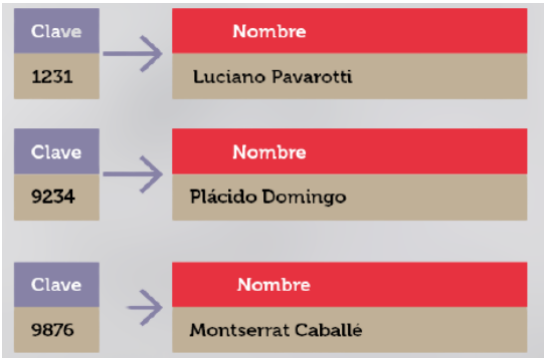

Figura 11. Ejemplo gráfico de almacenamiento clave-valor

- "Habrá un mapa donde los usuarios podrán geolocalizar y visualizar los analizadores, donde además cada punto que los señala en el mapa tendrá un código de color sobre el estado del analizador, indicando si se haya OK o fuera de servicio.

  Hay una variedad de paquetes de mapas que pueden ser utilizados en aplicaciones, sin embargo, en nuestro caso nos decantamos por la utilización de los mapas de Google, los más utilizados en la actualidad. Necesitamos una librería de mapas compatible que además, entre sus funcionalidades, nos de la posibilidad de colocar Markers o puntos posicionables en el mapa.

  Encontramos estas funcionalidades en el paquete google-map-react, donde podemos renderizar un mapa simplemente importando

```
import GoogleMapReact from 'google-map-react';
```

Código 5. Importación del mapa

  y devolviendo el código JSX con la etiqueta `</GoogleMapReact />`.

Figura 12. Ejemplo de mapa de Google Maps

- "Habrá gráficas que muestren los datos de estadísticas de los analizadores de red y de los cargadores asociados a ellos en el último periodo de tiempo, tanto en términos de energía activa como en términos de potencia activa".

  Para ello, necesitamos una librería de gráficos que nos permita realizar los dos tipos de gráficos que queremos implementar en este proyecto:

  - Gráfico de barras: para representar la potencia activa de los analizadores y de los cargadores

  - Gráfico de líneas: para representar la potencia activa igualmente de analizadores e inversores

  Existen también una cantidad variada de librerías que nos permiten dibujar este tipo de gráficas. En un primer lugar fue probada Highcharts pero las gráficas no quedaban adaptadas a las dimensiones de las pantallas del móvil.

Este es un tema crítico, pues lo que buscamos es poder representar gráficas que sean lo más legibles y claras posibles, y al ser la pantalla del móvil pequeña, necesitamos una herramienta que nos renderice estas gráficas de un modo claro, sencillo y sobretodo visual para el usuario que las observa.

Finalmente fue utilizado el paquete react-chartjs-2, que se trata del wrapper de chart.js para React. Se importa mediante la línea de código

```
import { Bar, Line } from "react-chartjs-2";
```

Código 6. Importación de la librería de gráficos

y permite renderizar gráficos de tipo línea mediante la devolución JSX de `<Line ... />` y gráficos tipo barra mediante el código JSX `<Bar ... />`.

Figura 13. Ejemplo de gráficos realizados con Chart.js

- "La aplicación deberá tener soporte para múltiples idiomas, dado que el usuario final de la aplicación podrá tener diferentes nacionalidades". Entre los idiomas contemplados para el desarrollo de la aplicación se han considerado:
    - Español
    - Catalán
    - Inglés
    - Francés

Para adaptar la aplicación a i18n, utilizamos el paquete react-i18next, que nos permite utilizado el hook useTranslation y aplicar traducciones desde tantos archivos del código fuente como necesitemos. Importamos la librería en cada uno de los archivos mediante

```
import { useTranslation } from "react-i18next";
```

Código 7. Importación de la librería de internacionalización

y mediante el hook podemos obtener las instancias así:

```
const { t, i18n } = useTranslation();
```

Código 8. Hook de i18n

Las traducciones están definidas en el archivo i18n.js. Cada idioma cuenta con JSONs asociados, los cuales contienen una clave común para todos los idiomas y valores diferentes para cada idioma concreto.

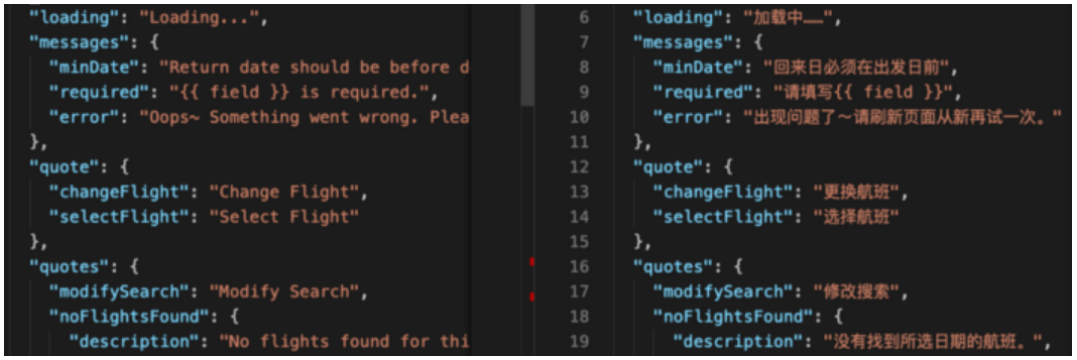

Figura 14. Diferentes JSON para diferentes idiomas

La función t("...") busca en el archivo i18n.js la clave que se especifica en su único parámetro de entrada, y esta función devuelve el valor de dicha traducción. Resulta en una manera muy sencilla de configurar que nos permite desarrollar aplicaciones adaptables a usuarios que dominen diferentes idiomas.

- "El dispositivo móvil del usuario recibirá notificaciones emergentes con mensajes relevantes acerca de diferentes acontecimientos y causas del entorno de la aplicación".

  Para ello, como recalcábamos antes, se hace uso de la utilidad Cloud Messaging de Firebase. Dejamos para más adelante la explicación de implementación de esta utilidad.

Además de las especificaciones extraídas directamente de los requisitos que ha de incorporar la aplicación, también se necesitarán otras herramientas comunes seguramente a la mayoría de apps pero que cabe destacar en esta sección.

- Será necesario poder hacer peticiones a la API del backend de EVCharge Energy para poder obtener los datos que representaremos en el frontend de la aplicación.

  Podemos utilizar las funciones de XMLHttpRequest de Javascript para hacer peticiones al servidor, que nos devolverá los datos a presentar en la app.

  Explicaremos brevemente la estructura de estas peticiones:

  - Las peticiones HTTP al servidor de EvCharge Energy hacen una autenticación en el lado del servidor, que se ejecuta al llegar la petición al servidor. El servidor recibe un Header

    ```
    Authorization: Token token="hidden_token"
    ```

con el cual se hace la identificación de usuario (dado que cada usuario tienen asociado un token de autenticación único en la base de datos); así es capaz de reconocer qué usuario hace la petición y responden de la manera correspondiente.

- Las peticiones HTTP serán en su mayoría de tipo GET, pues la aplicación mayormente presenta datos y estadísticas. Las peticiones GET serán en su mayoría asíncronas, permitiendo que el hilo de ejecución de la aplicación continúe en curso mientras la petición se está ejecutando, y no deteniendo así la interfaz de usuario, lo cual daría una muy mala experiencia de uso a este.

  Sin embargo, el uso de la petición GET síncrona viene como anillo al dedo a la hora de autenticar al usuario mediante el inicio de sesión. El hilo de ejecución se detiene, se espera a obtener los datos del servidor y, según la respuesta, se permite que el usuario inicie sesión o no.

- Mientras se cumplen las peticiones HTTP hechas al server, se mostrará al usuario loaders de manera que pueda recibir feedback de que el dispositivo está realizando una tarea y no dé la sensación que el dispositivo no responda.

  Disponemos del paquete de Javascript "react-loader-spinner", que nos aportará ayuda en este asunto.

  Podemos dibujar una variedad de diferentes loaders solo con su devolución en JSX `<Loader .../>`.

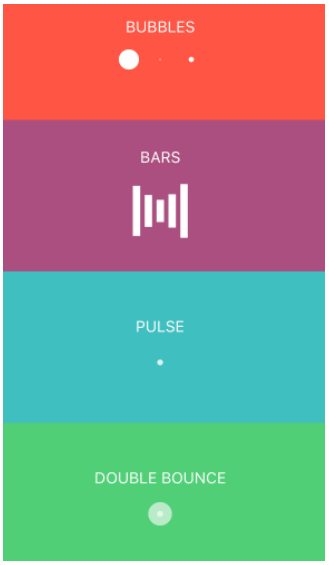

Figura 15. Ejemplos de Loaders

Es necesario también establecer cuál va a ser la navegación que deseamos aportar al cliente en una aplicación móvil (respetando la figura 9). Para ello, se dispone de la herramienta IonReactRouter, la cual permite definir diferentes rutas de navegación dentro de nuestra aplicación, asociar pantallas a determinadas rutas,

establecer acciones de navegación a suceder determinados sucesos (presionar sobre un botón, por ejemplo).

4.3. Explicación de la estructura del proyecto

Vamos a desarrollar a continuación cuál es la estructura del proyecto de frontend de EvCharge Energy. Explicaremos brevemente los directorios y archivos más relevantes a la hora de escribir el código:

- `src/` : es el directorio que contiene el código fuente ejecutable principal de la aplicación.
    - `src/components/` : los componentes de React son elementos gráficos que aparecen de manera recurrente en la aplicación. Se podría decir que son elementos que serán renderizados en varias posiciones diferentes de la aplicación.
    - `src/pages/` : son las páginas que van a componen nuestro proyecto. Las páginas son elementos de la UI que ocupan toda la pantalla y dibujan dentro de ellas diferentes componentes que componen la página.
    - `src/theme/` : contiene aquellos archivos relacionados con el diseño de la aplicación (colores, estilos CSS, imágenes…).
        - `src/theme/variables.css` : es el archivo que contiene la definición de variables globales de colores y estilos de Ionic. Además, se ha utilizado también para definir nuestras propias clases de estilo CSS que utilizaremos para dar formato a nuestros elementos de interfaz gráfica.
    - `index.tsx` : es el fichero padre de la ejecución de Ionic. Se ejecuta cuando se arranca la aplicación y es el encargado de renderizar todos sus componentes, páginas… situados bajo él.
    - `App.tsx` : es el fichero padre de todo el resto de páginas y componentes. Todo componente y página tiene como raíz o padre el elemento <App />.
- `capacitor.config.json` : archivo de configuración de diferentes parámetros de Capacitor.
- `ionic.config.json` : archivo de configuración de diferentes parámetros relacionados con Ionic.
- `ts.config` : archivo de configuración de diferentes ajustes relacionados con el lenguaje y sintaxis Typescript

Además de estos directorios y archivos destacables (los de arriba son comunes a cualquier aplicación móvil escrita en Ionic y React, se implementan los que enumeramos a continuación:

- `src/components/Chart.js`
- `src/components/Details.tsx`
- `src/componentsMarker.tsx`
- `src/pages/Home.tsx`
- `src/pages/Homepage.tsx`
- `src/pages/Login.tsx`
- `src/pages/Map.tsx`
- `src/pages/Profile.tsx`
- `src/pages/Settings.tsx`

Estos archivos corresponden con la definición de la interfaz gráfica y las acciones de las diferentes pantallas que visualizaremos en la aplicación móvil.

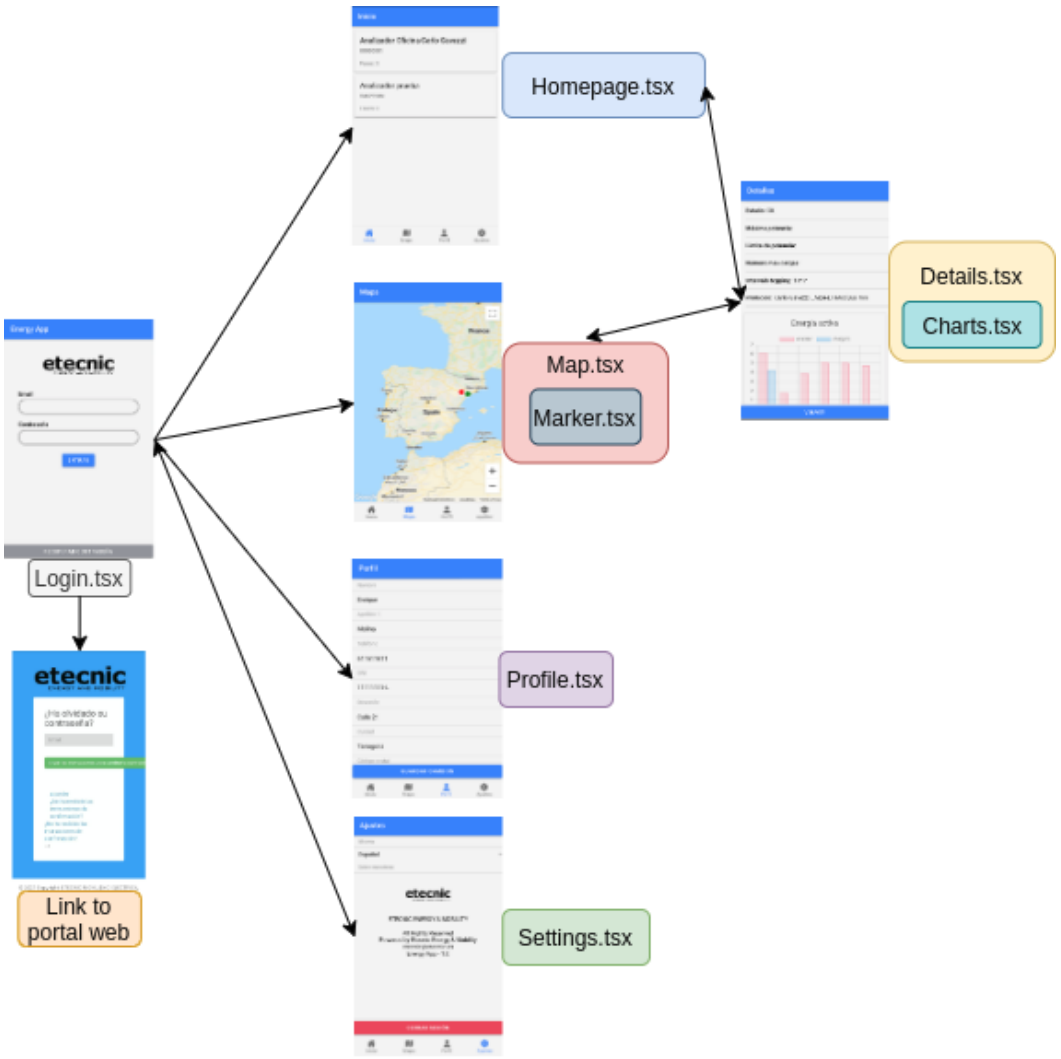

Figura 16. Relación entre pantallas y archivos

A continuación, trataremos en detalle cada una de estas páginas, entrando en el detalle de sus funcionalidades y explicando su comportamiento más a fondo.

4.4. Explicación de las diferentes pantallas de la aplicación

4.4.1. Archivo App.tsx

Es el elemento padre de todo el resto del proyecto, todo el resto de componentes y de páginas se hallan bajo su jerarquía.

Cabe destacar entre las tareas que realiza este archivo, las siguientes:

- Inicialización de la base de datos SQLite: al arrancar la aplicación por primera vez, este es el archivo encargado de inicializar la base de datos para la aplicación del dispositivo móvil por primera vez.

```
useEffect(() => {
   openStore({}).then(() => {
     ...
   })
}, [])
```

Código 9. Inicialización de base de datos SQLite

Con la acción `useEffect` conseguimos disparar y ejecutar esta tarea cuando el componente es renderizado. En futuros inicios de la aplicación, también se ejecutará este fragmento de código pero, si la base de datos ya ha sido inicializada previamente, la función `openStore({})` no crea una nueva sino que mantiene la ya creada.

- Se encarga de identificar si algún usuario tiene sesión iniciada en el dispositivo o no. Consulta a la base de datos si hay alguna clave cuyo nombre sea email; en caso afirmativo significa que un usuario tiene la sesión iniciada y en caso negativo no.

```
useEffect(() => {
  openStore({}).then(() => {
    isKey('email').then((x) => {
      if (x) setAuthenticated(true)
    })
  })
}, [])
```

Código 10. Identificación de inicio de sesión

Podemos observar que de existir la clave email, el estado de authenticated se convierte en true.

Esta variable authenticated nos sirve para conocer unívocamente si el usuario tiene sesión abierta o no.

Podemos observar que pasar a la pantalla de inicio de sesión o por el contrario visualizar la pantalla de inicio (lista de analizadores) depende justamente del valor de este booleano authenticated.

```
<Route path="/home" render={() => { return authenticated ? <Home />
: Login /> }} />
```

Código 11.Pantalla Login o Home según autenticación

4.4.2. Página Login.tsx

Se trata de la página mostrada cuando el usuario abre la aplicación por primera vez, puesto que todavía no habrá tenido la ocasión de iniciar sesión. También es la pantalla que nos encontraremos cuando cerremos sesión de manera voluntaria desde nuestro apartado "Settings" de la aplicación.

Podemos observar cómo es esta pantalla en la siguiente imagen:

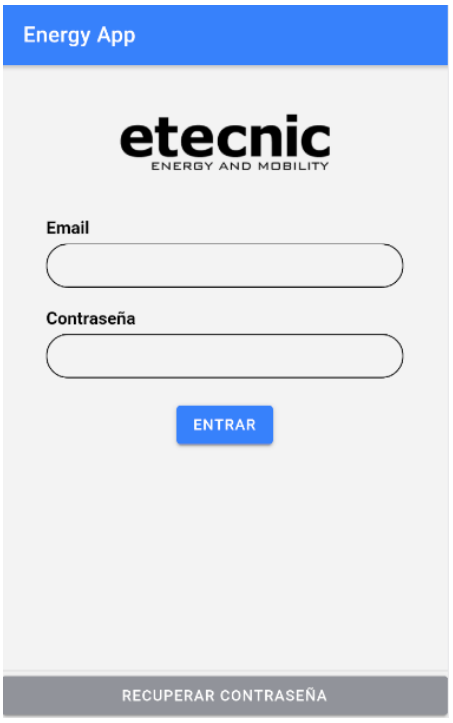

Figura 17. Pantalla de inicio de sesión

El usuario tiene la opción de insertar sus credenciales (usuario y contraseña) en los campos de texto agregados a tal efecto. El usuario también dispone de un botón en el pie de página que le redirige al link de recuperación de contraseña.

El usuario colocará su usuario y contraseña y acto seguido pulsará el botón entrar para iniciar sesión. La pulsación sobre "Entrar" va a disparar una función que se encarga de

hacer la petición API al backend de EvCharge Energy que verifica si un usuario está registrado y si su contraseña coincide.

```
let PROTOCOL = "HTTP"
   let HOST = "energysmartmanager.com"
   let urlSearchParams = new URLSearchParams()
   urlSearchParams.append("email", email)
   urlSearchParams.append("password", password)
   // Preparing URL
   let LOGIN_AUTHORIZATION_URL = PROTOCOL + "://" + HOST +
"/api/v1/energysmartmanager/users/check-authenticated?"
   let URL = LOGIN_AUTHORIZATION_URL + urlSearchParams.toString();
   var xhttp = new XMLHttpRequest()
   xhttp.withCredentials = false;
   xhttp.open('GET', URL, false);
   xhttp.setRequestHeader('Authorization', 'Token token="' + TOKEN + '"')
   xhttp.setRequestHeader('APP_ETECNIC', 'EnergyApp')
   xhttp.onload = function () {
     response = JSON.parse(xhttp.responseText)
   };
   xhttp.send()
```

Código 12. Petición de inicio de sesión

Además, al recibir la respuesta, la función la analiza para resolver si los credenciales del usuario son correctos.

- Si son correctos, el usuario accede a la aplicación y se muestra la pantalla de inicio. Además, se guardan en la base de datos SQLite del móvil valores que podemos necesitar durante la ejecución de la aplicación, y los podemos observar a continuación.

```
setItem('email', email).then(() => {
       setItem('password', password).then(() => {
         setItem('user_id',
response['user_id'].toString()).then(()=>{
           setItem('auth_token', response['auth_token']).then(() =>
{
             putAuthenticated(true)
```

Código 13. Definición de valores en SQLite al iniciar sesión

- Si por el contrario estas credenciales son incorrectas, se muestra al usuario una alerta de que su usuario y contraseña no corresponden con ningún perfil en la base de datos.

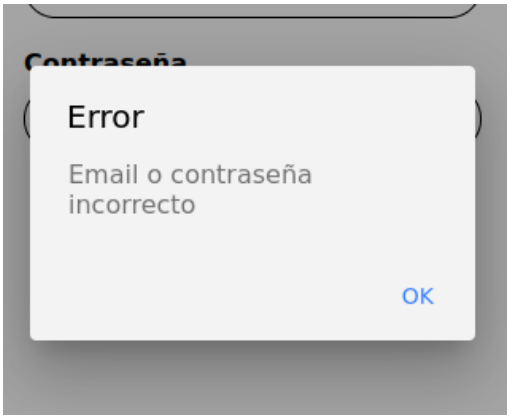

Figura 18. Alerta de fracaso en el inicio de sesión

Además, al iniciar sesión de manera satisfactoria, se procede a la obtención del token de Firebase del dispositivo móvil, y a la publicación de dicho token en la base de datos de EvCharge Energy (más adelante se trata la modificación hecha en el lado del servidor para publicar este token). Básicamente, el frontend lo que hace es registrarse en la plataforma de envío de notificaciones push de Firebase.

```
// Register with Apple / Google to receive push via APNS/FCM
            PushNotifications.register();
```

Código 14. Registro de la aplicación en la plataforma Firebase Cloud Messaging

Y acto seguido se procede a la publicación en el servidor del token de Firebase obtenido, mediante una llamada a API tipo PUT que explicaremos más adelante.

4.4.3. Página Home.tsx

Está pagina corresponde con el tabulador de la aplicación que el usuario puede observar una vez haya iniciado la sesión.

Esta página es la encargada de definir la negación por las diferentes rutas que presenta la aplicación, mediante el uso de la utilidad `IonRouterOutlet`.

```
<IonTabs>
   <IonRouterOutlet>
     <Route path="/home/homepage" component={Homepage} exact={true} />
     <Route path="/home/profile" component={Profile} exact={true} />
     <Route path="/home/settings" component={Settings} exact={true} />
     <Route path="/home/map" component={Map} exact={true} />
     <Route path="/home" render={() => <Redirect to="/home/homepage" />}
exact={true} />
     <Route path="/" render={() => <Redirect to="/home/homepage" />}
exact={true} />
   </IonRouterOutlet>
```

```
  <IonTabBar slot="bottom" >
    <IonTabButton tab="homepage" href="/home/homepage">
      <IonIcon icon={home} />
      <IonLabel>{t("home")}</IonLabel>
    </IonTabButton>
    <IonTabButton tab="map" href="/home/map">
      <IonIcon icon={map} />
      <IonLabel>{t("map")}</IonLabel>
    </IonTabButton>
    <IonTabButton tab="profile" href="/home/profile">
      <IonIcon icon={person} />
      <IonLabel>{t("profile")}</IonLabel>
    </IonTabButton>
    <IonTabButton tab="settings" href="/home/settings">
      <IonIcon icon={settings} />
      <IonLabel>{t("settings")}</IonLabel>
    </IonTabButton>
  </IonTabBar>
</IonTabs>
```

Código 15. Enrutador de pestañas

Se puede apreciar la correspondencia que se define entre páginas y rutas:

- `/home/homepage` nos conduce a la página `Homepage`.
- `/home/map` nos conduce a la página `Map`.
- `/home/profile` nos dirige a la pantalla de ajustes de usuario (`Profile`).
- `/home/settings` nos lleva a la pantalla de ajustes (`Settings`).

Además, el código refleja que se coloca una barra de pestañas en la parte inferior de la pantalla, donde se colocan diferentes botones que nos intercambian entre las diferentes páginas. Estos botones incluyen iconos extraídos de la utilidad ionicons, una fuente de iconos de código abierto para Ionic que nos ofrece iconos sencillos y elegantes.

El resultado de la barra de navegación inferior es el siguiente:

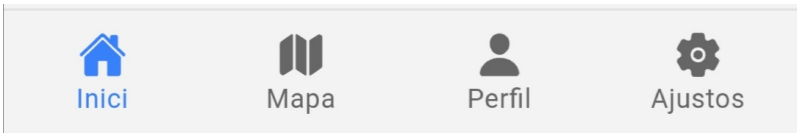

Figura 19. Barra de navegación inferior de la aplicación

4.4.4. Página Homepage.tsx

Es la página que aparece inicialmente por defecto en la aplicación tras iniciar sesión. En la siguiente imagen podemos apreciar una imagen de esta página.

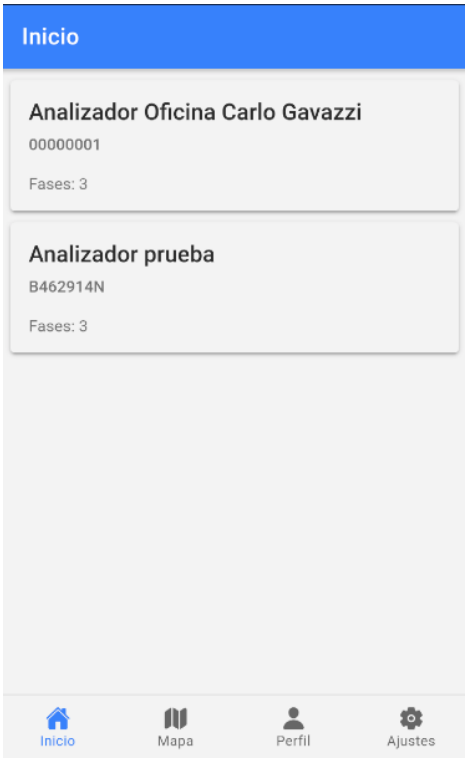

Figura 20. Pantalla de inicio (lista de analizadores) de la app

La funcionalidad que busca ofrecer esta pantalla es mostrar la lista de analizadores del usuario. Para ello, necesita de una función que haga la petición HTTP correspondiente al backend de EvCharge Energy, para después hacer el fetch de estos datos y presentarlos al usuario.

```
function getAnalyzers() {
        ...
        var xhttp = new XMLHttpRequest();
        ...
        xhttp.open("GET", URL, true);
        xhttp.setRequestHeader("Authorization",'Token token="'+token+'"');
        xhttp.setRequestHeader("APP_ETECNIC", "EnergyApp");
        xhttp.addEventListener("load", function (e) {
          console.log(xhttp.responseText);
          var response = JSON.parse(xhttp.responseText);
          setItem("analyzers", JSON.stringify(response["analyzers"]));
          setAnalyzers(JSON.stringify(response["analyzers"]), () => {
            setLoading(false);
```

```
        });
      });
      xhttp.ontimeout = function (e) {
        setShowAlertTimeout(true)
      };
      xhttp.send();
    }
  });
}
```

Código 16. Función getAnalyzers - src/pages/Home.tsx

La función `getAnalyzers` es disparada en el hook `useEffect` de este mismo archivo. Podemos observar como la función superior se encarga de realizar la petición GET a la URL correspondiente mediante `XMLHttpRequest`. Se trata de una petición asíncrona, con las siguientes pautas:

- Mientras se procesa la petición, se muestra al usuario un loader de espera. Este loader tiene la misión de transmitir al usuario la idea de que la aplicación está trabajando en procesar y cargar datos.

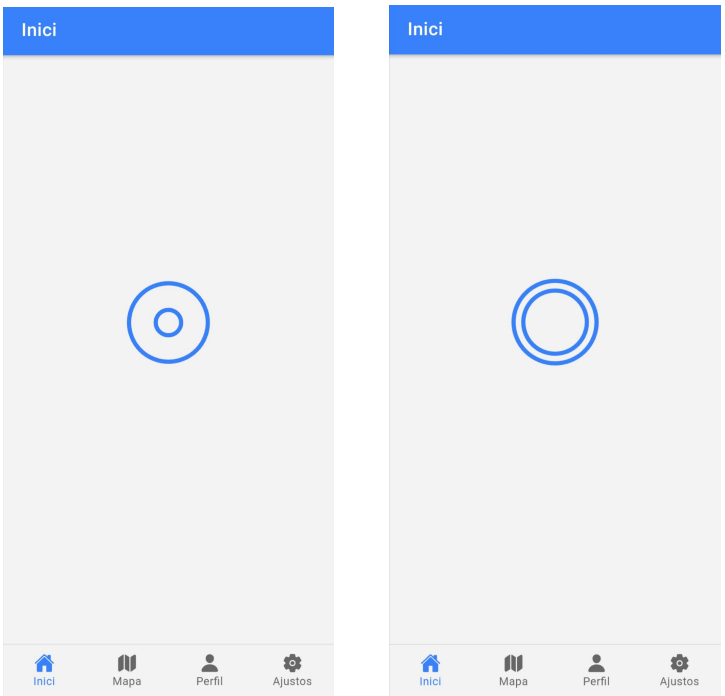

Figura 21. Loader de carga de la lista de analizadores

- Se aplica un tiempo de espera máximo antes del cual la petición tendrá que ser satisfecha. El timeout nos permite definir la situaciones en las cuales por problemas de conectividad o del servidor (entre otras posibles causas) no se recibe una respuesta en un tiempo considerable, y se muestra al usuario un mensaje alertándolo de que se ha agotado el tiempo de espera.

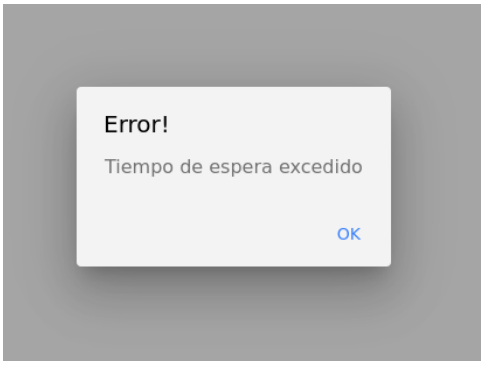

Figura 22. Alerta de tiempo de espera excedido

Una vez se recibe la respuesta a la petición al servidor, se establece la variable isLoading a false, con lo cual pasaremos de renderizar el loader a renderizar la lista de analizadores.

```
<IonContent>
          {JSON.parse(analyzers).map((value) => {
            return (
              <IonCard
                key={value["id"]}
                onClick={() => {
                  setId(value["id"]);
                  setShowModal(true)
                  console.log("ID: " + id);
                }}
              >
                <IonCardHeader>
                  <IonCardTitle>{value["title"]}</IonCardTitle>
                <IonCardSubtitle>{value["serial_number"]}</IonCardSubtitle>
                </IonCardHeader>
                 <IonCardContent>{t('phases')}:
{value["phases"]}</IonCardContent>
              </IonCard>
            );
          })
          }
...
</IonContent>
```

Código 17. Renderizado de cards de analizadores en pantalla de inicio

Cada card estará compuesta de diferentes campos, explicados aquí:

- Nombre del analizador: es el texto superior y más grande, resaltado en negrita. Da información sobre el nombre que recibe dicho analizador.
- Número de serie: se trata de un identificador único atribuido a cada analizador.
- Número de fases: indica el número de fases con las que opera cada analizador. El número de fases más común es 3, se trata de analizadores trifásicos.

Una pulsación sobre una card (analizador), desemboca en la visualización de un modal que muestra todos sus detalles (información sobre el analizador + gráficas de estadísticas).

4.4.5. Componente Details.tsx

Se trata de un modal que contiene toda la información de interés que podemos visualizar en la aplicación sobre un cargador.

En su parte superior muestra una lista de detalles acerca de cada analizador:

- Nombre del analizador
- Número de serie
- Coordenadas (latitud + longitud)
- Dirección IP
- Puerto IP
- Número de fases
- Estado del analizador (Correcto - Fuera de servicio)
- Máxima potencia (muestra 3 valores, 1 por fase)
- Límite de potencia: (muestra 3 valores, 1 por fase)
- Número de cargas máximo
- Intervalo tcpping
- Protocolo que utiliza el analizador

Figura 23. Modal de detalles de analizador

Todos estos detalles del analizador vienen proporcionados en la respuesta del servidor a la petición de API que pide la lista de analizadores.

En esta misma pantalla, podemos observar que tenemos un loader en el fin de los detalles, durante los primeros segundos de visualización de la aplicación.

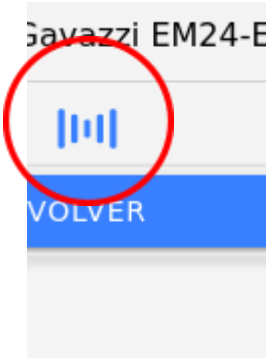

Figura 24. Loader de gráficas de estadísticas del analizador

Tras satisfacer la petición al servidor que devuelve las estadísticas de energía activa y de potencia activa de las 6 últimas horas (funcionalidad explicada en detalle más adelante), se muestran al usuario dichas gráficas a continuación de los detalles del analizador.

4.4.6. Componente Charts.tsx

Se encarga de crear y mostrar al usuario las gráficas de potencia activa y de energía activa de las 6 últimas horas. Analizaremos las diferentes gráficas que se muestran en detalle a continuación:

4.4.6.1. Gráfica de energía activa

La gráfica de energía activa muestra los kWh que han consumido los cargadores (lectura de cargadores) y que ha consumido la instalación al completo (lectura del analizador). Podemos ver una gráfica de ejemplo a continuación, para después explicarla en detalle:

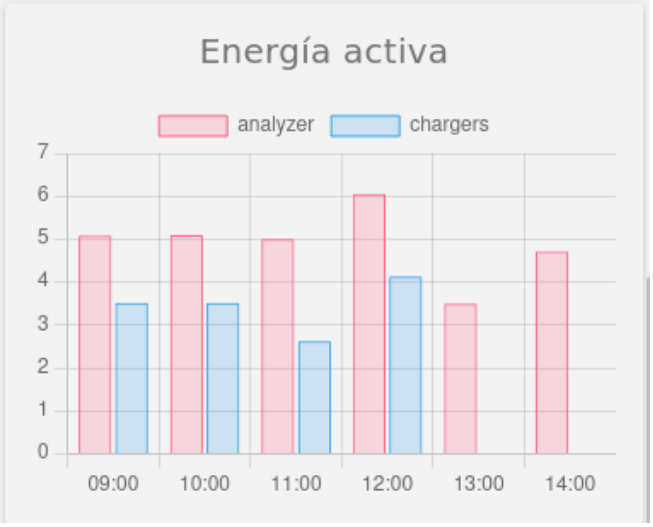

Figura 25. Gráfica de energía activa de la aplicación móvil

Esta gráfica recibe como categorías (valores en el eje de las abscisas) las horas para las cuales se van a mostrar los datos de energía.

Además, se compone de dos datasets, los cuales son:

- Rojo: dataset de lecturas de energía de analizadores
- Azul: dataset de energía de cargadores

A continuación se muestran como se construyen la categoría y los datasets de la gráfica a partir del JSON recibido, y como se define la const data, que es pasada como parámetro al componente de la gráfica.

```
analyzers_hash["chart_data"].map((x) => {
      categ.push(x["date"]);
      energy_chargers.push(x["energy_chargers"]);
      energy_analyzer.push(x["energy_analyzer"]);
      ...
   });
   const data = {
```

```
    labels: categ,
    datasets: [
      {
        label: "analyzer",
        data: energy_analyzer,
        backgroundColor: ["rgba(255, 99, 132, 0.2)"],
        borderColor: ["rgba(255, 99, 132, 1)"],
        borderWidth: 1,
      },
      {
        label: "chargers",
        data: energy_chargers,
        backgroundColor: ["rgba(54, 162, 235, 0.2)"],
        borderColor: ["rgba(54, 162, 235, 1)"],
        borderWidth: 1,
      },
    ],
  };
```

Código 18. Construcción de estructura para dibujar gráfico de energía activa

Así, tras devolver el JSX de

```
<Bar data={data} options={{ maintainAspectRatio: false }} />
```

Código 19. Renderizado de gráfica de energía activa

se consigue dibujar la gráfica de potencia activa que podemos ver en la figura superior.

4.4.6.2. Gráfica de potencia activa

De modo parecido al anterior se crea también la gráfica de potencia activa, aunque cabe destacar que en este caso la gráfica es de tipo líneas y no barras como la energía. En la gráfica observamos los diferentes valores que toman la potencia del analizador, la potencia de los cargadores, y los límites de potencia (límites de potencia explicados en el punto 1.1.4).

Podemos observar como quedaría esta gráfica a continuación.

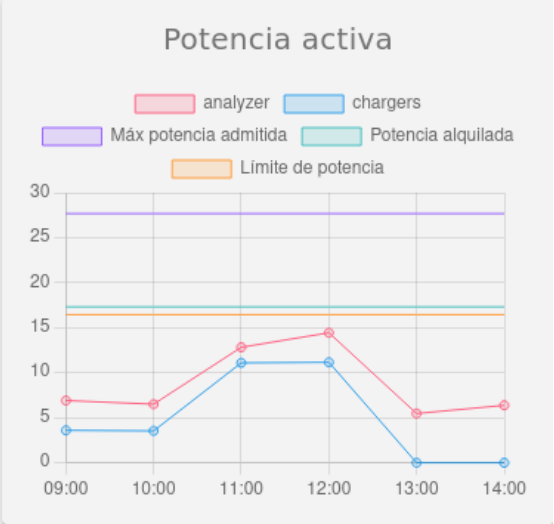

Figura 26. Gráfica de potencia activa de la aplicación móvil

La categoría de la gráfica es exactamente la misma del caso de la gráfica anterior (se corresponden las mismas franjas horarias), y los datasets (que en este caso son 5, y no 2 como anteriormente) se construyen también de manera similar. Sin embargo, podemos ver cómo los datasets de la potencia de los analizadores y de los cargadores tiene puntos insertados en la línea en cada una de las horas. Este es el comportamiento normal de dibujar gráficas de líneas.

Sin embargo, como queremos que la máxima potencia admisible, la potencia contratada y el límite de potencia se muestren como horizontales en la gráfica, lo que hacemos es componer 3 datasets (uno para cada uno de estos valores de potencia) donde todos sus valores serán iguales (tomarán el mismo valor en el eje de ordenadas; como resultado queda una línea horizontal) y acto seguido se indica que estos 3 datasets no muestren puntos en sus líneas. Esta es una manera sencilla que nos permite dibujar una línea horizontal en la gráfica sin la necesidad de tener que importar plugins añadidos.

Finalmente, renderizamos la gráfica mediante:

```
<Line data={data_power} options={{ maintainAspectRatio: false }} />
```

Código 20. Renderizado de gráfica de potencia activa

4.4.7. Página Map.tsx

Esta página muestra un mapa de Google Maps, centrado en el país de España. Podemos hacer sobre este mapa las acciones típicas de cualquier mapa que manejemos en un dispositivo móvil (hacer zoom in y zoom out), desplazarnos a lo largo de todo el mapa… Observamos puntos sobre el mapa que nos indican dónde se hallan dichos analizadores, pues el punto se sitúa sobre la ubicación del analizador.

Figura 27. Mapa de posicionamiento de analizadores

Además, estos puntos sobre el mapa tienen un código de color que nos da información acerca del estado de los analizadores:

- Verde: el analizador funciona de manera correcta
- Rojo: el analizador se encuentra fuera de servicio

```
<GoogleMapReact
            bootstrapURLKeys={{ key: "hidden_token" }}
            defaultCenter={{ lat: 40.416640, lng: -3.7032700 }}
            defaultZoom={5}
          >
            {
              JSON.parse(analyzers).map((value) => {
                var color = ['green', 'yellow', 'yellow',
'red'][value['id_status']]
                return (
                  <Marker
                    lat={value['latitude']}
                    lng={value['length']}
                    text="My Marker"
                    key={value['id']}
```

```
                    color={color}
                    onClick={() => { setId(value['id']);
setShowModal(true) }}
                   />)
               })
             }
          </GoogleMapReact>
```

Código 21. Código relevante del archivo Map.tsx

El código más relevante del archivo lo podemos observar en el fragmento superior. De la lista de analizadores, para cada uno de ellos se dibuja un elemento Marker en las coordenadas (latitud y longitud) que le corresponden.

Además, una pulsación sobre un punto nos mostrará, del mismo modo a como lo hacía una pulsación sobre una card en la pantalla de inicio, la información y gráficas de dicho analizador (al establecer showModal a true se renderizará dicho modal).

4.4.8. Componente Marker.tsx

Este componente es un pequeño punto que puede tomar diferentes colores y que viene definido principalmente mediante CSS, hacemos uso del siguiente estilo para indicar cómo tendrá que ser este pequeño punto de color.

```
const Wrapper = styled.div`
 position: absolute;
 top: 50%;
 left: 50%;
 width: 18px;
 height: 18px;
 background-color: ${(props) => (props.color)};
 border: 2px solid #fff;
 border-radius: 100%;
 user-select: none;
 transform: translate(-50%, -50%);
 cursor: ${(props) => (props.onClick ? 'pointer' : 'default')};
 &:hover {
   z-index: 1;
 }
`;
```

Código 22. Definición del estilo de Marker

4.4.9. Página Profile.tsx

La pagína de perfil de usuario nos permite visualizar los datos de nuestro perfil de usuario. Estos datos coinciden concretamente con:

- Correo electrónico (identificador de usuario, no modificable)
- Nombre
- Apellidos
- Teléfono
- DNI
- Dirección
- Ciudad
- Código postal

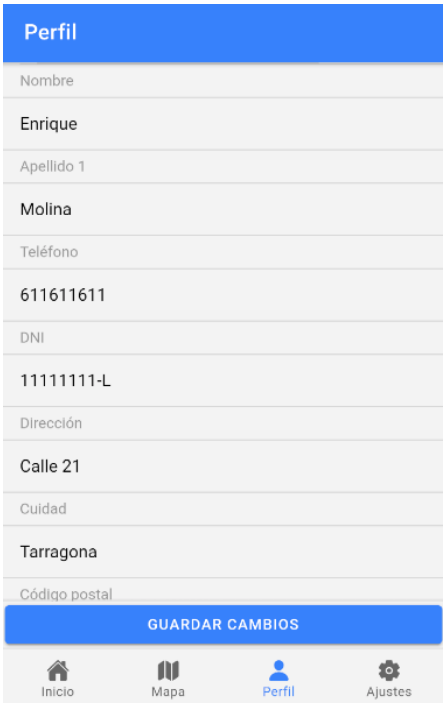

Figura 28. Pantalla de perfil de usuario

Cuando se dibuja este elemento, el hook `useEffect` llama a la función `initProfile`, que se encarga de hacer una petición GET al backend de EvCharge Energy. El frontend recibe como respuesta un JSON con los datos del usuario citados anteriormente. Estos datos se colocan en sus respectivos campos de texto, campos que (a excepción del correo electrónico) son modificables.

Podemos modificar estos campos de texto y guardan los cambios para que nuestro perfil se actualice en el servidor. Ello se hará de nuevo mediante una petición HTTP, en este caso se trata de una petición PUT que indica todos los campos de usuario a modificar como parámetros de la URL.

Al satisfacerse la petición de modificación del usuario en el servidor, una alerta aparece al usuario indicándole que sus cambios se han guardado correctamente.

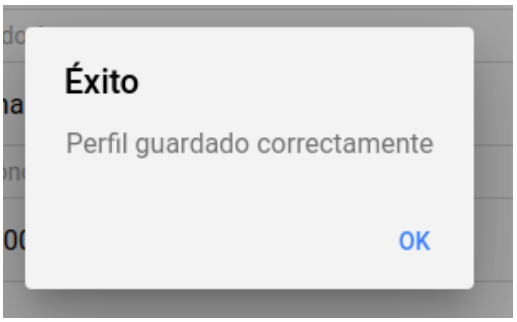

Figura 29. Perfil guardado correctamente

4.4.10. Página Settings.tsx

La página de ajustes nos ofrece tres funcionalidades varias que podemos ver en esta imagen:

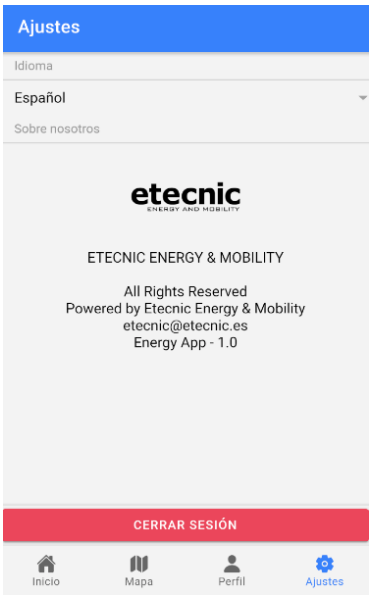

Figura 30. Página de ajustes de la aplicación

4.4.10.1. Selección del idioma

La primera de las opciones consiste en un `IonSelect` que nos permite seleccionar el idioma que deseamos que utilice la aplicación. Podemos elegir entre utilizar la aplicación en español, inglés, francés o catalán. Se muestra el idioma que utiliza actualmente y, al

pulsar sobre este, se despliega un seleccionable donde podemos escoger el idioma que más interese.

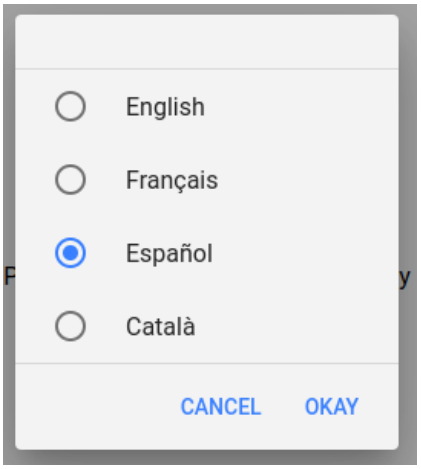

Figura 31. Seleccionable de idioma

4.4.10.2. Información acerca de la empresa

Es una sección donde se pueden observar datos relacionados con la aplicación y la empresa a la que pertenece (ETECNIC). Esta información contiene el logo corporativo de la empresa, el nombre completo de la empresa, la reserva de derechos sobre la aplicación, un email de contacto de la empresa y el nombre y versión de la aplicación.

4.4.10.3. Botón para cerrar sesión

El el pie de la página de ajustes tenemos un botón rojo que nos permite cerrar la sesión del usuario. Pulsar sobre él ejecutará la función `handleLogout()`, que deja la base de datos SQLite del dispositivo móvil vacía y además establece la variable authenticated a false, con lo que acudiremos directamente a la pantalla de inicio de sesión.

```
const handleLogout = () => {
  removeItem('email')
  removeItem('password')
  removeItem('user_id')
  removeItem('analyzers')
  removeItem('chart_data')
  removeItem('auth_token')
  putAuthenticated(false)
};
```

Código 23. Función handleLogout

5. Modificaciones en el backend

En el apartado anterior hemos comentado las implementaciones realizadas para desarrollar el frontend de la aplicación. Ahora bien, aunque el backend de EvCharge Energy es bastante completo y se han podido reutilizar bastantes llamadas a la API en el frontend de la aplicación, ha habido otras peticiones que han sido necesarias implementar específicamente para el desarrollo de esta aplicación móvil.

Además de implementar ciertas llamadas a la API, también han sido necesarios otros pequeños cambios en el backend; todo lo explicaremos en detalle más adelante.

El servidor de EvCharge Energy está programado utilizando el framework de desarrollo web Ruby On Rails, que a su vez se sirve del lenguaje de programación Ruby.

5.1. Modificación de política CORS para peticiones desde app móvil

Anteriormente al desarrollo de esta aplicación móvil, el servidor estaba configurado con su política CORS por defecto. Utilizando peticiones asíncronas, los recursos situados en dominios diferentes al de la página actual no están permitidos por defecto. Es lo que se suele denominar protección de CORS. Su finalidad es dificultar la posibilidad de añadir recursos ajenos en un sitio determinado.

Es necesario modificar la política CORS del servidor para que acepte y sirva este tipo de peticiones. Para ello, instalamos la gema de Rails "`rack-cors`".

Modificando el archivo `config/application.rb`, y añadiendo las siguientes líneas de código

```
config.middleware.insert_after Rails::Rack::Logger, Rack::Cors, :logger =>
Rails.logger do
 allow do
   origins '*' # Add IP origin request to our API (* -> accept all)
   # resource '*', headers: :any, methods: [:get, :post, :patch, :options]
   resource '*', headers: :any, methods: :any
 end
end
```

Código 24. Modificación de política CORS en el servidor

conseguimos que se puedan hacer peticiones al servidor desde cualquier origen, poniendo así solución al problema con el que nos encontrábamos inicialmente.

5.2. Implementación de método GET get_last_6_hours_analyzers_data

La misión es desarrollar un método que nos devuelva las estadísticas de las 6 últimas horas de un analizador concreto. Las estadísticas deben ser sobre la energía activa y la potencia activa del analizadores y de los cargadores asociados a él.

Para ello, añadimos al fichero de rutas del proyecto la línea que nos indica la URL de acceso al método, y la ubicación del controlador que implementa dicho método.

```
get
'energysmartmanager/analyzer_sockets/get_last_6_hours_analyzer_data/:id' =>
'analyzer_sockets#get_last_6_hours_analyzer_data'
```

Código 25. Definición de ruta de get_last_6_hours_analyzer_data

Los datos relativos a los analizadores se almacenan dentro de la base de datos del proyecto de EvCharge Energy, con lo que accediendo a las tablas correspondientes podemos encontrar los datos de estadísticas de las 6 últimas horas de las mediciones de los analizadores. Sin embargo, las estadísticas de los cargadores no se encuentran en este proyecto, sino que se encuentran dentro del proyecto de EvCharge Mobility, que es el encargado de realizar toda la gestión de los puntos de carga, entre otras misiones.

Por lo tanto, nuestro método en el controlador de EvCharge Energy hará peticiones a la API de EvCharge Mobility para recoger las estadísticas de los cargadores, y las juntará con las estadísticas de los analizadores para responder con todas ellas.

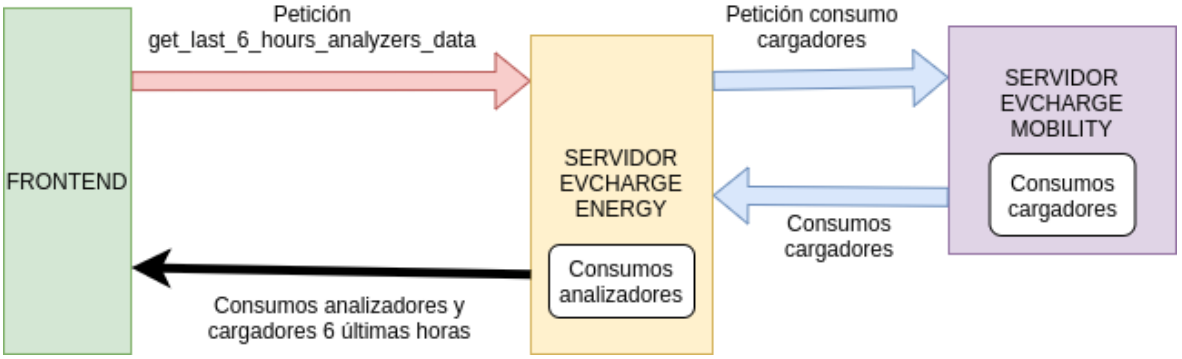

Figura 32. Obtención de estadísticas de consumos

5.3. Migración de la tabla de usuarios de EvCharge Energy

Será necesario almacenar en el servidor el token de Firebase Cloud Messaging que obtenemos cuando iniciamos sesión en la aplicación móvil. Para ello, se crea una columna dentro de la tabla de usuarios de EvCharge Energy de tipo string que pueda contener el token.

El método por el cual se debe modificar la base de datos asociada a un proyecto de RubyMine es mediante las migraciones. Una migración es un archivo que especifica qué cambio o cambios se deben aplicar en la base de datos.

Todas las migraciones que se aplican a una base de datos se almacenan en el directorio db/migrate. Las migraciones se ordenan por antigüedad y, es muy importante que se conserven, pues en caso contrario la congruencia de la base de datos se puede ver expuesta.

En nuestro caso particular de agregar una columna tipo string a la tabla `users`, la migración que definimos es la siguiente:

```
class AddFirebaseTokenToUser < ActiveRecord::Migration[5.2]
 def change
   add_column :users, :firebase_token, :string
 end
end
```

Código 26. Migración crear columna firebase_token

5.4. Implementación de método PUT set-firebase-token

Necesitamos implementar un método PUT que permita guardar en la base de datos el token de Firebase del usuario. Implementamos un método en el controlador de usuarios que realiza las siguientes acciones:

- Identifica al usuario a partir del token de autenticación recibido en los Headers de la petición HTTP.
- Inserta en el campo creado con la migración anterior el token de FCM del usuario.
- Responde a la petición informando de si la acción ha tenido éxito o no.

5.5. Envío de las notificaciones push desde el backend

El backend de EvCharge Energy dispone de alertas enviadas por correo electrónico cuando ciertas reglas se cumplen. Las acciones notificadas relacionadas con analizadores de red son las siguientes:

- Informar de que existe un error de conexión en un analizador.
- Informar de que existe un error relacionado con la potencia en un analizador.
- Informar de que un analizador está devolviendo datos nulos (null).

ActionMailer de Rails se encarga de enviar correos electrónicos cuando estas acciones se cumplen. Buscamos en el código cuándo se llaman a estos envíos de correos electrónicos, y acto seguido a la sentencia que envía el correo, se escribe también la llamada a una función que nos permite enviar la notificación push.

El envío de notificaciones push consiste en hacer una petición a los servidores de Firebase, que recibe el token de identificación, localiza el dispositivo al que tiene que mandar la notificación y la envía.

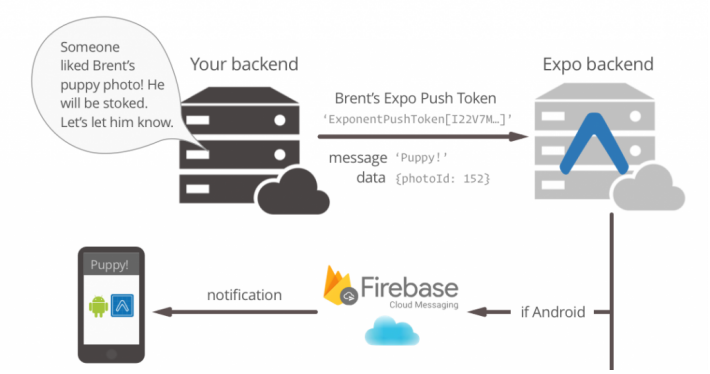

Figura 33. Flujo de envío de notificación emergente

Un ejemplo de petición podría ser el siguiente:

```
POST - https://fcm.googleapis.com/fcm/send

Cuerpo de la petición (ejemplo)
{      "to":"hidden_firebase_token",
       "notification": {
              "title":"Error en analizador 1",
              "body":"El analizador 1 no cuenta con conexión"
}
}
```

Código 27. Envío de notificación mediante petición HTTP

6. Exportación del proyecto en Ionic a plataformas móviles

El objetivo principal por el cuál se desarrolla esta aplicación móvil mediante el framework Ionic es que nos permite fácilmente obtener instalables para diferentes plataformas (Android e iOS entre las más populares).

En nuestro caso, vamos a generar el proyecto para Android de la aplicación, y la instalaremos en un dispositivo Android real sobre el que realizaremos las últimas pruebas de funcionamiento.

Para añadir la plataforma de desarrollo Android a nuestro proyecto de Ionic, podemos lograrlo mediante el siguiente comando:

```
> ionic capacitor add android
```

Código 28. Comando de inicialización del proyecto para Android

Y para crear generar el proyecto en Android, será necesario con ejecutar este otro comando:

```
> ionic capacitor run android
```

Código 29. Comando de arranque del proyecto en Android

Entonces se abre el framework de desarrollo de Android Studio, en el que bastará con compilar el proyecto para obtener el instalable de la aplicación.

Apenas unas pequeñas modificaciones son necesarias en el proyecto de Android; estas modificaciones son las necesarias para integrar el servicio de Firebase Cloud Messaging en el proyecto. Podemos encontrar la información sobre las modificaciones realizadas en https://firebase.google.com/docs/cloud-messaging/android/client.

Obtener el instalable para iOS también sería una opción interesante, debido a que muchos usuarios de smartphones hoy en día hacen uso de un dispositivo del fabricante Mac. Sin embargo, el framework de desarrollo XCode es una herramienta necesaria para poder crear la aplicación para iOS, y XCode solamente puede ser instalado y ejecutado en dispositivos de la propia marca. Al no disponer de un dispositivo con estas características, no ha sido posible obtener la aplicación para iOS. De disponer de eĺ, apenas seguir unos pasos e invertir un mínimo de tiempo nos permitiría obtener la aplicación para iOS.

7. Conclusiones del proyecto y líneas futuras

La realización de este proyecto supone un buen punto de partida como entrada al mundo del desarrollo de software. Durante el último curso del grado, he cursado las asignaturas de la especialidad de computación. Considero que poder desarrollar un trabajo final de grado enfocado en el software me permite aprender y enriquecerme de conocimiento sobre este área, abriendo también las puertas a esta especialidad y practicando sobre ella.

Puedo concluir que se trata de un proyecto que, por seguro, me ha enriquecido académicamente y profesionalmente.

Centrándonos en el proyecto en sí mismo, podemos extraer las siguientes conclusiones:

- Se ha dotado a la aplicación web de EvCharge Energy de una aplicación móvil que, aunque ofrece una funcionalidad más limitada que el portal web, permite disponer de los datos más relevantes de la aplicación de una manera muy rápida y accesible, como es utilizar el teléfono móvil.

- Este proyecto ha servido para darse cuenta de la importancia de la gestión de la energía en las instalaciones eléctricas, más concretamente en nuestro caso sobre la gestión de la energía en instalaciones de cargadores de vehículos eléctricos. Hemos podido observar en este documento cómo gestionar la energía de la instalación se ve reflejado en una ahorro sobre la factura de la luz.

- La realización de este proyecto se traducirá en una mejora para el usuario que utiliza este gestor de energía, ya que además de la funcionalidad web también ahora tiene acceso a la aplicación para móvil.

Ahora bien, este proyecto puede seguir su línea de vida en el tiempo, y añadir más funcionalidades para el usuario; la más urgente sería generar y compilar el proyecto para el sistema operativo iOS, para que así tanto usuarios de la plataforma Android como los usuarios de su competencia directa pudieran tener acceso a la aplicación.

Bibliografía

[1]https://www.certicalia.com/blog/que-es-analizador-redes-electricas

[2]https://es.wikipedia.org/wiki/Analizador_de_redes

[3]https://es.wikipedia.org/wiki/Sistema_trif%C3%A1sico#:~:text=En%20ingenier%C3%ADa%20el%C3%A9ctrica%2C%20un%20sistema,el%C3%A9ctricos%2C%20y%20est%C3%A1n%20dadas%20en

[4]http://circutor.es/es/formacion/energia-reactiva#:~:text=La%20energ%C3%ADa%20reactiva%20es%20un,y%20transportarla%20hasta%20los%20equipos.

[5]https://www.pepeenergy.com/blog/glosario/definicion-potencia-reactiva/

[6]https://tarifasgasluz.com/faq/potencia-contratada/normalizada

[7]https://www.recargacocheselectricos.com/que-es-spl-sistema-proteccion-la-linea-general-alimentacion/

[8]https://ionicframework.com/docs

[9]https://jorgesanchez.net/manuales/abd/bases-sgbd.html

[10]https://wilcityservice.com/how-to-resolve-notification-issue-on-ios-app/

[11]https://www.monterail.com/blog/vue-vs-react-2021#:~:text=One%20of%20the%20biggest%20differences,other%20hand%2C%20there's%20solely%20JSX.&text=React's%20JavaScript%20Expressions%20(JSX)%20combine,and%20CSS%20together%20into%20JavaScript

[12]https://desarrolloweb.com/articulos/caracteristicas-react.html

[13]https://www.arquitecturajava.com/que-es-el-virtual-dom-y-como-funciona/

[14]https://programmingwithmosh.com/javascript/stateful-stateless-components-react/

[15]https://es.reactjs.org/docs/state-and-lifecycle.html

[16]https://es.reactjs.org/docs/introducing-jsx.html

[17]https://midu.dev/react-hooks-introduccion-saca-todo-el-potencial-sin-class/

[18]https://capacitorjs.com/docs

[19]https://developer.mozilla.org/en-US/docs/Web/API/XMLHttpRequest

[20]https://developer.ibm.com/es/technologies/web-development/articles/ws-restful/

[21]https://firebase.google.com/docs

[22]https://github.com/jepiqueau/react-data-storage-sqlite-hook

[23]https://github.com/google-map-react/google-map-react

[24]https://github.com/reactchartjs/react-chartjs-2

[25]https://react.i18next.com/

[26]https://github.com/mhnpd/react-loader-spinner

[27]https://midu.dev/react-hooks-use-effect-funcionalidad-en-el-ciclo-vida-componentes/

[28]https://capacitorjs.com/docs/apis/push-notifications

[29]https://cevichejs.com/6-xhr.html

[30]https://developer.mozilla.org/es/docs/Web/HTTP/CORS

[31]https://www.ionos.es/digitalguide/paginas-web/desarrollo-web/cross-origin-resource-sharing/

[32]https://guides.rubyonrails.org/active_record_migrations.html

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

[34]https://developer.android.com/studio/run?hl=es-419