Trabajo Fin de Grado
Grado en Ingeniería de las Tecnologías de Telecomunicación

# Aplicación móvil para la consulta del consumo de una operadora telefónica con React Native y Expo

Autor: Pablo Diego Sánchez Garrido
Tutor: José Manuel Fornés Rumbao

**Departamento de Ingeniería Telemática**
**Escuela Técnica Superior de Ingeniería**
**Universidad de Sevilla**

Sevilla, 2022

Trabajo Fin de Grado
Grado en Ingeniería de las Tecnologías de Telecomunicación

# Aplicación móvil para la consulta del consumo de una operadora telefónica con React Native y Expo

Autor:
Pablo Diego Sánchez Garrido

Tutor:
José Manuel Fornés Rumbao
Profesor titular

Departamento de Ingeniería Telemática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2022

Trabajo Fin de Grado: Aplicación móvil para la consulta del consumo de una operadora telefónica con React Native y Expo

Autor: Pablo Diego Sánchez Garrido

Tutor: José Manuel Fornés Rumbao

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2022

El Secretario del Tribunal

*A mis padres y a mi hermana*

# Agradecimientos

Estas líneas me gustaría dedicarlas a todas las personas que han formado parte de mi vida durante mi etapa de estudios en Sevilla.

En primer lugar, a mis padres por todo su apoyo y por haber hecho posible el que hoy me encuentre aquí.

En segundo lugar, a todos los compañeros que han pasado esta etapa a mi lado, especialmente a Juan, Fran, Romera y Fernando, por todo su apoyo y las innumerables tardes de estudio en la ETSI.

Por último, agraceder a mi tutor, José Manuel Fornés, por darme a conocer la tecnología con la que he desarrollado la aplicación que se va a presentar en este documento.

*Pablo Diego Sánchez Garrido*

*Huelva, 2022*

# Resumen

El uso de los teléfonos móviles inteligentes o *smartphones* es cada vez mayor a nivel mundial para muchas de las tareas que se realizan a diario. Estos dispositivos acompañan cada vez a más personas en su día a día y son uno de los principales medios por los que se accede a Internet actualmente desde cualquier lugar. Es por ello, por lo que surge la necesidad de adecuar ciertos servicios presentes en Internet a estos dispositivos.

Estamos hablando del mercado de las aplicaciones móviles, cuyo constante crecimiento año tras año ha cambiado el hábito de miles de millones de personas en todo el mundo. Tareas que antes se realizaban con ayuda de un ordenador, ahora es frecuente realizarlas con un teléfono móvil y la aplicación de ese servicio en cuestión de segundos.

Por tanto, el desarrollo de aplicaciones móviles juega un papel fundamental en todo esto, siendo una profesión cada vez más popular. La existencia de varios sistemas operativos móviles, principalmente iOS y Android, tiene en consecuencia la necesidad de aprender el lenguaje de programación y las herramientas específicas para cada plataforma, es decir, un desarrollo nativo.

Con este proyecto se pretende desarrollar una aplicación móvil multiplataforma, es decir, una aplicación que se pueda ejecutar en ambos sistemas operativos sin necesidad de aprender el lenguaje propio de cada plataforma. La tecnología que se va a emplear para ello es React Native, un *framework* para el desarrollo de este tipo de aplicaciones. Además, se usará Expo, un cliente pensado específicamente para React Native. Y todo ello, haciendo uso del lenguaje JavaScript.

# Abstract

The use of smartphones is increasing worldwide for many of the tasks performed on a daily basis. These devices accompany more and more people in their daily lives and are one of the main means by which the Internet is currently accessed from anywhere. This is why the need arises to adapt certain services present on the Internet to these devices.

We are talking about the mobile applications market, whose constant growth year after year has changed the habits of billions of people around the world. Tasks that used to be carried out with the help of a computer are now frequently performed with a cell phone and the application of that service in a matter of seconds.

Therefore, the development of mobile applications plays a fundamental role in all this, being an increasingly popular profession. The existence of several mobile operating systems, mainly iOS and Android, has consequently the need to learn the programming language and specific tools for each platform, that is, a native development.

This project aims to develop a cross-platform mobile application, that is, an application that can run on both operating systems without the need to learn the language of each platform. The technology to be used for this purpose is React Native, a framework for the development of this type of applications. In addition, we will use Expo, a client specifically designed for React Native. And all this, using the JavaScript language.

# Índice

# ÍNDICE DE FIGURAS

# ÍNDICE DE TABLAS

# 1 INTRODUCCIÓN

*La clave del éxito en los negocios es detectar hacia dónde va el mundo y llegar ahí primero*

*Bill Gates*

El constante desarrollo de la tecnología en los últimos años ha provocado una transformación total en el ámbito de la telefonía móvil. Según un estudio realizado en 2020 por el Instituto Nacional de Estadística [1], el teléfono móvil está presente en el 99.5% de los hogares; y según estudios de la CNMC[1], un 86% dispone de un teléfono inteligente o *smartphone*.

El uso de estos dispositivos inteligentes ha provocado que buena parte del acceso a Internet se realice a través de ellos, sustituyendo para ciertas tareas el uso de ordenadores. Las aplicaciones móviles, o *apps*, tienen un papel fundamental en este ámbito, ya que permiten acceder a servicios en Internet de una forma más rápida y cómoda que accediendo desde un navegador web. Por este motivo, las aplicaciones móviles han fundado un mercado que no para de crecer, y cada vez más empresas desarrollan una *app* para acceder a sus servicios. Un mercado que actualmente se puede decir que está liderado por los dos grandes sistemas operativos para móvil: iOS de Apple y Android de Google.

El desarrollo de una aplicación móvil estará, por tanto, ligado al sistema operativo sobre el que se quiera ejecutar, dando lugar a dos caminos diferentes: el desarrollo para iOS empleando el lenguaje Swift o Objective-C; y el desarrollo para Android empleando Kotlin o Java.

Gran parte de las empresas desean lanzar su aplicación para ambos sistemas operativos, por lo que un desarrollador, en principio, debería conocer ambas plataformas, las herramientas para trabajar con cada una de ellas y crear dos aplicaciones, una para cada sistema operativo. Esto es lo que se conoce por desarrollo nativo.

Por el contrario, el desarrollo multiplataforma es aquel que permite el desarrollo de una única aplicación que es capaz de ejecutarse en ambos sistemas operativos, sin necesidad de conocer el lenguaje propio de cada plataforma. Este desarrollo es el que se va a estudiar en este proyecto. Se usará JavaScript como lenguaje de programación y React Native, un marco de desarrollo multiplataforma. También se hará uso de Expo, un conjunto de herramientas creadas para React Native que facilitan el desarrollo de estas aplicaciones.

A continuación, se van a comentar los objetivos que se desean alcanzar con este desarrollo y la estructura que presenta este documento.

[1] CNMC son las siglas de la Comisión Nacional de los Mercados y la Competencia

## 1.1 Objetivos

Los principales objetivos que se desean alcanzar con este proyecto son:

- Desarrollar una aplicación móvil multiplataforma haciendo uso de un único lenguaje de programación, en concreto, JavaScript.
- Emplear React Native como herramienta para este desarrollo.
- Crear una aplicación que permita consultar el consumo de los clientes de una operadora telefónica para así tener un mejor control acerca del gasto que realizan tanto diaria como mensualmente.

## 1.2 Estructura de la memoria

La memoria se ha estructurado en los apartados que se mencionan a continuación, donde se explica brevemente de qué trata cada uno.

- Introducción: se pone en contexto este proyecto y se explican los objetivos que se desean alcanzar.
- Las Bases de React Native: se explican los aspectos más importantes de las tecnologías que conforman los pilares de React Native.
- React Native: se hace una descripción de esta tecnología y se detallan los conceptos fundamentales para el desarrollo de aplicaciones móviles.
- Expo: en este apartado se presenta el cliente de React Native que se ha utilizado para el desarrollo de la aplicación y algunos aspectos de interés.
- Arquitectura y APIs usadas: se detalla la aplicación móvil desarrollada. Su estructura, funcionamiento, así como todas las librerías adicionales y componentes de interés empleados en su desarrollo.
- Pruebas: se detallan las pruebas realizadas para el correcto funcionamiento de la aplicación junto al *backend*.
- Conclusiones y líneas de mejora: se presentan las conclusiones acerca del uso de React Native para el desarrollo de aplicaciones móviles y posibles mejoras.

# 2 LAS BASES DE REACT NATIVE

*Sólo hay un truco en el desarrollo de software, y es utilizar un componente software que ya haya sido escrito*

*Bill Gates*

En este apartado, se van a presentar dos tecnologías que son las que conforman los pilares de React Native: JavaScript y React. Se detallarán los aspectos más importantes de cada una de ellas para el desarrollo de aplicaciones móviles usando React Native.

## 2.1 JavaScript

JavaScript [2] es un lenguaje de programación interpretado, conocido por ser el lenguaje más popular en la programación web. Es un lenguaje interpretado en el cliente, aunque también puede ser interpretado del lado del servidor en entornos como Node.js [3].

Este lenguaje es el mayormente utilizado para el desarrollo de aplicaciones móviles usando React Native, ya que tanto React Native como React, ambos son librerías de JavaScript. Otra alternativa es el uso de TypeScript, un lenguaje similar a JavaScript.

JavaScript fue inventado por Brendan Eich en 1995, y se convirtió en un estándar de la ECMA (*European Computer Manufacturers Association*) en 1997.

A lo largo de toda su historia, se han lanzado diferentes versiones que han ido añadiendo nuevas funcionalidades. De todas las funcionalidades actualmente presentes, se van a comentar a continuación las más interesantes para el desarrollo de aplicaciones móviles.


***Ilustración 1 - Logo de JavaScript***

### 2.1.1 JavaScript Asíncrono

JavaScript es un lenguaje de un solo hilo, esto quiere decir que cuando se ejecuta código, el resultado se obtiene tan pronto como la máquina puede hacerlo. Solo puede ocurrir una cosa a la vez, en un único hilo, bloqueándose todo lo demás hasta que una operación se completa.

Por estas razones, nace la necesidad de implementar el asincronismo en JavaScript [4]. Fundamentalmente en aplicaciones que acceden a recursos externos, ya sea recuperar cierta información de una API o acceder a una base de datos y obtener su respuesta. En todos estos casos, el resultado no lo podemos obtener de forma inmediata, por lo que es necesario que el código espere hasta que reciba la respuesta del servidor antes de continuar con la ejecución y trabajar con dicha respuesta.

Actualmente existen dos tipos de mecanismos para implementar el asincronismo en JavaScript, el primero de ellos son las funciones de *callback* y el segundo, las *promesas*, un concepto más novedoso.

#### 2.1.1.1 Funciones de Callback

Las funciones de *callback* [5], son un tipo de funciones que se especifican como argumentos al llamar a otra función que será la que comience a ejecutar código en segundo plano. Una vez finalice la ejecución de esta última, se llama a la función de *callback* para indicar que la ejecución ha finalizado.

Actualmente, el uso de este mecanismo no es tan utilizado como el que se va a presentar a continuación, ya que el nuevo concepto de *promesa* simplifica todo este procedimiento.

#### 2.1.1.2 Promesas

Una promesa [6], o *promise* en inglés, son un tipo de objeto incorporado a JavaScript en el año 2015 y representan el resultado de una operación asíncrona.

Una promesa puede estar en varios estados: en el momento inicial de la creación, se dice que está *pendiente* de resolver, puesto que aún no tendrá asignada ningún valor. Una vez que se obtiene la respuesta del servidor, se dice que la promesa es resuelta, por lo que pasará a ser una promesa *cumplida* si la operación finalizó con éxito, o bien *rechazada* en el caso de que la operación falle o devuelva algún error.

Actualmente existen varios mecanismos para trabajar con promesas, el primero que se va a presentar es el método `then()`.

El método `then()` [7] es un método del objeto promesa que recibe dos argumentos: el primero de ellos es una función que se ejecutará en el caso de que la promesa se cumpla, y el segundo, de carácter opcional, una función que se ejecutará en el caso de que sea rechazada.

Al finalizar la ejecución de la promesa, el valor devuelto queda almacenado en una variable. Dicha variable adoptará el valor de retorno en caso de que la promesa se cumpla, y el error en caso de que sea rechazada. En la Ilustración 2 se puede observar un ejemplo de ello.

```
var resultado = promesa.then(promesa_cumplida(){
    //código a ejecutar en caso de que la promesa
    //se cumpla
}, promesa_rechazada(){
    //código a ejecutar en caso de error
});
```

***Ilustración 2 - Uso del método then()***

Como se ha comentado, el segundo parámetro es de carácter opcional y se ejecuta en caso de que la promesa sea rechazada. Si obviamos este parámetro y la promesa es rechazada, entonces es posible que obtengamos un error. En este caso, la manera que tenemos de solucionarlo es mediante el uso del bloque `catch`, tal como se muestra en la Ilustración 3.

```
var resultado = promesa.then(promesa_cumplida(){
    //código a ejecutar en caso de que la promesa
    //se cumpla
}).catch(error){
    //código a ejecutar en caso de error
}
```

***Ilustración 3 - Captura de errores con el método then()***

El segundo mecanismo para trabajar con promesas es mediante el uso de las expresiones `async/await`, [8] incorporadas a JavaScript en el año 2017 y cuyo uso es el más frecuente en la actualidad.
Para declarar una función como asíncrona, basta con añadir `async` delante del nombre de la función. Esto hará que el valor que devuelva sea un elemento *promesa* que podría estar en cualquiera de los estados mencionados anteriormente.

Cuando una función `async` contiene la expresión `await`, se pausará la ejecución de esta a la espera de que la promesa sea resuelta, para luego reanudar la ejecución y devolver el resultado. Un ejemplo de ello se muestra en la Ilustración 4.

```
const funcion_asincrona = async () => {
    //cuerpo de la función

};

const resultado = await funcion_asincrona();
console.log(resultado);
```

***Ilustración 4 - Uso de async/await***

Como se puede observar, la expresión `console.log(resultado)` no imprimirá nada en la consola hasta que la promesa haya sido resuelta.

Actualmente, existen en JavaScript multitud de librerías para el manejo del asincronismo. Posteriormente se mostrará el uso de *ApiSauce*, una de las más populares y la que se ha usado para este proyecto.

## 2.2 React

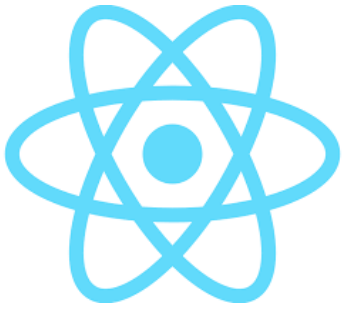

*Ilustración 5 - Logo de React*

React [9], también conocido como ReactJS, es una librería de JavaScript para la construcción de interfaces de usuario, desarrollada por Facebook y lanzada en el año 2013. Constituye otro de los pilares fundamentales de React Native, pero en lugar de usarse para el desarrollo de aplicaciones móviles, se usa para el desarrollo web.

React está basado en componentes [10] -piezas reutilizables de código que se encargan de describir partes de la interfaz de usuario- que cuando se juntan, permiten crear interfaces de usuario más complejas.

Estos componentes, aceptan un tipo de parámetro llamado `props` que representa cualquier valor que se le quiera pasar al componente, y un método `render` encargado de recibir datos y mostrarlos en pantalla. En la Ilustración 6 se puede ver un ejemplo de cómo sería una clase en React.

```
class Ejemplo extends React.Component{
    render(){
        return(
            <div> Hola, {this.props.name} </div>
            );
    }
}
```

*Ilustración 6 - Ejemplo de clase React*

Además del uso de `props`, los componentes pueden almacenar datos en su estado interno. A ellos se accede mediante el uso de `state`. Cuando React detecta que los datos de un componente han cambiado, invoca el método `render` para actualizar el estado y así modificar `state` a su nuevo valor.

Como se puede observar en la Ilustración 6, es habitual el uso de expresiones JSX que, en este caso, se está empleando para representar la etiqueta `<div>`. Al igual que ocurre en HTML es posible la anidación de estas etiquetas.

El uso de JSX [11], aunque es lo más extendido, no es obligatorio. Otro mecanismo para crear este tipo de etiquetas sin utilizar JSX es empleando el método `createElement()`, el cual recibe como parámetro el tipo de elemento que se desea crear. En efecto, cuando empleamos expresiones JSX, al compilar el código React convierte todas estas expresiones en llamadas a `createElement()`. En la Ilustración 7 podemos ver un ejemplo de ello, donde ambas formas son equivalentes.

```
class Ejemplo extends React.Component{
    render(){
        return <div>Hola</div>;
    }
}

class OtroEjemplo extends React.Component{
    render(){
        return React.createElement('div',null,'Hola');
    }
}
```

*Ilustración 7 – Uso de JSX y createElement()*

### 2.2.1 Ciclo de vida de una aplicación

Uno de los conceptos más importantes de React es el de ciclo de vida de una aplicación [12]. Los componentes deben liberar recursos cuando no están siendo utilizados, y para conseguirlo, React incorpora varios métodos, de los cuales los más utilizados son los tres que se explican a continuación.

#### 2.2.1.1 componentDidMount

`componentDidMount()` [13] es un método de React que se invoca automáticamente cuando una nueva instancia de un componente se crea y se inserta en el DOM[2]. Es buena práctica insertar en el cuerpo de este método cualquier llamada para obtener datos del servidor y almacenarlos en el estado interno mediante el uso del método `setState()`. En la siguiente ilustración se puede ver un ejemplo de su uso.

```
componentDidMount(){
    //Llamada para obtener datos del servidor
    this.setState({datos}); //Almacenar esos datos
}
```

*Ilustración 8 - Ejemplo de componentDidMount()*

#### 2.2.1.2 componentDidUpdate

`componentDidUpdate()` [14] se invoca automáticamente cuando cambia el estado o alguna propiedad de un componente. Es habitual usarlo cuando queremos comparar si el estado o la propiedad previa de un componente ha cambiado a un nuevo valor, para, por ejemplo, solicitar nuevos datos del servidor. En la siguiente ilustración se puede ver un ejemplo de uso.

[2] DOM son las siglas de Document Object Model

```
componentDidUpdate(propPrevio){
    if(propPrevio.value !== this.props.value){
        //Llamada al servidor para obtener
        //nuevos datos
    }
}
```

*Ilustración 9 - Ejemplo de componentDidUpdate()*

#### 2.2.1.3 componentWillUnmount

`componentWillUnmount()` [15] se invoca automáticamente cuando un componente se elimina del DOM. Es buena práctica utilizar este método para cancelar la suscripción a cualquier evento previo, por ejemplo, un Listener, y así liberar memoria. A continuación, se puede ver un ejemplo.

```
componentWillUnmount(){
    //Eliminar cualquier temporizador
    //Cancelar suscripciones
    document.removeListener() //Eliminar Listeners

}
```

*Ilustración 10 - Ejemplo de componentWillUnmount()*

# 3 REACT NATIVE

*La innovación diferencia a un líder de un seguidor*

*Steve Jobs*

Una vez conocidas las tecnologías que conforman las bases de React Native, en este apartado se va a hacer una descripción de React Native y se explicarán los conceptos más importantes para el desarrollo de aplicaciones móviles usando esta tecnología.

## 3.1 Introducción a React Native

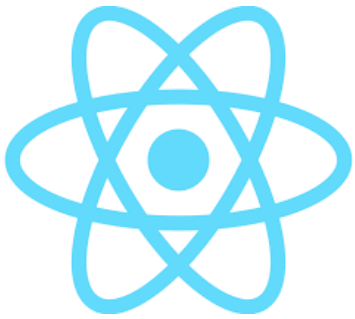

***Ilustración 11 - Logo de React Native***

React Native [16][17] es un marco de desarrollo de software creado por Facebook[3] para la creación de aplicaciones móviles multiplataforma. Esta tecnología nos permite crear una única aplicación que es capaz de ejecutarse en diferentes sistemas, principalmente para dispositivos iOS y Android, aunque también es válido para dispositivos con Android TV, tvOS e incluso aplicaciones web.

Está basado en React, una librería de JavaScript para la creación de interfaces de usuario comentada en el apartado anterior, pero en lugar de usar componentes web, usa componentes nativos.

React Native crea en tiempo de ejecución las correspondientes vistas de Android y iOS para los componentes

[3] Facebook es actualmente conocido como Meta Platforms Inc.

que se están renderizando. Por lo que el resultado final será una aplicación real nativa para el propio dispositivo en el que se esté ejecutando. Como si hubiese sido desarrollada en Java o Kotlin para el caso de Android, o, Swift o Objective-C para el caso de iOS.

La elección de React Native como marco de desarrollo para una aplicación móvil trae consigo numerosas ventajas. En primer lugar, al tratarse de desarrollo multiplataforma solo es necesario escribir código una vez, por lo que su uso ahorra tiempo y dinero comparado con el desarrollo nativo para Android y iOS, donde sería necesario un equipo de desarrollo para cada plataforma.

Otra de las ventajas de su uso es el empleo de JavaScript como lenguaje de programación, ya que su gran popularidad lo convierten en uno de los lenguajes más utilizados para el desarrollo. Esto hace que tenga una gran comunidad de desarrolladores y la existencia de cientas de librerías que se pueden usar con React Native.

## 3.2 Definición de Componentes: Clases vs Funciones

En React Native, es posible la definición de componentes haciendo uso tanto de clases como de funciones. A continuación, se explica en qué consiste cada uno de ellos.

### 3.2.1 Clases

La definición de componentes mediante clases es muy similar a la forma de hacerlo en React. Se tiene una clase que hereda de la clase `Component`, con el método `render()` que será el encargado de mostrar el resultado en pantalla. A continuación, se muestra un ejemplo.

```
import React, { Component } from 'react';
import { View } from 'react-native';


class Ejemplo extends Component{
    render(){
        return(
                <View></View>
            )
    }
}

export default Ejemplo;
```

*Ilustración 12 - Ejemplo de clase en React Native*

Como se puede observar en la Ilustración 12, el contenedor utilizado para albergar el contenido a mostrar en pantalla es un componente de tipo `<view>`, a diferencia de React, donde se utiliza componentes `<div>`.

El componente `<view>` es fundamental en la construcción de interfaces de usuario. Se mapea directamente al componente nativo propio para el dispositivo en el que se esté ejecutando. Por tanto, en Android se mapeará a un componente `android.view`, mientras que en iOS el equivalente será `UIView`.

La definición de componentes mediante clases implica mayor número de líneas de código, por lo que es preferible la definición mediante funciones, que es la forma por la que se ha optado en este proyecto.

### 3.2.2 Funciones

La definición de componentes mediante funciones, como se ha comentado, es más simple.

Los componentes serán funciones que pueden recibir parámetros y tendrán el método `return` que devolverá el contenido a mostrar en pantalla. Nótese que de esta forma se prescinde del método `render()`. La definición de las funciones es habitual hacerlas empleando las denominadas *funciones flecha*. En la siguiente Ilustración se muestra un ejemplo.

```
import React  from 'react';
import { View } from 'react-native';


const Ejemplo = () => {
    return(
        <View></View>
        )
}
```

***Ilustración 13 - Ejemplo de función en React Native***

## 3.3 Propiedades y parámetros de un componente

Los componentes tienen propiedades que permiten modificar su comportamiento. Estas propiedades varían de un componente a otro dependiendo de su naturaleza, por lo que no todas están disponibles para todos los componentes.

En el desarrollo de la aplicación que se presenta en esta memoria se han utilizado varios componentes propios de React Native, los cuales se explicarán en detalle en el apartado 5 conforme vayan apareciendo.

A continuación, se muestran a modo de ejemplo algunas propiedades del componente `Text`. Todas las propiedades que se pueden aplicar a un componente se pueden consultar en la página web de React Native, siguiendo el enlace: https://reactnative.dev/docs/components-and-apis

#### Algunas propiedades del componente Text

Como la propia palabra indica, `Text` [18] es un componente utilizado para mostrar un texto en pantalla. Cuenta con una larga lista de propiedades, entre las cuales se destacan las siguientes:

- `numberOfLines`: esta propiedad se utiliza para limitar el número de líneas que puede ocupar un texto en pantalla. De modo que, si ocupa más de las definidas, el texto se trunca.
- `onPress`: es una de las propiedades más utilizadas, y dota a un componente de la capacidad de ser pulsado, y, por tanto, convertirse en un botón, el cual se podría emplear para navegar a otra pantalla de la aplicación.
- `selectable`: esta propiedad permite que un texto pueda ser seleccionado para, por ejemplo, usar las funciones de copiar y pegar.

- `style`: como se ha comentado anteriormente, esta propiedad es usada para aplicarle un estilo al componente.

También es posible la creación de nuevas propiedades para pasar datos de interés (parámetros) a un componente, tal y como se muestra en la Ilustración 14, donde se está creando un componente `Ejemplo` que permite pasarle como parámetro un nombre. Posteriormente, en el componente `Prueba` se hace uso de esa propiedad del componente `Ejemplo` para mostrar un saludo en pantalla.

```
import React from 'react';
import { Text, View } from 'react-native';

const Ejemplo = ({nombre}) => {
  return (
    <View>
      <Text>Hola, {nombre}</Text>
    </View>
  );
}

const Prueba = () => {
  return (
    <View>
      <Ejemplo nombre="Pablo" />
    </View>
  );
}
```

***Ilustración 14 - Paso de parámetros y propiedades de un componente***

### 3.1.1 Componentes utilizados

En este subapartado se van a detallar los componentes de React Native utilizados en el desarrollo de la aplicación.

#### 3.1.1.1 ScrollView

El componente `ScrollView` [37] es un componente de React Native con el que podemos desplazarnos hacia arriba y hacia abajo por una pantalla, permitiendo así, visualizar más contenido que el que admiten los límites de la pantalla del dispositivo.

A modo de ejemplo, como se observa en la Ilustración 15, será posible hacer *scroll* por todo aquello que esté contenido dentro de este componente.

```
<ScrollView>
    <Text>
        Aquí podremos poner un texto muy largo
        y deslizar para verlo completo....
    </Text>
</ScrollView>
```

*Ilustración 15 - Ejemplo de ScrollView*

#### 3.1.1.2 TouchableOpacity

Para implementar los botones presentes en la aplicación se ha utilizado el componente `TouchableOpacity` [38] de React Native, el cual muestra una pequeña opacidad cuando se pulsa el botón. En la Ilustración 16 se ejemplifica este procedimiento.

```
<TouchableOpacity onPress={}>
    <Text>
        Esto es un botón
    </Text>
</TouchableOpacity>
```

*Ilustración 16 - Ejemplo de TouchableOpacity*

#### 3.1.1.3 Swiper

El componente `Swiper` [39] se encuentra en una librería externa llamada `react-native-swiper`. Con este componente, es posible deslizar a derecha o izquierda un conjunto de elementos, en este caso, se ha usado para mostrar imágenes. En la siguiente ilustración se muestra un ejemplo de su uso.

```
import Swiper from "react-native-swiper"

<Swiper>
    <Image />
    <Image />
    <Image />
</Swiper>
```

*Ilustración 17 - Ejemplo de Swiper*

Este componente además tiene la opción de poder activar el índice de la imagen que se está mostrando. En este caso, se han usado unos puntos de color naranja para indicar la imagen activa en pantalla.

#### 3.1.1.4 Linking

El método `openURL()` del componente `Linking` de React Native permite interactuar con una URL para abrirla en el navegador, además de poder utilizarse también para abrir algunas aplicaciones instaladas en el dispositivo.

El mecanismo para abrir una URL en el navegador es el que se muestra en la siguiente Ilustración.

```
Linking.openURL("https://www.lemonvil.com");
```

*Ilustración 18 - Ejemplo de Linking*

Este método también es capaz de abrir las aplicaciones de teléfono, mail y sms tan solo con usar los prefijos "tel", "mailto" y "sms" respectivamente. En el caso de la ilustración, se está abriendo la aplicación del teléfono marcando automáticamente el número que aparece.

```
Linking.openURL(`tel:${900804412}`);
```

*Ilustración 19 - Ejemplo de Linking.openURL('tel')*

#### 3.1.1.5 FlatList

`FlatList` [41] es un componente que permite renderizar un conjunto de datos y mostrarlos en pantalla a modo de lista. Es bastante configurable y tiene una larga lista de propiedades que se pueden consultar siguiendo el enlace: https://reactnative.dev/docs/flatlist

Este componente se ha utilizado para crear todas las listas de la aplicación. En todos los casos se han utilizado las tres propiedades que se muestran en la Ilustración 20.

```
<FlatList
    data={facturas}
    keyExtractor={factura => factura.num_factura}
    renderItem={({item}) => <FacturaComponent />}
/>
```

*Ilustración 20 - Ejemplo de FlatList*

- `data`: esta propiedad permite indicar el conjunto de datos que se va a mostrar en pantalla. En esta ocasión, se corresponde con el listado de facturas que devuelve el servicio correspondiente del *backend* de la aplicación.
- `keyExtractor`: esta propiedad se usa para extraer una clave única de cada elemento del conjunto

de datos que se pasan a la propiedad `data`. En esta ocasión, se ha elegido el número de la factura como identificador unívoco.

- `renderItem`: esta propiedad permite indicar el componente que va a renderizar los datos en pantalla. En esta ocasión, se ha creado un componente específico para ello, llamado `FacturaComponent`, el cual extrae de cada item el número de factura, la fecha y el importe para presentarlos en pantalla con el aspecto que se observa en la Ilustración 20.

#### 3.1.1.6 React Native Chart Kit

React Native Chart Kit [44] es una librería de terceros desarrollada para React Native que permite dibujar multitud de gráficos y diagramas. Todos ellos son totalmente personalizables y además son compatibles con Expo.

Toda la documentación relativa a esta librería se puede consultar en el siguiente enlace: https://www.npmjs.com/package/react-native-chart-kit

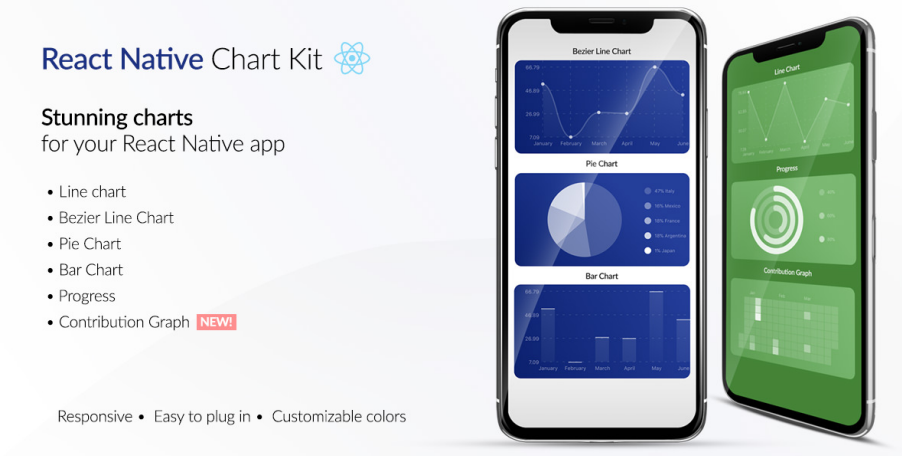


***Ilustración 21 - React Native Chart Kit***

El componente `BarChart` permite, a partir de un conjunto ordenado de datos, dibujar un diagrama de barras. Cuenta con una serie de propiedades que permiten modificar el aspecto de la gráfica, tales como las dimensiones, las etiquetas de los ejes, la rotación de las etiquetas, así como una propiedad que recibe un objeto de configuración de estilos, con el cual se pueden personalizar los colores tanto de las barras como del fondo de la gráfica.

La propiedad `data` se encarga de recibir los datos que se van a mostrar en la gráfica. Estos datos hay que pasarlos en forma de objeto con dos propiedades: `labels` y `datasets`.

La propiedad `labels` permite indicar las etiquetas que acompañarán a cada una de las barras del diagrama, y con la propiedad `datasets` se indicarán los valores numéricos asignados a cada una de las barras. Todos estos datos deben estar ordenados y corresponderse entre ambas propiedades. A continuación, se muestra un ejemplo de ello con los meses del año y el número de días de cada uno.

```
const data = {
    labels: ["Enero", "Febrero", "Marzo"],
    datasets: [
        {
            data: [31, 28, 31]
        }
    ]
};
```

*Ilustración 22 - Ejemplo de datos del componente BarChart*

El componente `ProgressChart` que permite mostrar una gráfica de tipo *Progress Ring* o anillo de progreso que se va rellenando conforme aumenta el valor del dato que se le pasa.

Este componente tiene varias propiedades que permiten modificar el aspecto de la gráfica como el grosor y el radio del anillo, sus dimensiones y la leyenda. La propiedad `data` es a la que se le pasan los datos que se quieren mostrar. Estos datos deben ser de tipo objeto con el valor a representar en tanto por uno, por lo que se han realizado los cálculos necesarios para que los megas consumidos se muestren de forma correcta. A continuación, se muestra un ejemplo de este objeto.

```
const data = {
  data: [0.075] //Representa el 7.5% del consumo
};
```

*Ilustración 23 - Ejemplo de datos en ProgressChart*

#### 3.1.1.7 Lottie

Lottie [45] es una librería compatible con React Native y Expo que permite añadir animaciones a las aplicaciones. Todas las animaciones están creadas con *Adobe After Effects* y son exportadas en formato JSON para poder ser usadas en el desarrollo de una aplicación.

El componente `LottieView` [46] de esta librería es el que permite cargar las animaciones en la aplicación. Este componente cuenta con las siguientes propiedades:

- `source`: esta propiedad permite indicar la ruta donde se encuentra la animación que se quiere usar.
- `loop`: esta propiedad booleana permite indicar si se quiere que la animación se reproduzca en bucle.
- `autoPlay`: esta propiedad booleana permite indicar si se quiere que la animación se reproduzca automáticamente cuando se carga.

En la siguiente ilustración se muestra un ejemplo de uso de este componente.

```
import LottieView from 'lottie-react-native';

<LottieView source={require('./animation.json')} autoPlay loop />
```

*Ilustración 24 - Ejemplo de LottieView*

Las animaciones presentes en la aplicación se han obtenido de la web *Lottie Files* la cual se puede consultar en el siguiente enlace: https://lottiefiles.com/featured

***Ilustración 25 - Logo de LottieFiles***

En esta web podemos encontrar cientas de animaciones creadas por diseñadores para todo tipo de ámbitos. Cuenta con un buscador donde se puede filtrar por una temática de animaciones en concreto, por ejemplo, animaciones de carga, o *loading*.

Todas las animaciones son totalmente personalizables ya que la forma de editarlas funciona por capas, pudiendo, por tanto, modificar el color de cada elemento de una animación. Además, es posible modificar otros aspectos como la velocidad de la animación y sus dimensiones.

Una vez se ha elegido una animación y se ha personalizado como convenga, se descarga en formato JSON y este fichero se incluye, normalmente, en el directorio donde estén almacenadas las imágenes que usa la aplicación, para posteriormente, usarlo con el componente `LottieView`.

Como recurso adicional, existe un *plugin* para VSCode que permite agilizar todo este proceso. Con este *plugin* es posible modificar desde el propio editor de código los colores de la animación, los fotogramas por segundo, sus dimensiones, así como buscar una nueva animación. Se puede encontrar en el siguiente enlace: https://lottiefiles.com/plugins/visual-studio-code

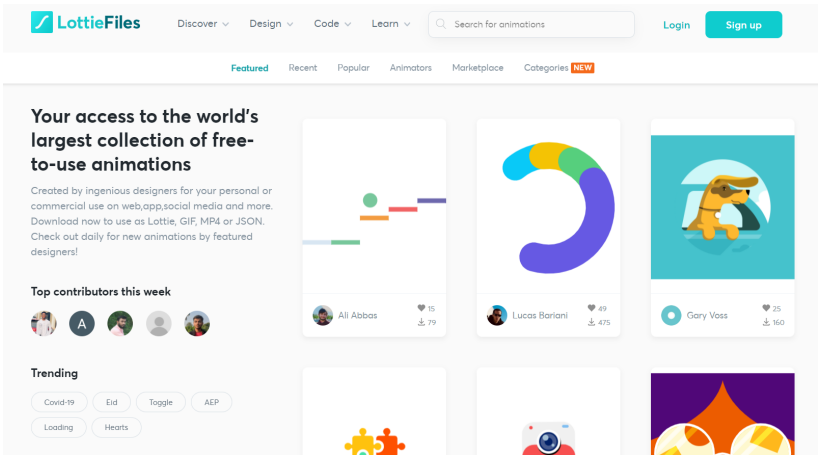


***Ilustración 27 - Web de LottieFiles***

#### 3.1.1.8 Modal

El componente `Modal` de React Native permite mostrar contenido sobre una vista ya existente. Tiene varias propiedades entre las que se destacan las que aparecen en la Ilustración 28.

- `animationType`: con esta propiedad se puede controlar la animación del componente cuando aparece y desaparece de la pantalla. En el caso de la Ilustración 28, fijando el valor a `slide` el componente aparecerá deslizándose desde la parte inferior de la pantalla.
- `transparent`: permite controlar si el fondo del componente será transparente o no.
- `visible`: esta propiedad determina si el componente está visible en pantalla o no. Para ello, se suele usar junto a una variable de estado -llamada `modalVisible` en el caso de la Ilustración 28- que estará fijada a `false` inicialmente. Se pondrá a `true` cuando se muestre este componente en pantalla y, finalmente, haciendo uso de la propiedad `onRequestClose` o mediante el uso de un botón junto a la propiedad `onPress`, se pondrá nuevamente a `false` para ocultar el componente.
- `onRequestClose`: esta propiedad permite ejecutar un fragmento de código cuando se cierra el componente. En el caso de la Ilustración 28 se fija la variable de estado a `false` para ocultar el componente en pantalla.

Todas estas propiedades, junto a otras más, se pueden consultar en la web de React Native siguiendo el enlace: https://reactnative.dev/docs/modal

```
const [modalVisible, setModalVisible] = useState(false);

<Modal
    animationType="slide"
    transparent={true}
    visible={modalVisible}
    onRequestClose={() => {
        setModalVisible(!modalVisible);
    }}
>
```

*Ilustración 28 - Fragmento del componente Modal*

#### 3.1.1.9 Formik


*Ilustración 29 - Logo de Formik*

Formik [33] es una librería para la creación de formularios con React y React Native muy ligera y fácil de usar. Se integra muy bien con Yup, otra librería para la validación de formularios de la cual se hablará posteriormente.

Como se observa en la Ilustración 30, se suele emplear con tres propiedades.

La primera de ellas, `initialValues`, recibe los campos de los que se compone el formulario y su valor inicial. En este caso, los campos son *usuario* y *contraseña* y como valor inicial se ha dejado una cadena vacía para ponerlos en blanco.

La segunda propiedad, `onSubmit`, sirve para indicar la función que se debe ejecutar una vez se han rellenado los campos y se desean enviar al servidor. En esta función se hace la llamada al servicio correspondiente de la API pasándole los valores de los campos rellenos del formulario para hacer el inicio de sesión.

Por último, la propiedad `validationSchema` se usa para indicar qué esquema de validación de formularios se quiere usar. Esta propiedad ayuda a garantizar que los datos que se envían al servidor coincidan con los requisitos establecidos antes de ser enviados. En este caso, el esquema de validación se ha creado con Yup, el cual se explica a continuación.

```
<Formik
  initialValues={initialValues}
  onSubmit={onSubmit}
  validationSchema={validationSchema}
>
```

*Ilustración 30 - Uso de Formik*

#### 3.1.1.10 Yup

Yup [34] es una librería para la validación de formularios con React y React Native que se integra muy bien con Formik. Permite comprobar la estructura de todos los campos del formulario antes de ser enviados al servidor. Como se observa en la Ilustración 31, un esquema de validación con Yup es en realidad un objeto del cual se usa el método `shape()` para indicar la forma de validar el formulario.

En este caso, como se ha comentado anteriormente, los campos que se van a validar son los de usuario y contraseña. Ambos deben ser de tipo *string*, indicado mediante el uso del método `string()`. Deben ser campos requeridos, por lo que no es posible dejarlos en blanco, si eso ocurre, se mostraría en pantalla el mensaje que aparece dentro del método `required()`. Por último, haciendo uso del método `label()` se indica el nombre que se mostrará en el campo del formulario.

```
const validationSchema = Yup.object().shape({
  username: Yup.string().required("El usuario es obligatorio").label("Usuario"),
  password: Yup.string()
    .required("La contraseña es obligatoria")
    .label("Contraseña"),
});
```

*Ilustración 31 - Uso de Yup*

#### 3.1.1.11 ApiSauce

La librería que se ha utilizado para invocar a los servicios del *backend* de la aplicación se trata de *ApiSauce* [55]. Está basada en *Axios* -otra librería muy popular para llamar APIs- con la diferencia de que ApiSauce tiene estandarizados los errores y es capaz de transformar peticiones y respuestas. Todas estas características permiten que cuando se realiza una llamada a un servidor la petición siempre es resuelta, incluso si se obtiene un error. Por tanto, no es necesario el uso de bloques *try/catch* para comprobar la existencia de errores.

ApiSauce cuenta con varios métodos para manejar APIs. El método `create()` es el que se ha utilizado para crear el componente encargado de realizar las peticiones al *backend* difiniendo la dirección IP del mismo mediante el uso de la propiedad `baseURL`. En la siguiente ilustración se muestra un ejemplo de uso. Como se puede observar, también es posible la inclusión de cabeceras.

```
const apiClient = create({
  baseURL: "https://www.lawifi.es:3031/api/v1",
  headers: {
    "Content-Type": "application/json",
  },
});
```

***Ilustración 32 - ApiSauce: uso del método create()***

Una vez definida la dirección IP del *backend*, bastará con hacer uso del objeto que devuelve (*apiClient* en la Ilustración 32) para realizar peticiones del cualquier tipo: *get*, *post*, *put*, etc. En ellas, se indicarán el servicio que se quiere invocar y los parámetros de la petición. A continuación, se muestra un ejemplo.

```
const endpoint = "/tarifalinea";

const getTarifa = (username, linea) =>
  apiClient.post(endpoint, { username, linea });
```

***Ilustración 33 - ApiSauce: uso del método post()***

Al iniciar sesión en la aplicación, el servidor devuelve un *token* de autenticación que deberá incluirse en todas las peticiones que se realicen en adelante. Para llevar a cabo esta tarea, se ha hecho uso del método `addAsyncRequestTransform()`. Como se puede observar en la Ilustración 34, este método permite modificar una petición de forma asíncrona, de manera que, en primer lugar, se comprueba la existencia del *token* en la aplicación y, en caso de que exista, se añade de forma automática a la cabecera *Authentication* del protocolo HTTP siguiendo el esquema *Bearer*.

```
apiClient.addAsyncRequestTransform(async (request) => {
  const authToken = await authStorage.getToken();
  if (!authToken) return;
  request.headers["Authorization"] = "Bearer " + authToken;
});
```

***Ilustración 34 – ApiSauce: uso del método AddAsyncRequestTransform()***

#### 3.1.1.12 JWT Decode

Esta librería permite la decodificación de tokens JWT (JSON Web Token) con codificación Base64 para acceder a su información. Todo lo relativo a JWT se explicará más adelante.

#### 3.1.1.13 useNetInfo

Este *hook* de react Native permite conocer el estado de la conexión a Internet del dispositivo. `useNetInfo` [54] proporciona el método `useNetInfo()` que devuelve un objeto con varias propiedades. En concreto, se han usado las propiedades `type` y `isInternetReachable` las cuales permiten conocer el tipo de conexión (datos móviles o WiFi) y si el dispositivo tiene capacidad de conectarse al punto de acceso, respectivamente.

## 3.4 Aplicación y creación de estilos

La creación de estilos [19] para los componentes en React Native se realiza normalmente haciendo uso de la librería *StyleSheet*.

Todos los componentes propios de React Native aceptan una propiedad llamada *style*, donde podemos incluir directamente los estilos que se deseen, o bien, hacer uso de esta librería y definirlos en otro lugar.

Al hacer uso de la librería *StyleSheet*, se crea un contenedor de estilos donde podemos definir de forma ordenada todos los estilos para los componentes a los que se quiera aplicar. En la Ilustración 35 se ejemplifica este procedimiento.

Los componentes en React Native tienen algunos estilos que son comunes a todos los componentes, y otros, que son propios para un componente en específico. Por ejemplo, en la Ilustración 35, se puede apreciar que el componente `Text` tiene propiedades como `fontWeight` y `fontSize` que cambian el grosor y el tamaño de la letra respectivamente, y estas propiedades no están disponibles, por ejemplo, para el componente `View` puesto que no es un componente propio para definir texto.

```
import React from 'react';
import { StyleSheet, Text, View } from 'react-native';

const EjemploEstilos = () => {
    return (
      <View style={styles.contenedor}>
        <Text style={styles.titulo}>Hola, React Native</Text>
      </View>
    );
};

const styles = StyleSheet.create({
  contenedor: {
    marginTop: 50,
  },
  titulo: {
    color: 'blue',
    fontWeight: 'bold',
    fontSize: 30,
  },
});
```

***Ilustración 35 - Ejemplo de uso de estilos en React Native***

## 3.5 Hooks

Los *Hooks* [20] son un concepto novedoso añadido a React y React Native en el año 2018. Se tratan de funciones que permiten manejar el estado de los componentes junto al ciclo de vida de una aplicación.

En React Native, el ciclo de vida de una aplicación es igual al de React, como se presentó en el punto 2.2.1. Sin embargo, los métodos que se vieron no se pueden utilizar haciendo uso de funciones, solo mediante clases. Es por ello por lo que surge la necesidad de incorporar un nuevo mecanismo para controlar estas labores mediante funciones.

React Native proporciona algunos *hooks* muy útiles, aunque también es posible la creación de nuevos por parte del usuario para agilizar cualquier tarea. Como acuerdo, el nombre de todos los *hooks* deben comenzar con la palabra *use*.

Es importante destacar que las llamadas a los *hooks* solo se pueden realizar en el nivel superior de una función, no siendo posible su llamada desde dentro de funciones regulares de JavaScript ni expresiones condicionales. De esta manera, nos aseguramos de que las llamadas a los *hooks* siempre se realicen en el mismo orden cuando un componente se ejecuta.

A continuación, se presentan tres de los *hooks* más utilizados.

### 3.5.1 useEffect

El *hook* `useEffect()` [21], o hook de efecto, es el equivalente al `componentDidMount()` que se vio en el apartado 2.2.1.1. Permite ejecutar código adicional después de que se haya ejecutado un componente y presentado en pantalla.

Es muy útil para el caso de las peticiones de red como una llamada a una API [4] o a una base de datos. En estas ocasiones, la petición de red se maneja completamente desde el cuerpo de esta función, para una vez obtenida la respuesta, pasar los datos a los componentes para, por ejemplo, que lo muestren en pantalla. En la Ilustración 36 se muestra un ejemplo.

```
useEffect(() => {
    llamada_api();
  }, []);

const llamada_api = async () => {
    //cuerpo de la funcion
}
```

***Ilustración 36 - Ejemplo de useEffect()***

Como se puede observar, dentro del cuerpo de `useEffect()` se realiza la llamada a una función asíncrona que se encarga de realizar la petición de red. Esto es así puesto que no es posible utilizar directamente el asincronismo dentro del cuerpo del *hook*, por lo que hay que recurrir a este procedimiento.

`useEffect()` admite como segundo parámetro un vector de opciones que permiten modificar algunos

[4] API son las siglas de *Application Programming Interface*, o Interfaz de Programación de Aplicaciones

aspectos de su comportamiento. En el caso de la ilustración anterior, ese vector se ha dejado vacío mediante el uso de corchetes `[]`, lo cual indica a React Native que la ejecución del *hook* solo se realizará una vez.

### 3.5.2 useState

`useState()` [22], o *hook* de estado, es otro de los más utilizados. Permite añadir el estado de React Native a un componente de función. Por lo que si necesitamos que un componente tenga una variable que no se elimine tras la ejecución, usaremos este *hook* ya que el valor quedará almacenado en el estado de React Native.

Admite un solo parámetro que indica el estado inicial de la variable que se está creando. En la siguiente ilustración se puede observar su uso.

```
const [contador, setContador] = useState(0);
```

***Ilustración 37 - Ejemplo de useState()***

Como se puede observar en la Ilustración 37, `useState()` devuelve un vector con una pareja de valores. El primer valor hace referencia al estado actual, que, en este caso, es una variable `contador` inicializada a cero. El segundo valor, `setContador` es una función que permite modificar el estado de la variable posteriormente.

En resumen, haciendo uso de este *hook* podemos conservar el valor actual de una variable entre renderizados de componentes.

### 3.5.3 useContext

`useContext()` [23], o hook de contexto, es un *hook* que como la palabra indica, permite manejar contextos. Recibe como parámetro un objeto de contexto y devuelve el valor del contexto actual. Los objetos de contexto son creados haciendo uso del método `createContext()` de React. En la siguiente ilustración se muestra un ejemplo.

```
const Contexto = React.createContext();

const valor = useContext(Contexto);
```

***Ilustración 38 - Ejemplo de useContext()***

Este *hook* es útil para guardar el contexto cuando, por ejemplo, se tiene una aplicación móvil y se desea mantener la sesión de un usuario abierta durante todas las pantallas que compongan la aplicación.

# 4 EXPO

*La función de un buen software es hacer que lo complejo aparente ser simple*

*Grady Booch*

En este apartado, se va a presentar otra de las tecnologías fundamentales empleadas en el desarrollo de la aplicación. Se trata de Expo, el cliente más popular para el desarrollo de aplicaciones móviles con React Native.

## 4.1 Introducción a Expo


***Ilustración 39 - Logo de Expo***

Actualmente existen dos formas de desarrollar aplicaciones con React Native: la primera de ellas es usando el propio cliente que proporciona React Native, llamado *React Native CLI* [5], y la segunda, usando el cliente que proporciona Expo, llamado *Expo CLI*.

Expo [24] es un *framework* para el desarrollo de aplicaciones móviles con React Native que proporciona un conjunto de herramientas que facilitan algunas tareas en el desarrollo, compilación, depuración y despliegue de las mismas, ocultando gran parte de la complejidad de algunas tareas.

Cuando usamos Expo como cliente, encontramos algunos inconvenientes como el no poder acceder a las APIs

[5] *CLI* son las siglas de Command Line Interface

nativas de iOS y Android, estando limitados a las herramientas que proporciona el SDK[6] de Expo. Sin embargo, esto no debería ser un motivo para dejar de usar Expo, ya que proporciona gran cantidad de características nativas que permiten desarrollar aplicaciones complejas.

Una de las principales ventajas a la hora de utilizar el cliente de Expo es que permite lo que se denomina *hot reloading*. Esta característica nos permite refrescar instantáneamente la aplicación durante su desarrollo, ya sea en un dispositivo físico o un simulador, sin necesidad de hacerlo manualmente. Basta con guardar los cambios en el fichero con el que se esté trabajando para que aparezcan reflejados automáticamente en el dispositivo, lo cual agiliza bastante las tareas de testeo de una aplicación.

Todo lo relacionado a la instalación de Expo y a la creación de una nueva aplicación se encuentra explicado en el Anexo A de esta memoria.

## 4.2 Flujos de Trabajo: Managed Workflow vs Bare Workflow

Existen dos formas de desarrollar aplicaciones usando las herramientas de Expo. Son los denominados flujos de trabajo, o *workflow* en inglés. A continuación, se explica en qué consisten y las diferencias entre ellos.

### 4.2.1 Managed Workflow

Con el *Managed Workflow* [25], o flujo de trabajo gestionado, el desarrollador solo se limita a escribir código JavaScript que se ejecuta tanto en Android como en iOS, sin tener que acceder a las APIs nativas de estas plataformas. Por lo que en el caso de necesitar acceder a la cámara o al sistema de archivos del dispositivo, se emplean las librerías propias de SDK de Expo.

Es importante destacar que es posible dejar de usar Expo en cualquier momento. Una aplicación que esté siendo desarrollada empleando el SDK de Expo puede dejar de utilizar sus librerías cuando se quiera, por lo que tendremos una aplicación nativa haciendo uso de las librerías de React Native que se hayan instalado.

Como se ha comentado anteriormente, el SDK de Expo ofrece gran cantidad de librerías que permiten el desarrollo de aplicaciones complejas, por lo que este flujo de trabajo ha sido el elegido para el desarrollo de la aplicación que se va a presentar en el apartado 5.

### 4.2.2 Bare Workflow

Con el *Bare Workflow* [26], o flujo de trabajo simple, el desarrollador tiene control total de la aplicación. Es posible el uso de librerías propias de Android y iOS, aunque también algunas presentes en el SDK de Expo. En este caso, el desarrollo de la aplicación puede ser más complejo que utilizando el *Managed Workflow*, ya que entre los ficheros que componen una aplicación, encontraremos dos nuevos directorios que no estaban presentes usando el otro flujo de trabajo. Estos directorios son los específicos para Android y iOS y contienen código nativo para esa plataforma, por lo que es posible modificar, por ejemplo, en el caso de Android, el fichero `AndroidManifest.xml`. Por estos motivos, es posible el desarrollo de la aplicación en entornos como Android Studio y Xcode.

Es recomendable el uso de este flujo de trabajo cuando se necesite acceder a alguna librería o característica nativa de un dispositivo y no esté presente en el SDK de Expo, de lo contrario, por su sencillez, siempre es preferible usar el flujo de trabajo gestionado.

[6] SDK son las siglas de Software Development Kit

## 4.3 Componentes de Expo utilizados

A continuación, se van a detallar los componentes de Expo utilizados en el desarrollo de la aplicación. Todos estos componentes y la manera en la que se usan se pueden consultar en la web oficial de Expo siguiendo el enlace: https://docs.expo.dev/versions/latest/

### 4.3.1 ImagePicker

El componente `ImagePicker` [47] de la librería de Expo `expo-image-picker`, como su nombre indica, permite seleccionar una imagen de la galería de fotos del dispositivo. Para ello, se hace uso del método `requestMediaLibraryPermissionsAsync()` con el que se le pregunta al usuario si desea conceder permisos para acceder a la galería de fotos. En caso de que se le conceda dicho permiso, se hace uso del método `launchImageLibraryAsync()` para lanzar la galería y que el usuario pueda seleccionar la imagen de perfil que desee.

En el caso de que no se le conceda permiso para acceder a la galería, no será posible seleccionar ninguna imagen a menos que posteriormente se conceda dicho permiso de forma manual desde los ajustes del dispositivo, habilitándolo para esta aplicación.

### 4.3.2 Vector Icons

Esta librería permite acceder a multitud de componentes que manejan iconos vectorizados completamente personalizables. Las principales librerías usadas en este proyecto son *MaterialCommunityIcons* y *FontAwesome*.

### 4.3.3 Secure Store

Esta librería proporciona un almacenamiento seguro y encriptado para almacenar parejas de clave-valor de forma segura en el dispositivo. Durante el desarrollo de la aplicación se ha usado para almacenar datos como el token de autenticación o las credenciales de un usuario.

### 4.3.4 Push Notifications

Las *notificaciones push* permiten enviar mensajes desde un servidor remoto hasta los dispositivos que tienen la aplicación instalada. Se entregan de forma inmediata, por lo que no es necesario tener la aplicación abierta para poder recibirlas y, además, son independientes del dispositivo que se esté usando, ya sea iOS o Android. El abanico de posibilidades de este tipo de notificaciones es muy amplio, permitiendo avisar automáticamente en el caso de una aplicación de mensajería instantánea o de alertar de alguna noticia en el caso de otras aplicaciones, creando así una mejor experiencia de usuario.

Existen varios mecanismos para implementar este tipo de notificaciones en una aplicación. Los más usados actualmente son *Firebase Cloud Messaging* de Google [49], *Amazon Simple Notification Service* [50] (Amazon SNS) o el propio sistema de notificaciones push de Expo [51] en el caso de usar React Native.

Para la aplicación que se está desarrollando se ha elegido este último, ya que al usar React Native con Expo, el proceso de desarrollo de este tipo de notificaciones se simplifica bastante comparado con los otros servicios comentados. Tiene total compatibilidad con cualquier dispositivo físico, sin embargo, no es posible enviar

notificaciones a un simulador, ya sea de Android o de iOS.

El sistema de notificaciones push de Expo suscribe a un usuario que inicia sesión en la aplicación a este tipo de notificaciones. Para ello, en primer lugar, se debe pedir permiso al usuario de poder utilizar las notificaciones en la aplicación haciendo uso del método `getPermissionsAsync()` de la librería de notificaciones de Expo llamada `expo-notifications` [52]. En el caso de autorizarlas, a continuación se hace uso del método `getExpoPushTokenAsync()` que genera un token que identifica de forma unívoca a un terminal. En la siguiente ilustración se ejemplifica este procedimiento.

```
const permission = await Notifications.getPermissionsAsync();
if (!permission.granted) return;

const token = await Notifications.getExpoPushTokenAsync();
```

***Ilustración 40 - Ejemplo de getExpoPushTokenAsync()***

Una vez se ha obtenido el token, podrá utilizarse para enviar notificaciones a ese terminal en concreto. Una forma de proceder a ello es usando la propia plataforma online de Expo llamada "Expo push notifications tool" [53], la cual se puede consultar en el siguiente enlace: https://expo.dev/notifications

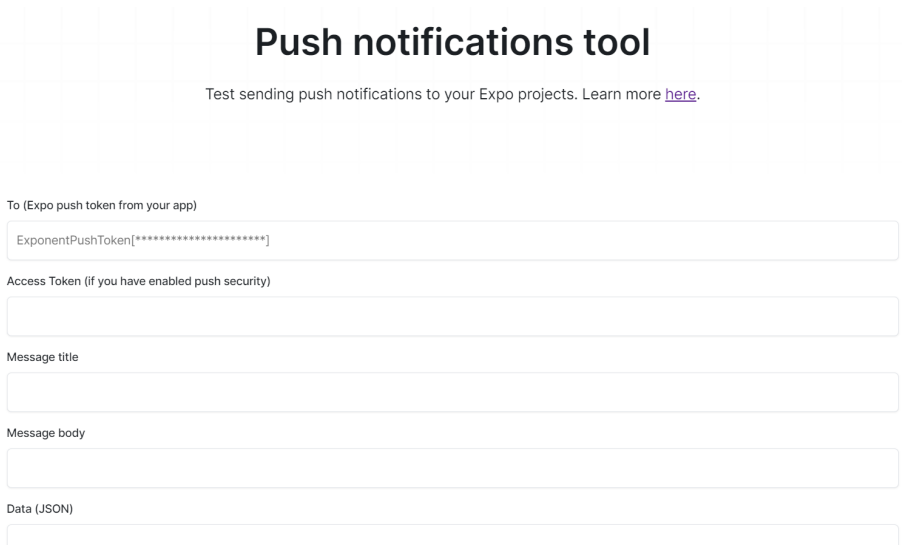


***Ilustración 41 - Expo Push Notification Tool***

Como se aprecia en la Ilustración 41, la web consta de un formulario donde es posible configurar ciertos aspectos acerca del envío de una notificación.

En primer lugar, será necesario indicar el token obtenido previamente en el campo *"To (Expo push token from your app)"*. Para indicar el título de la notificación se usará el campo *"Message Title"* y en el campo *"Message Body"* se podrá escribir el texto que se desee notificar al usuario.

Existe otro campo llamado *"Data (JSON)"* que permite enviar datos en formato JSON para controlar algún aspecto de la notificación cuando se recibe en el dispositivo. En este caso, se ha añadido la propiedad `{"_displayInForeground": true}` para hacer que la notificación sea visible sin necesidad de tener la aplicación abierta en el dispositivo destino.

Una vez que se han rellenado los datos en el formulario, se procederá al envío de la notificación, la cual llegará

al *backend* de Expo y desde aquí, se podrán distinguir dos caminos distintos dependiendo de si el dispositivo destino es Android o iOS.

En el caso de que el dispositivo destino ejecute Android, se usará el servicio *Firebase Cloud Messaging* de Google para enviar la notificación y, en el caso de que el dispositivo destino sea iOS, se usará el servicio *Apple Push Notification Service* o APNS de Apple para enviar la notificación.

En la Ilustración 42 se esquematiza todo este procedimiento.

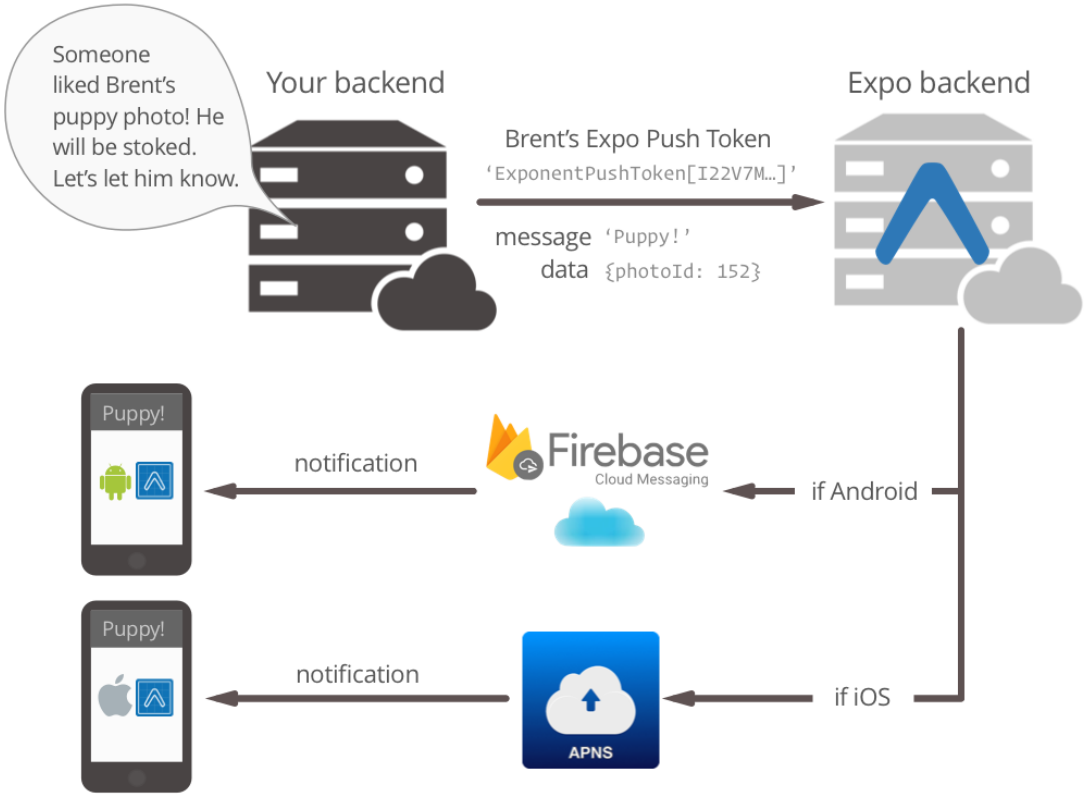


***Ilustración 42 - Funcionamiento de Expo Push Notifications***

# 5 ARQUITECTURA Y APIS USADAS

*Cualquier tecnología suficientemente avanzada es indistinguible de la magia*

*Arthur C. Clarke*

En este apartado, se van a detallar los aspectos relacionados con el desarrollo de la aplicación y del sistema en conjunto. Se comenzará mostrando un diagrama de componentes para tener una visión global de la arquitectura del sistema, para posteriormente, detallar cada una de las partes que componen dicho sistema.

## 5.1 Arquitectura del Sistema

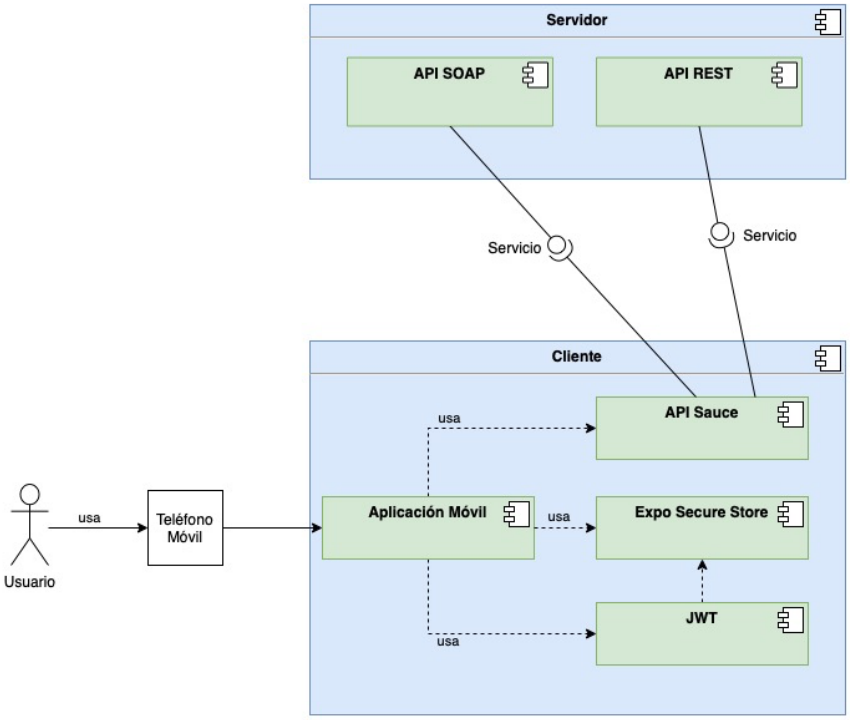


***Ilustración 43 - Diagrama de Componentes: Arquitectura del sistema***

En la Ilustracion 43 se puede observar la arquitectura completa del sistema. Para cada uno de los componentes se ha destacado los elementos más importantes que permiten la interacción entre las distintas partes del sistema. A continuación, se explica cada una de ellas.

- Cliente: el componente principal del cliente se trata de la aplicación móvil desarrollada. Para la interacción con el servidor es fundamental el empleo de:
    - *JWT*: JSON Web Token se trata del componente que permite decodificar el token de autenticación proporcionado por el servidor en el momento de iniciar sesión. Más adelante se detallará su uso y aplicación.
    - *Expo Secure Store*: este componente permite el guardado seguro de los datos sensibles. En este caso se está usando para almacenar el token de autenticación proporcionado por el servidor.
    - *Api Sauce*: este componente permite la interacción del cliente con el servidor. Se encarga de establecer la conexión con el mismo y de invocar el servicio correspondiente para cada una de las APIs presentes en el servidor.
- Servidor: en el servidor se encuentran alojadas las diferentes APIs que se han implementado en el cliente.
    - *API SOAP*: se trata de una API que utiliza como medio de transmisión de la información el protocolo SOAP (Simple Object Access Protocol). Cuenta con una serie de servicios que se invocan desde el cliente por medio del componente ApiSauce. Para la descripción de esos servicios se emplea WSDL junto a XML.
    - *API REST*: se trata de una API que utiliza la transferencia de estado representacional (REST) sobre HTTP para la transmisión de la información y en formato JSON. Al igual que la API SOAP, también cuenta con una serie de servicios que se invocan desde el cliente por medio de ApiSauce.

## 5.2 Aplicación móvil

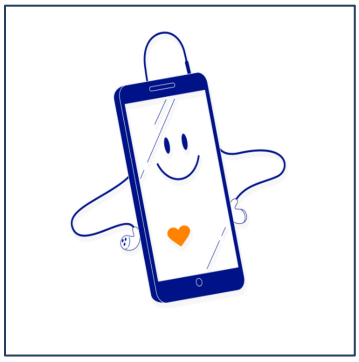

*Ilustración 44 - Logo de la app*

La aplicación móvil que se va a desarrollar permite consultar, entre otros datos, el consumo de los clientes y usuarios de la empresa de telefonía Lemonvil.

Lemonvil es un operador móvil virtual, o OMV, que utiliza la red de ORANGE para ofrecer cobertura 2G, 3G, 4G y 4G+ en todo el territorio español.

Como ya se ha adelantado, la aplicación será multiplataforma, por lo que cualquier usuario de iOS y Android podrá hacer uso de ella sin problemas.

Con ella, se podrá consultar de forma rápida y cómoda todos los siguientes aspectos:

- Líneas contratadas por un cliente
- Facturas de un cliente
- Consulta del consumo de datos
- Consulta del consumo de llamadas
- Consulta del consumo de SMS[7]
- Tarifa contratada
- PUK [8]de la SIM[9]
- ICC [10]de la SIM

Alguna de la información disponible variará dependiendo de si una persona autenticada en la aplicación sea un cliente de Lemonvil, o bien un usuario de una línea en concreto. Esto da lugar a tener dos roles distintos dentro de la aplicación, los cuales se explican a continuación.

### 5.2.1 Roles

Como se acaba de comentar, dentro de la aplicación tendremos dos roles distintos dependiendo de si una persona es cliente o usuario de una línea de Lemonvil.

[7] SMS son las siglas de Short Message Service o Servicio de mensaje corto.
[8] PUK son las siglas de Personal Unlocking Key o Clave de desbloqueo personal.
[9] SIM son las siglas de Subscriber Identity Module o Módulo de identificación de abonado
[10] ICC son las siglas de International Circuit Card ID o Identificador Internacional de la Tarjeta de Circuitos

#### 5.2.1.1 Rol Cliente

Si una persona es el titular de un contrato con Lemonvil asumirá este rol. Todas las personas que sean titular tendrán acceso completo a la aplicación, pudiendo consultar el consumo de todas las líneas que tengan contratada, el importe de las facturas, así como los números PUK e ICC de cada línea contratada.

#### 5.2.1.2 Rol Usuario de Línea

Si una persona no es el titular de una línea y únicamente se limita a usar una de las líneas contratadas por un cliente, aumirá este rol. Todas las personas que sean usuario de una línea en concreto solamente podrán consultar el consumo de la línea que estén utilizando y el PUK e ICC de la SIM, por tanto, no tendrán acceso a consultar las facturas.

### 5.2.2 Estructura de los ficheros

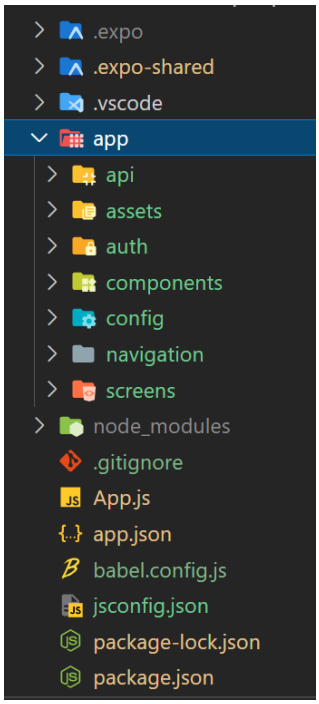


*Ilustración 45 - Estructura de los ficheros*

En este apartado se va a comentar la organización de los ficheros que componen la aplicación.

Como se puede observar en la Ilustración 45, existen varios directorios y algunos ficheros sueltos en el directorio raíz del proyecto. Todos esos ficheros sueltos que se pueden observar en la parte inferior de la ilustración han sido generados automáticamente al crear el proyecto. Únicamente se comentarán dos de ellos:

- **App.js**: este fichero se puede decir que es el principal de la aplicación. Cuando una aplicación se ejecuta, lo que se está cargando realmente es el contenido de este fichero. En aplicaciones más

complejas, desde este fichero se irán llamando al resto de ficheros o bien mediante el uso de navegadores, los cuales se comentarán posteriormente.

- **app.json**: este fichero .json permite controlar algunos parámetros generales de la aplicación entre los que se destacan la elección del logotipo de la aplicación y si se permite el uso de la app en modo horizontal.

Los directorios que se pueden observar en la ilustración también han sido generados automáticamente al crear el proyecto, excepto el directorio *app* y todos sus subdirectorios, los cuales se van a comentar a continuación.

- **app**: este directorio contiene todo el código de la aplicación dividido en subdirectorios para organizar de forma más cómoda los ficheros cuyo objetivo sea similar.
- **api**: este directorio contiene todos los ficheros de código que se encargan de manejar la API que usa la aplicación.
- **assets**: contiene todas las imágenes y animaciones que se usan en la aplicación.
- **auth**: contiene todos los ficheros de código que se encargan de la autorización de un usuario en la aplicación y el guardado de datos sensibles de forma segura.
- **components**: este directorio contiene todos los componentes creados para la aplicación, los cuales se explicarán posteriormente.
- **config**: contiene un fichero de configuración donde se recoge toda la gama de colores empleados en la aplicación.
- **navigation**: contiene todos los ficheros necesarios para crear la navegación por la aplicación.
- **screens**: contiene todos los ficheros que conforman las pantallas en las que se divide la aplicación.

### 5.2.3 Navegadores

En este apartado se van a recoger los navegadores empleados para moverse por la aplicación de una pantalla a otra. Todos los ficheros que componen la navegación están recogidos en el directorio *navigation* mencionado anteriormente.

#### 5.2.3.1 AppNavigator

Este fichero crea el flujo principal de navegación de toda la aplicación. Además, crea la barra de navegación en la parte inferior de la pantalla, tal y como se muestra en la Ilustración 46, que permite moverse entre la pestaña de inicio, la de perfil de usuario y todas las pantallas contenidas en cada una de ellas.


***Ilustración 46 - AppNavigator***

Para la creación de la barra inferior se ha empleado el método `createBottomTabNavigator()` [31] el cual se puede observar en la siguiente ilustración. Este método pertenece a la librería *react-navigation*.

```
const Tab = createBottomTabNavigator();
```

***Ilustración 47 - createBottomTabNavigator()***

Este navegador se usa en el fichero principal App.js y juega un papel fundamental en el flujo de autenticación de un usuario, de forma que, si un usuario se autentica de forma correcta, se hará uso de este navegador y se le permitirá navegar por toda la aplicación.

#### 5.2.3.2 AuthNavigator

Con este fichero se crea la navegación del flujo de autenticación de un usuario en la aplicación. Permite navegar entre las pantallas de bienvenida y login que se mostrarán posteriormente. Para ello se hace uso del método `createStackNavigator()`[32] de la librería *react navigation* el cual crea una pila de navegación entre ambas pantallas. En la Ilustración 48 se muestra un ejemplo de ello.

```
const Stack = createStackNavigator();
```

***Ilustración 48 - createStackNavigator()***

Se hace uso de este fichero de navegación en el flujo de autenticación de un usuario que se encuentra en el fichero principal App.js, de forma que, si un usuario se autentica de forma incorrecta, continuará en este flujo y solo podrá moverse por la pantalla de login y la de bienvenida.

#### 5.2.3.3 ManagementNavigator

Este fichero contiene todo el flujo de navegación de la aplicación referente a la pestaña "Inicio". Permite moverse por todas las pantallas contenidas dentro de esta pestaña. Para ello se ha hecho uso del método `createStackNavigator()` para crear una pila de navegación entre todas las pantallas de esta pestaña.

#### 5.2.3.4 PerfilNavigator

Este fichero contiene todo el flujo de navegación referente a la pestaña "Perfil". Al igual que con el *ManagementNavigator*, se hace uso del método `createStackNavigator()`para crear una pila de navegación con todas las pantallas contenidas en esta pestaña.

## 5.3 API SOAP

En este apartado se va a describir la API SOAP con la que se ha trabajado. Esta API no se está usando actualmente en el cliente ya que ha sido sustituida por la API REST que se presentará posteriormente.

Para la descripción de sus servicios se emplea WSDL (*Web Services Description Language*), por lo tanto, como es habitual, está basado en XML.

### 5.3.1 Casos de uso

En este apartado se van a comentar las distintas funcionalidades que se implementan en la aplicación con esta API. En el siguiente diagrama de casos de uso se resumen todas ellas.

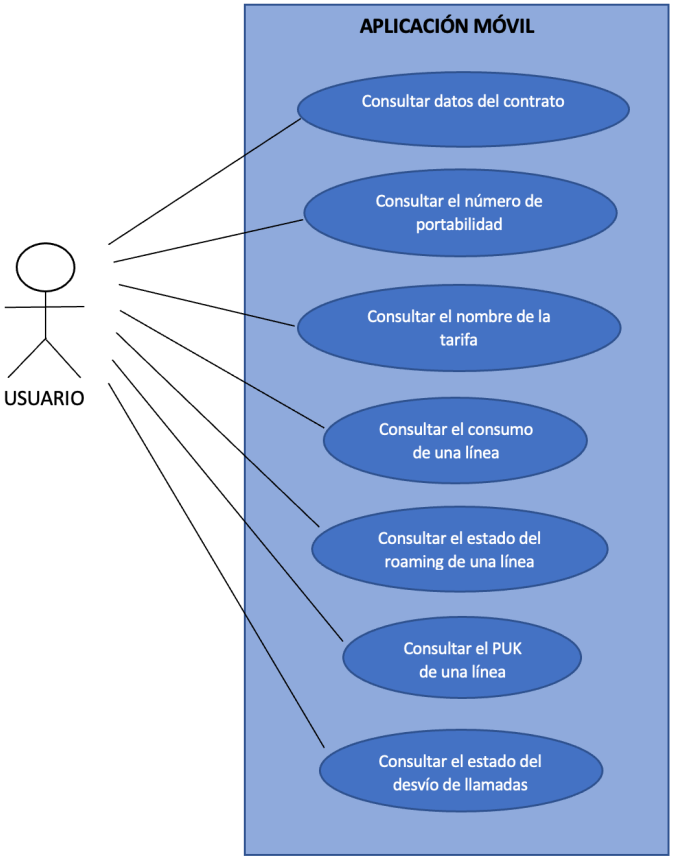


*Ilustración 49 - API SOAP: Diagrama de casos de uso*

| **CU-01** | **Iniciar Sesión en la Aplicación** | |
|---|---|---|
| Precondición | El usuario no se ha autenticado aún en la aplicación. | |
| Descripción | El usuario introduce su nombre de usuario y contraseña en el formulario para acceder a las funcionalidades de la aplicación. | |
| Secuencia | **Paso** | **Acción** |
| | 1 | El usuario introduce su nombre de usuario y contraseña. |
| | 2 | El usuario pulsa el botón de "Iniciar Sesión". |
| | 3 | La API SOAP comprueba que las credenciales son correctas. |
| | 4 | La API SOAP devuelve una sessionID necesaria para la llamada al resto de servicios. |
| Postcondición | El usuario ha iniciado sesión en la aplicación. | |
| Excepciones | **Paso** | **Acción** |
| | 1 | Un campo del formulario se ha dejado en blanco. |
| | | E.1 La aplicación informa al usuario que los campos del formulario son obligatorios. |
| | | E.2 Se cancela el caso de uso. |
| | 2 | El usuario y/o contraseña introducidos son incorrectos. |
| | | E.1 La API SOAP devuelve que los datos introducidos son incorrectos. |
| | | E.2 Se cancela el caso de uso. |

***Tabla 1 – CU-01: Inicio de sesión SOAP***

| **CU-02** | **Extraer consumo de una línea** | | |
|---|---|---|---|
| Precondición | El usuario debe haber iniciado sesión en la aplicación. | | |
| Descripción | El usuario solicita la consulta del consumo de una línea según una fecha. | | |
| Secuencia | **Paso** | **Acción** | |
| | 1 | El usuario accede a la gestión de una línea. | |
| | 2 | Se hace la petición a la API SOAP enviando la sessionID. | |
| | 3 | La API SOAP comprueba que los datos recibidos sean correctos. | |
| | 4 | La API SOAP devuelve el listado con el consumo de la línea. | |
| Postcondición | El usuario puede consultar el consumo de una línea. | | |
| Excepciones | **Paso** | **Acción** | |
| | 1 | La sessionID es incorrecta o ha expirado. | |
| | | E.1 | La aplicación informa al usuario del error ocurrido. |
| | | E.2 | Se cancela el caso de uso. |
| | 2 | Los datos enviados a la API SOAP son incorrectos. | |
| | | E.1 | La API SOAP devuelve un mensaje con el error que se ha producido. |
| | | E.2 | Se cancela el caso de uso. |

***Tabla 2 – CU-02: Extraer consumo de línea SOAP***

| **CU-03** | **Extraer datos de líneas** | |
|---|---|---|
| Precondición | El usuario debe haber iniciado sesión en la aplicación. | |
| Descripción | El usuario solicita la consulta de los datos del contrato. | |
| Secuencia | **Paso** | **Acción** |
| | 1 | El usuario accede a la gestión de una línea. |
| | 2 | Se hace la petición a la API SOAP enviando la sessionID. |
| | 3 | La API SOAP comprueba que los datos recibidos sean correctos. |
| | 4 | La API SOAP devuelve los datos del contrato. |
| Postcondición | El usuario puede consultar los datos del contrato de una o varias líneas. | |
| Excepciones | **Paso** | **Acción** |
| | 1 | La sessionID es incorrecta o ha expirado. |
| | | E.1 La aplicación informa al usuario del error ocurrido. |
| | | E.2 Se cancela el caso de uso. |
| | 2 | Los datos enviados a la API SOAP son incorrectos. |
| | | E.1 La API SOAP devuelve un mensaje con el error que se ha producido. |
| | | E.2 Se cancela el caso de uso. |

***Tabla 3 – CU-03: Extraer datos de línea SOAP***

| **CU-04** | **Consultar el estado del roaming de una línea** | |
|---|---|---|
| Precondición | El usuario debe haber iniciado sesión en la aplicación. | |
| Descripción | El usuario solicita la consulta del estado del roaming de una línea. | |
| Secuencia | **Paso** | **Acción** |
| | 1 | El usuario accede a la gestión de una línea. |
| | 2 | Se hace la petición a la API SOAP enviando la sessionID. |
| | 3 | La API SOAP comprueba que los datos recibidos sean correctos. |
| | 4 | La API SOAP devuelve el estado del roaming de la línea. |
| Postcondición | El usuario puede consultar el estado del roaming para la línea solicitada. | |
| Excepciones | **Paso** | **Acción** |
| | 1 | La sessionID es incorrecta o ha expirado. |
| | | E.1 La aplicación informa al usuario del error ocurrido. |
| | | E.2 Se cancela el caso de uso. |
| | 2 | Los datos enviados a la API SOAP son incorrectos. |
| | | E.1 La API SOAP devuelve un mensaje con el error que se ha producido. |
| | | E.2 Se cancela el caso de uso. |

***Tabla 4 – CU-04: Consultar el estado del roaming SOAP***

| **CU-05** | **Consultar el PUK de una línea** | | |
|---|---|---|---|
| Precondición | El usuario debe haber iniciado sesión en la aplicación. | | |
| Descripción | El usuario solicita la consulta del PUK de una línea. | | |
| Secuencia | **Paso** | **Acción** | |
| | 1 | El usuario accede a la gestión de una línea. | |
| | 2 | Se hace la petición a la API SOAP enviando la sessionID. | |
| | 3 | La API SOAP comprueba que los datos recibidos sean correctos. | |
| | 4 | La API SOAP devuelve los datos del contrato. | |
| Postcondición | El usuario puede consultar el código PUK de la SIM de una línea. | | |
| Excepciones | **Paso** | **Acción** | |
| | 1 | La sessionID es incorrecta o ha expirado. | |
| | | E.1 | La aplicación informa al usuario del error ocurrido. |
| | | E.2 | Se cancela el caso de uso. |
| | 2 | Los datos enviados a la API SOAP son incorrectos. | |
| | | E.1 | La API SOAP devuelve un mensaje con el error que se ha producido. |
| | | E.2 | Se cancela el caso de uso. |

***Tabla 5 – CU-05: consultar el PUK SOAP***

### 5.3.2 Sistema de seguridad: Clave (sessionID)

El mecanismo de seguridad de esta API consiste en el uso de una clave alfanumérica que se genera tras una llamada exitosa al servicio `/comprobarAcceso()`. Este servicio devuelve una clave (sessionID) que se debe enviar como parámetro al resto de servicios de la API. Esta clave, una vez obtenida, se almacena de forma segura en el almacenamiento seguro de Expo (Expo Secure Store) que ya se comentó anteriormente, de forma que al hacer la llamada a otro servicio se extrae de dicho almacenamiento y se envía como parámetro.

Esta clave expira pasados unos minutos, por lo que es necesaria la obtención de una nueva cuando esto ocurre. El mecanismo para llevar a cabo esta tarea consiste en comprobar la respuesta del servidor tras la llamada a un servicio. Si la respuesta es un mensaje de error de clave expirada, de forma transparente al usuario se solicita una nueva haciendo uso del servicio `/comprobarAcceso()` y se almacena de nuevo en el almacenamiento seguro de Expo, para posteriormente, hacer de nuevo la llamada al servicio correspondiente de la API.

### 5.3.3 Diagramas de Secuencia

A continuación, se van a presentar los diagramas de secuencia para el inicio de sesión de un usuario y para la invocación de un servicio cualquiera de la API.

#### 5.3.3.1 Diagrama de secuencia del inicio de sesión

Cuando un usuario inicia sesión en la aplicación, el servicio `/comprobarAcceso()` devuelve una clave (sessionID) que se debe enviar como parámetro en la llamada a cualquiera de los servicios de la API como mecanismo de seguridad. Esta clave se almacena de forma segura en el dispositivo haciendo uso de Expo Secure Store para recuperarla posteriormente en la llamada a otro servicio.

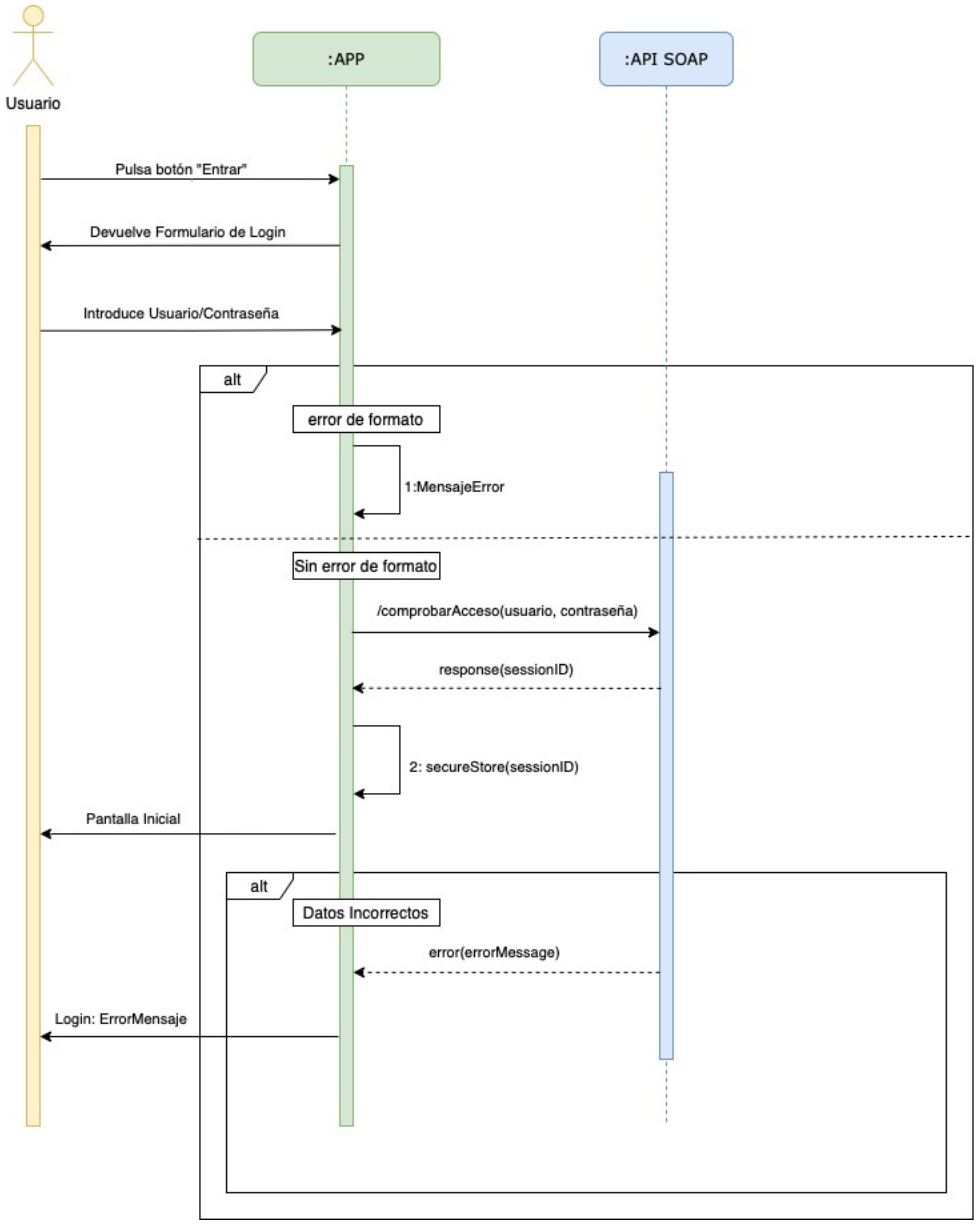


***Ilustración 50 - API SOAP: Diagrama de secuencia del inicio de sesión***

#### 5.3.3.2 Diagrama de secuencia de la llamada a un servicio

En este subapartado se van a presentar dos diagramas de secuencia para la llamada a cualquier servicio de la API. En primer lugar, se presentará el caso de cuando la clave no ha expirado y, en segundo lugar, cuando la API devuelve que la clave no es válida y por tanto debemos obtener una nueva.

##### 5.3.3.2.1 Clave No Expirada

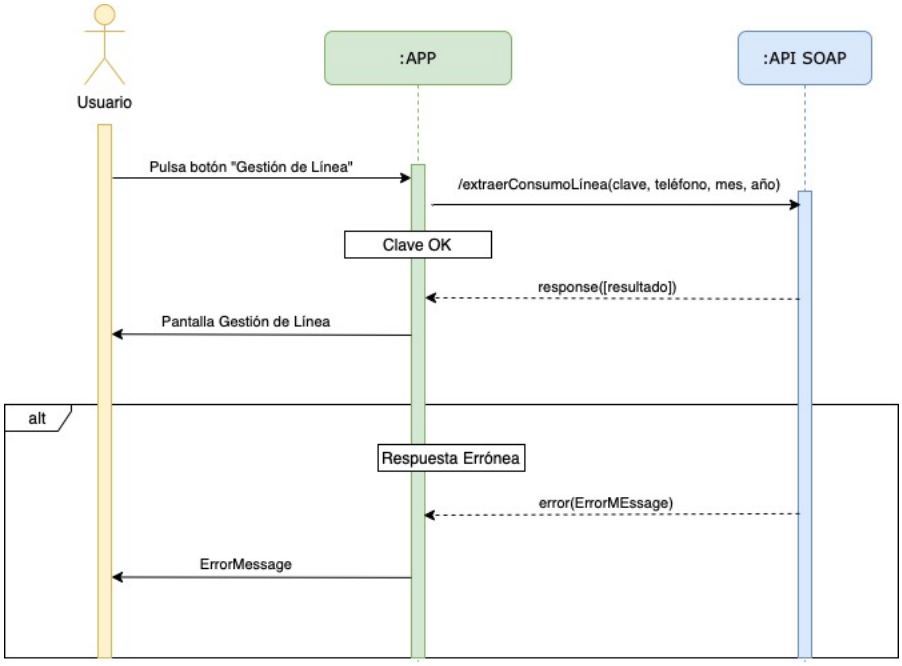


*Ilustración 51 - API SOAP: Diagrama de secuencia: Clave Correcta*

#### 5.3.3.2.2 Clave Expirada

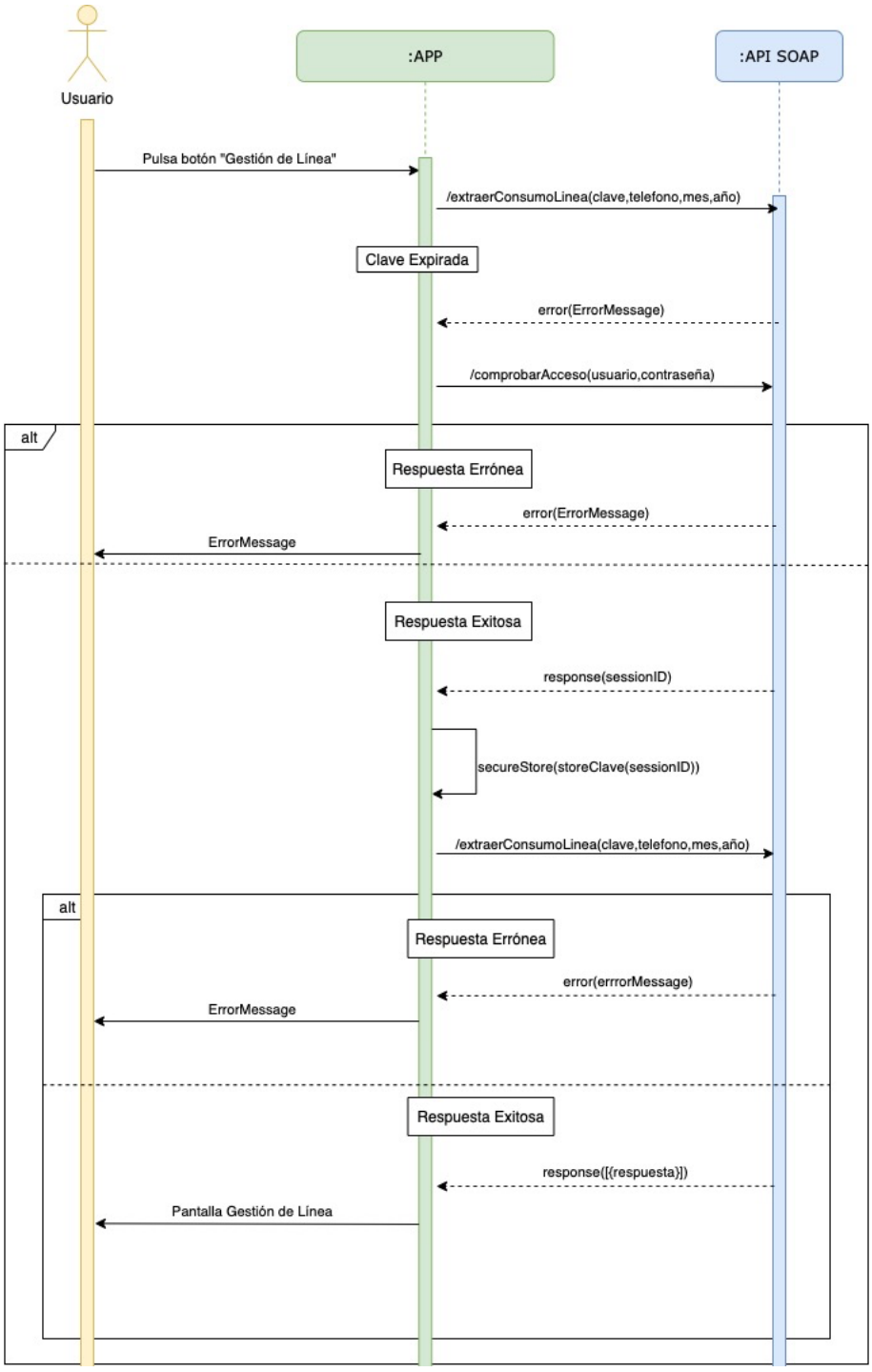


***Ilustración 52 - API SOAP: Diagrama de secuencia: Clave Expirada***

## 5.4 API REST

En este apartado se va a detallar la API REST implementada y que está en uso en el cliente -aplicación-.

### 5.4.1 Casos de uso

Como se ha comentado anteriormente, se distinguen dos roles distintos a la hora de iniciar sesión en la aplicación. A continuación, se van a detallar los diagramas de casos de uso para cada uno de ellos.

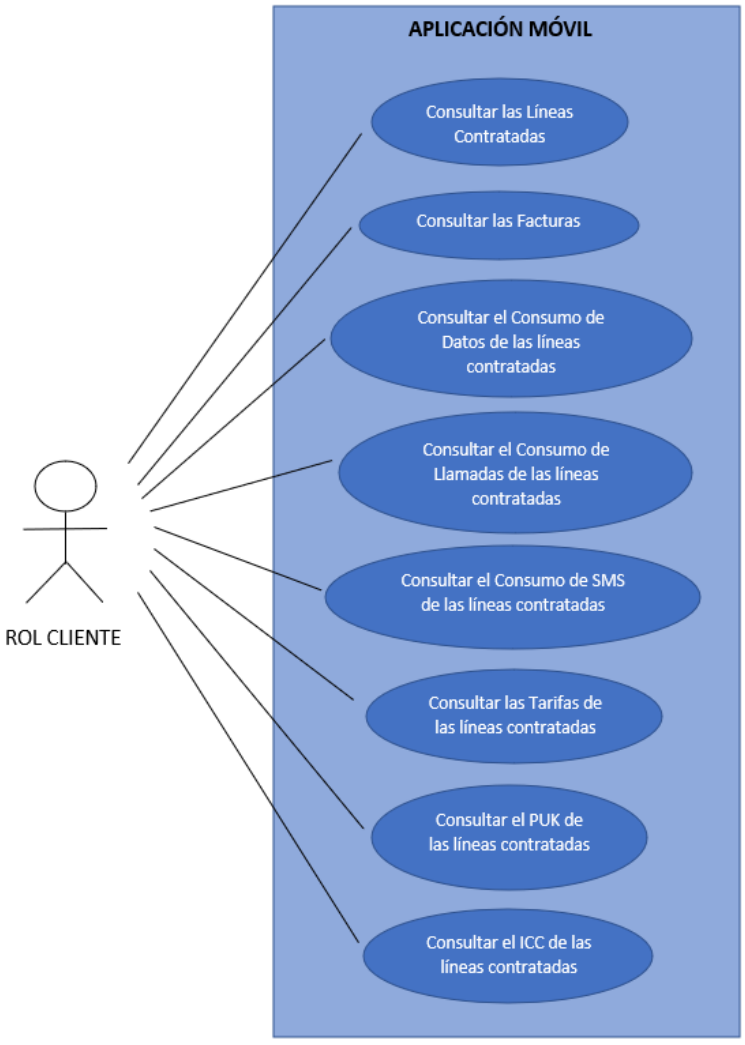


*Ilustración 53 - Diagrama de casos de uso Rol Cliente*

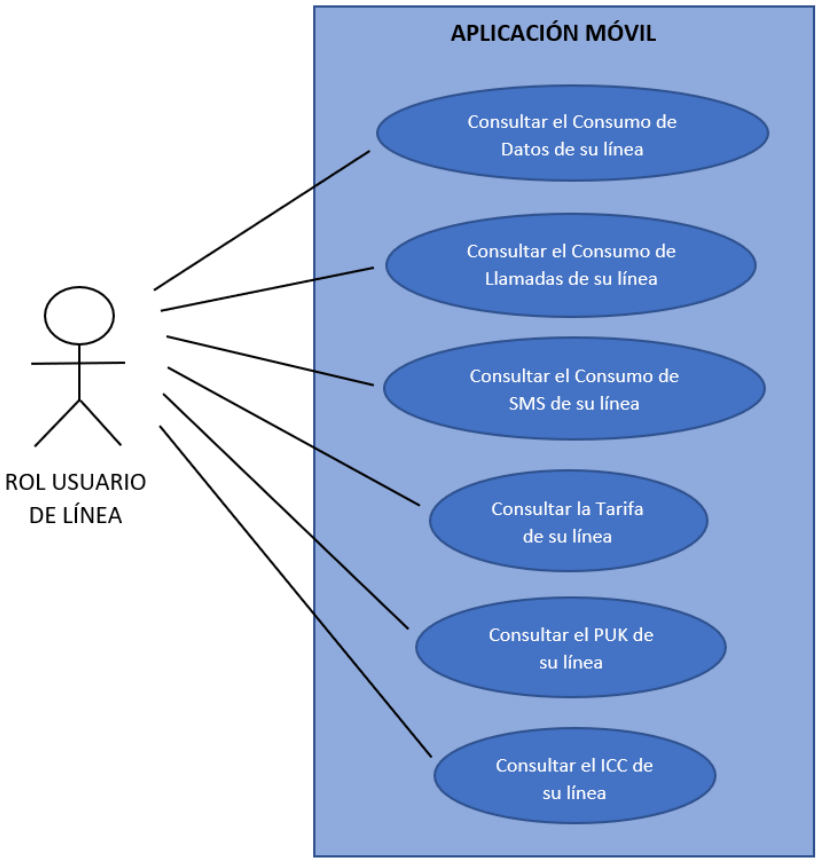


***Ilustración 54 - Diagrama de casos de uso Rol Usuario***

| **CU-01** | **Iniciar Sesión en la Aplicación** | | |
|---|---|---|---|
| Precondición | El usuario no se ha autenticado aún en la aplicación. | | |
| Descripción | El usuario introduce su nombre de usuario y contraseña en el formulario para acceder a las funcionalidades de la aplicación. | | |
| Secuencia | **Paso** | **Acción** | |
| | 1 | El usuario introduce su nombre de usuario y contraseña. | |
| | 2 | El usuario pulsa el botón de “Iniciar Sesión”. | |
| | 3 | La API REST comprueba que las credenciales son correctas. | |
| | 4 | La API REST devuelve un Bearer Token que es almacenado de forma segura. | |
| Postcondición | El usuario ha iniciado sesión en la aplicación. | | |
| Excepciones | **Paso** | **Acción** | |
| | 1 | Un campo del formulario se ha dejado en blanco. | |
| | | E.1 | La aplicación informa al usuario que los campos del formulario son obligatorios. |
| | | E.2 | Se cancela el caso de uso. |
| | 2 | El usuario y/o contraseña introducidos son incorrectos. | |
| | | E.1 | La API REST devuelve que los datos introducidos son incorrectos. |
| | | E.2 | Se cancela el caso de uso. |

***Tabla 6 – CU-01: Inicio de sesión REST***

| **CU-02-C** | **Consultar Facturas** | | |
|---|---|---|---|
| Precondición | El cliente debe haber iniciado sesión en la aplicación. | | |
| Descripción | El cliente solicita la consulta de las facturas emitidas a lo largo del tiempo. | | |
| Secuencia | **Paso** | **Acción** | |
| | 1 | El cliente pulsa sobre el botón “Facturas”. | |
| | 2 | Se hace la petición a la API REST enviando los parámetros del cliente. | |
| | 3 | La API REST comprueba que los datos recibidos sean correctos. | |
| | 4 | La API REST devuelve el listado de las facturas. | |
| Postcondición | El cliente puede consultar el listado de las facturas emitidas. | | |
| Excepciones | **Paso** | **Acción** | |
| | 1 | El rol que intenta acceder a las facturas se trata del “rol usuario” | |
| | | E.1 | La aplicación informa al usuario que no tiene permiso para acceder a la consulta de las facturas. |
| | | E.2 | Se cancela el caso de uso. |
| | 2 | Los datos enviados a la API REST son incorrectos. | |
| | | E.1 | La API REST devuelve un mensaje con el error que se ha producido. |
| | | E.2 | Se cancela el caso de uso. |

***Tabla 7 – CU-02-C: consultar facturas REST***

| **CU-03-C** | **Descargar Factura en PDF** | | |
|---|---|---|---|
| Precondición | El cliente debe haber iniciado sesión en la aplicación. | | |
| Descripción | El cliente solicita la consulta del detalle para una factura en concreto. | | |
| Secuencia | **Paso** | **Acción** | |
| | 1 | El cliente pulsa sobre una factura del listado. | |
| | 2 | Se hace la petición a la API REST enviando los parámetros necesarios para dicha factura. | |
| | 3 | La API REST comprueba que los datos recibidos sean correctos. | |
| | 4 | La API REST devuelve la factura solicitada en PDF. | |
| Postcondición | El cliente puede consultar el detalle de una factura en formato PDF. | | |
| Excepciones | **Paso** | **Acción** | |
| | 1 | La API REST devuelve un error en la solicitud. | |
| | | E.1 | La aplicación informa al cliente del error ocurrido. |
| | | E.2 | Se cancela el caso de uso. |

***Tabla 8 – CU-03-C: descargar factura en PDF REST***

| **CU-04-C** | **Gestionar una línea telefónica [Cliente]** | | |
|---|---|---|---|
| Precondición | El cliente debe haber iniciado sesión en la aplicación. | | |
| Descripción | El cliente solicita gestionar una línea telefónica de las contratadas. | | |
| Secuencia | **Paso** | **Acción** | |
| | 1 | El cliente pulsa sobre el botón “Gestión de Líneas”. | |
| | 2 | Se hace la petición a la API REST enviando los parámetros del cliente. | |
| | 3 | La API REST comprueba que los datos recibidos sean correctos. | |
| | 4 | La API REST devuelve el listado de las líneas del cliente. | |
| Postcondición | El cliente puede consultar el listado de las líneas que tiene contratadas. | | |
| Excepciones | **Paso** | **Acción** | |
| | 1 | Los datos enviados a la API REST son incorrectos. | |
| | | E.1 | La aplicación informa al cliente del error ocurrido. |
| | | E.2 | Se cancela el caso de uso. |

***Tabla 9 – CU-04-C: gestión de una línea telefónica REST***

| **CU-04-U** | **Gestionar una línea telefónica [Usuario]** | | |
|---|---|---|---|
| Precondición | El usuario debe haber iniciado sesión en la aplicación. | | |
| Descripción | El usuario solicita gestionar su línea telefónica. | | |
| Secuencia | **Paso** | **Acción** | |
| | 1 | El usuario pulsa sobre el botón “Gestión de Líneas”. | |
| | 2 | El usuario debe indicar en el formulario el número de su línea. | |
| | 3 | Se hace la petición a la API REST con el número de línea introducido. | |
| | 4 | La API REST devuelve los datos de la línea solicitada. | |
| Postcondición | El usuario puede consultar los datos de la línea que está usando. | | |
| Excepciones | **Paso** | **Acción** | |
| | 1 | La línea introducida no es correcta. | |
| | | E.1 | La aplicación informa al usuario del error ocurrido. |
| | | E.2 | Se cancela el caso de uso. |

***Tabla 10 – CU-04-U: Gestión de una línea telefónica REST***

| **CU-05** | **Consultar consumo de datos** | |
|---|---|---|
| Precondición | El usuario debe haber iniciado sesión en la aplicación. | |
| Descripción | El usuario solicita consultar el consumo de datos móviles de una línea telefónica | |
| Secuencia | **Paso** | **Acción** |
| | 1 | El usuario pulsa sobre el botón “Consumo de Datos”. |
| | 2 | Se hace la petición a la API REST. |
| | 3 | La API REST comprueba que los datos recibidos sean correctos. |
| | 4 | La API REST devuelve el listado con el consumo de los datos móviles. |
| Postcondición | El usuario puede consultar el listado con el consumo de datos móviles realizado en el último mes. | |
| Excepciones | **Paso** | **Acción** |
| | 1 | Los datos enviados a la API REST son incorrectos. |
| | | E.1 La aplicación informa al usuario del error ocurrido. |
| | | E.2 Se cancela el caso de uso. |

***Tabla 11 – CU-05: consultar consumo de datos REST***

| CU-06 | Consultar consumo de llamadas | |
|---|---|---|
| Precondición | El usuario debe haber iniciado sesión en la aplicación. | |
| Descripción | El usuario solicita consultar el consumo de llamadas de una línea telefónica | |
| Secuencia | **Paso** | **Acción** |
| | 1 | El usuario pulsa sobre el botón "Consumo de llamadas". |
| | 2 | Se hace la petición a la API REST. |
| | 3 | La API REST comprueba que los datos recibidos sean correctos. |
| | 4 | La API REST devuelve el listado con las llamadas realizadas. |
| Postcondición | El usuario puede consultar el listado las llamadas realizadas en el último mes. | |
| Excepciones | **Paso** | **Acción** |
| | 1 | Los datos enviados a la API REST son incorrectos. |
| | | E.1 La aplicación informa al usuario del error ocurrido. |
| | | E.2 Se cancela el caso de uso. |

***Tabla 12 – CU-06: consultar consumo de llamadas REST***

| **CU-07** | **Consultar consumo de SMS** | | |
|---|---|---|---|
| Precondición | El usuario debe haber iniciado sesión en la aplicación. | | |
| Descripción | El usuario solicita la consulta de los SMS enviados desde una línea telefónica. | | |
| Secuencia | **Paso** | **Acción** | |
| | 1 | El usuario pulsa sobre el botón “Consumo de SMS”. | |
| | 2 | Se hace la petición a la API REST. | |
| | 3 | La API REST comprueba que los datos recibidos sean correctos. | |
| | 4 | La API REST devuelve el listado con el consumo de SMS. | |
| Postcondición | El usuario puede consultar el listado con los SMS enviados en el último mes. | | |
| Excepciones | **Paso** | **Acción** | |
| | 1 | El listado de SMS está vacío. | |
| | | E.1 | La aplicación informa al usuario que no ha enviado ningún SMS mediante una animación. |
| | | E.2 | Se cancela el caso de uso. |
| | 2 | Los datos enviados a la API REST son incorrectos. | |
| | | E.1 | La API REST devuelve un mensaje con el error que se ha producido. |
| | | E.2 | Se cancela el caso de uso. |

***Tabla 13 – CU-07: consultar consumo de SMS REST***

| **CU-08** | **Consultar PUK o ICC de la SIM** | | |
|---|---|---|---|
| Precondición | El usuario debe haber iniciado sesión en la aplicación. | | |
| Descripción | El usuario solicita la consulta el código PUK o ICC de la tarjeta SIM asociada a una línea telefónica. | | |
| Secuencia | **Paso** | **Acción** | |
| | 1 | El usuario accede a la gestión de una línea telefónica. | |
| | 2 | Se hace la petición a la API REST. | |
| | 3 | La API REST comprueba que los datos recibidos sean correctos. | |
| | 4 | La API REST devuelve el código PUK e ICC de la SIM. | |
| Postcondición | El usuario puede consultar el PUK e ICC de la SIM asociada a una línea telefónica. | | |
| Excepciones | **Paso** | **Acción** | |
| | 1 | Los datos enviados a la API REST son incorrectos. | |
| | | E.1 | La API REST devuelve un mensaje con el error que se ha producido. |
| | | E.2 | Se cancela el caso de uso. |

***Tabla 14 – CU-08: consultar PUK e ICC de la SIM REST***

| **CU-09** | **Consultar la tarifa aplicada a una línea** | | |
|---|---|---|---|
| Precondición | El usuario debe haber iniciado sesión en la aplicación. | | |
| Descripción | El usuario solicita la consulta la tarifa aplicada a una línea telefónica. | | |
| Secuencia | **Paso** | **Acción** | |
| | 1 | El usuario accede a la gestión de una línea telefónica. | |
| | 2 | Se hace la petición a la API REST. | |
| | 3 | La API REST comprueba que los datos recibidos sean correctos. | |
| | 4 | La API REST devuelve la tarifa aplicada a la línea. | |
| Postcondición | El usuario puede consultar la tarifa aplicada a una línea telefónica (nombre de la tarifa, coste mensual y tope de megas). | | |
| Excepciones | **Paso** | **Acción** | |
| | 1 | Los datos enviados a la API REST son incorrectos. | |
| | | E.1 | La API REST devuelve un mensaje con el error que se ha producido. |
| | | E.2 | Se cancela el caso de uso. |

***Tabla 15 – CU-09: consultar tarifa aplicada a línea REST***

### 5.4.2 Diagramas de Secuencia

En este apartado se van a presentar los diagramas de secuencia para el inicio de sesión de un usuario/cliente en la aplicación y para la llamada a cualquiera de los servicios presentes en la API REST.

#### 5.4.2.1 Diagrama de secuencia del inicio de sesión

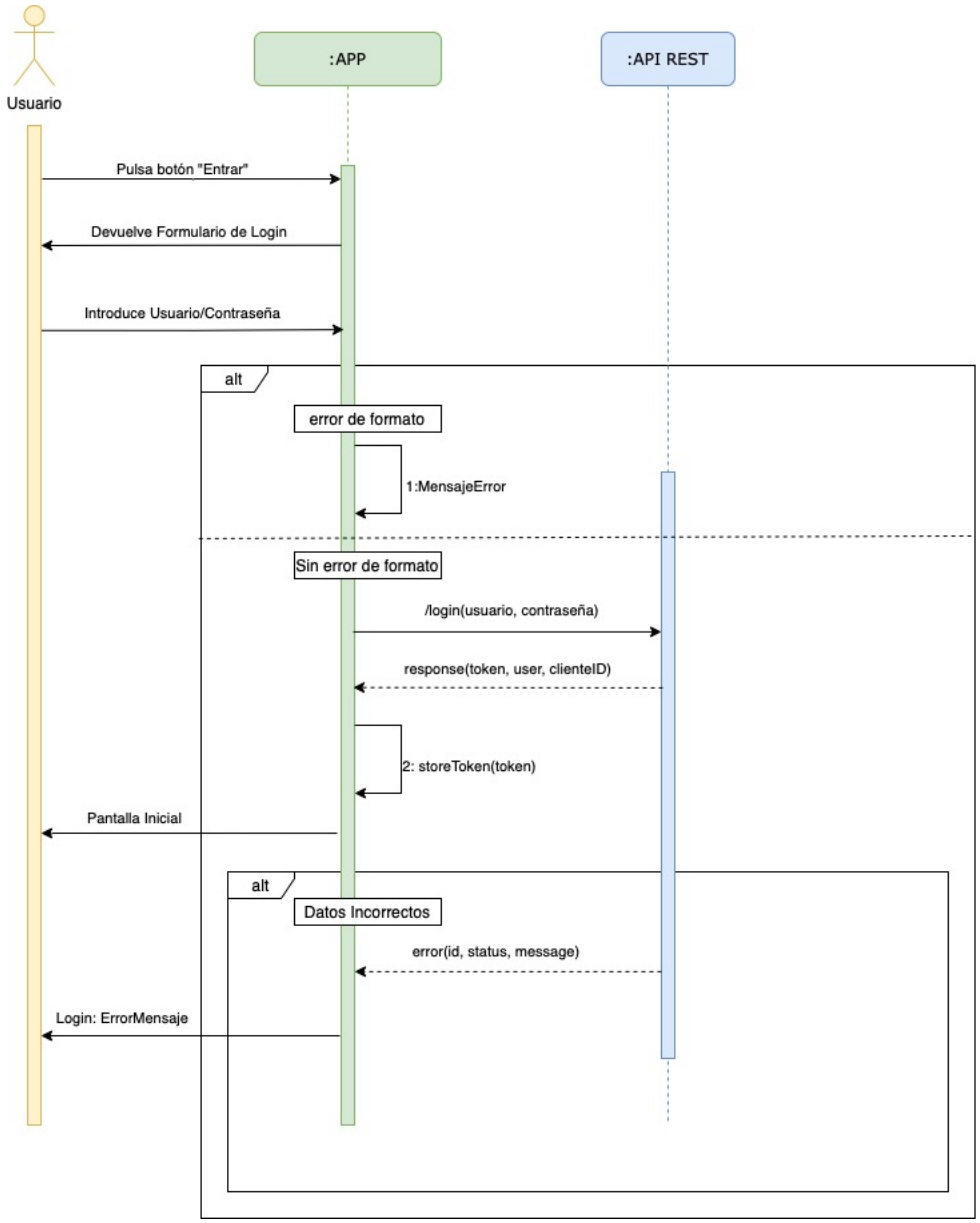


*Ilustración 55 - API REST: Diagrama de secuencia del inicio de sesión*

#### 5.4.2.2 Diagrama de secuencia de la llamada a un servicio

En este subapartado se van a presentar dos diagramas de secuencia para la llamada a cualquiera de los servicios presentes en la API REST.

Antes de realizar la llamada a un servicio se comprueba de forma local si el token ha expirado para, en ese caso, obtener uno nuevo haciendo uso del servicio `/login()`. En el siguiente apartado, se explicará en detalle el sistema del token de autenticación.

##### 5.4.2.2.1 Token No Expirado

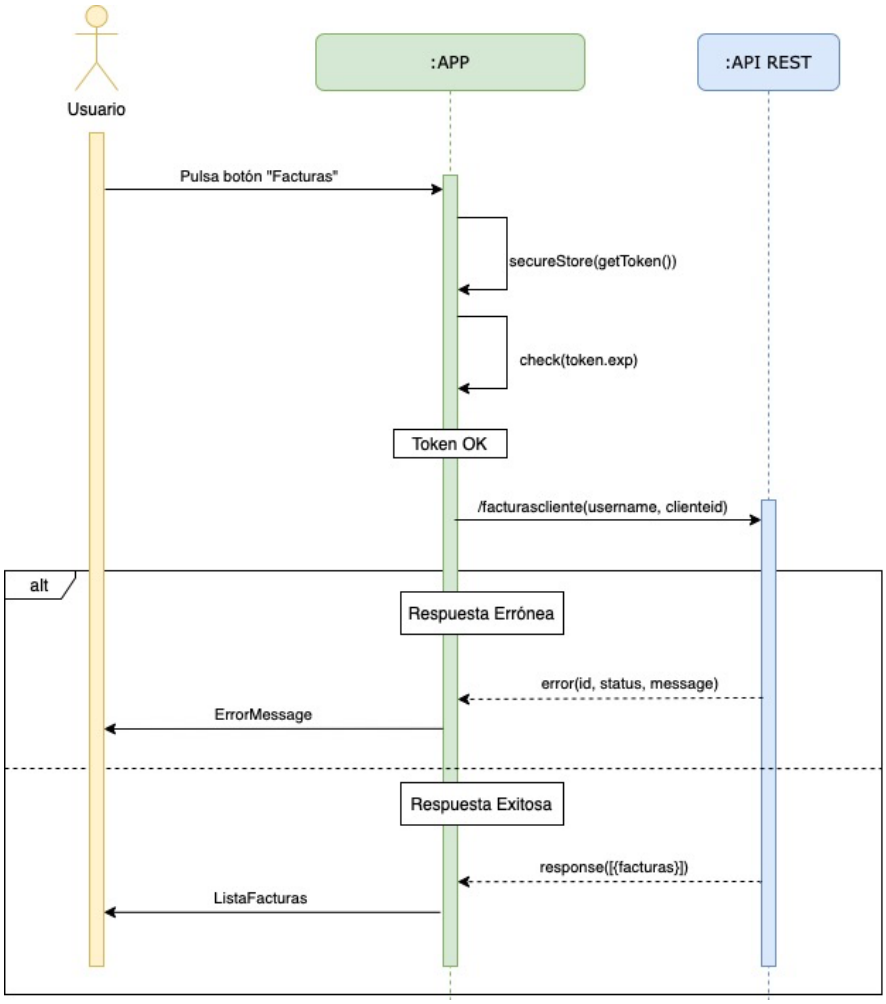


*Ilustración 56 - Diagrama de secuencia llamada a servicio: Token OK*

#### 5.4.2.2.2 Token Expirado

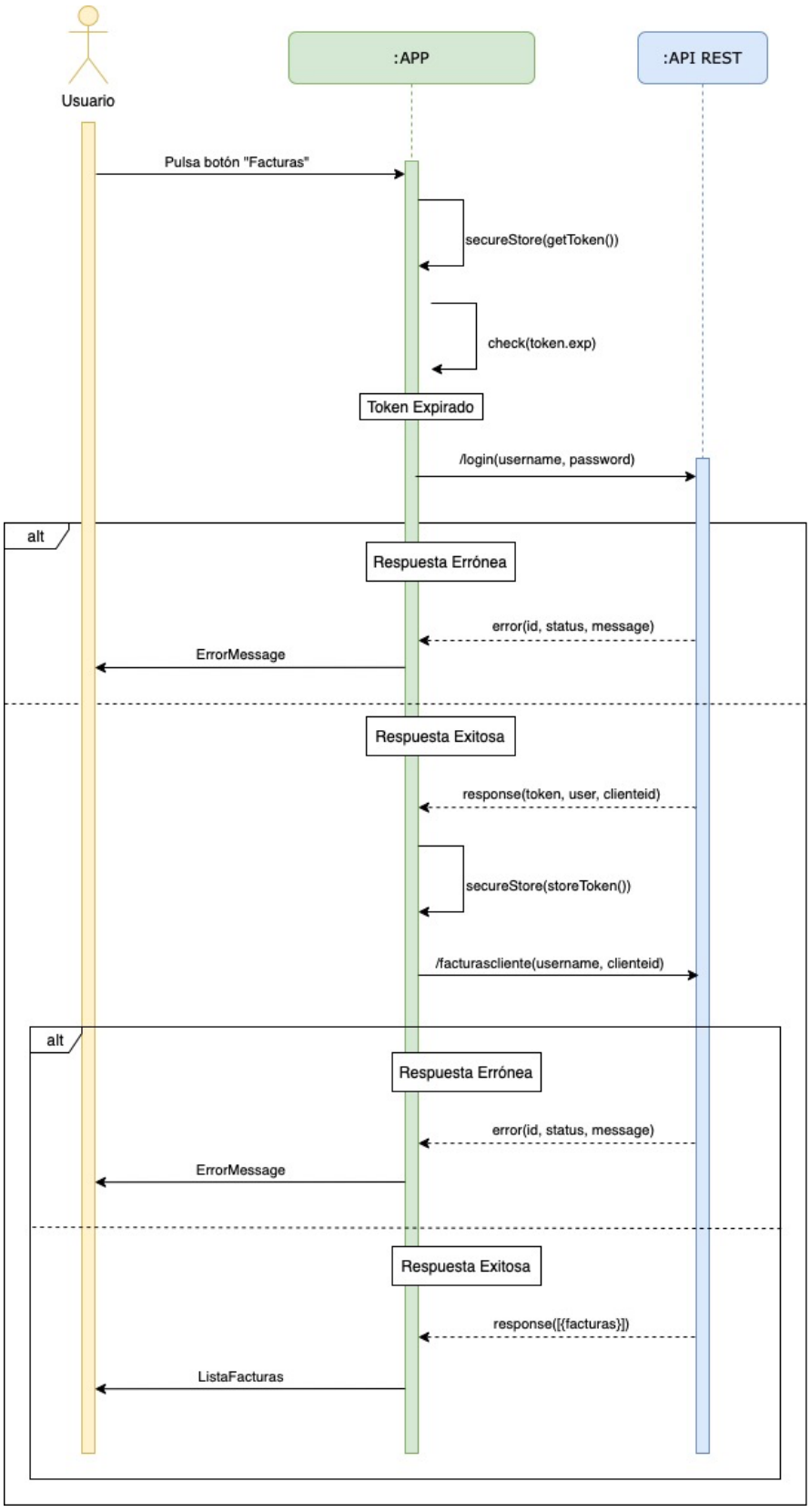


***Ilustración 57 - Diagrama de secuencia llamada a servicio: Token Expirado***

### 5.4.3 Sistema de Seguridad: Bearer Token

El procedimiento del inicio de sesión en la aplicación se ha realizado siguiendo el esquema de autenticación *Bearer* [35].

La autenticación *Bearer*, también conocida como autenticación *Token*, es un esquema de autenticación del protocolo HTTP para verificar la identidad de un usuario en una aplicación. Esta verificación se realiza por medio de un *token* que el usuario debe enviar al servidor en cada petición que haga.

Estos *tokens* se tratan de cadenas encriptadas en formato JSON que son generadas por un servidor en respuesta a un inicio de sesión. Para verificar la identidad, el cliente deberá enviar ese *token* en la cabecera `Authorization` del protocolo HTTP a la hora de hacer cualquier petición al servidor. A continuación, se muestra un ejemplo de como sería dicha cabecera.

```
Authorization: Bearer <token>
```

***Ilustración 58 - Cabecera Authorization HTTP***

Para la creación de un *token* se emplea el estándar *JSON Web Token*, o abreviado, JWT.

Los JWT constan normalmente de tres partes que se envían codificadas al cliente:

- ***header***: se trata de un encabezado que indica el algoritmo utilizado para generar la firma. En la siguiente ilustración se muestra un ejemplo de cabecera.

```
{
    "alg":"HS256",
    "typ":"JWT"
}
```

***Ilustración 59 - JWT Header***

- ***payload***: contiene toda la información del token. Esta compuesta por varias propiedades también llamadas *claims* del token. Entre ellas se destacan:
    - *iss*: esta propiedad se encarga de identificar al emisor del *token*.
    - *iat*: identifica la hora en la que fue creado el *token*.
    - *exp*: identifica la hora de expiración del *token*, a partir de la cual, ya no será valido para usarlo de nuevo.

A continuación, se muestra un ejemplo de *payload*.

```
{
    "iss": "https://sts.windows.net/common/",
    "iat": 1552212046,
    "exp": 1552215946,
}
```

***Ilustración 60 - JWT Payload***

- ***signature***: se trata de la firma del *token* calculada codificando la cabecera (*header*) y el contenido (*payload*), normalmente, en *base64*. A continuación se muestra un ejemplo utilizando el algoritmo RSA256.

```
signature = RSA256(
    encodeURI(base64(header))
    + "." +
    encodeURI(base64(payload)),
    private_key
)
```

***Ilustración 61 - JWT Signature***

Una vez que el cliente recibe un *token* tras un inicio de sesión, debe ser capaz de obtener las tres partes que se acaban de comentar. Para ello, se debe decodificar el *token* empleando alguna herramienta específica. La herramienta empleada en el desarrollo de la aplicación ha sido el método `jwtDecode()` de la librería `jwt-decode` [36].

Este método permite decodificar fácilmente un *token* y obtener toda su información. En la siguiente ilustración se muestra un ejemplo de uso.

```
const user = jwtDecode("token");
```

***Ilustración 62 - Uso de jwtDecode()***

También es posible realizar la decodificación de un *token* empleando herramientas online para comprobar su estructura. Una de ellas es *"jwt.io"*, la cual se puede encontrar en el siguiente enlace: https://jwt.io/ Esta herramienta, al igual que el método `jwtDecode()`, decodifica un *token* en todas sus partes. A continuación, se muestra la interfaz de esta web.

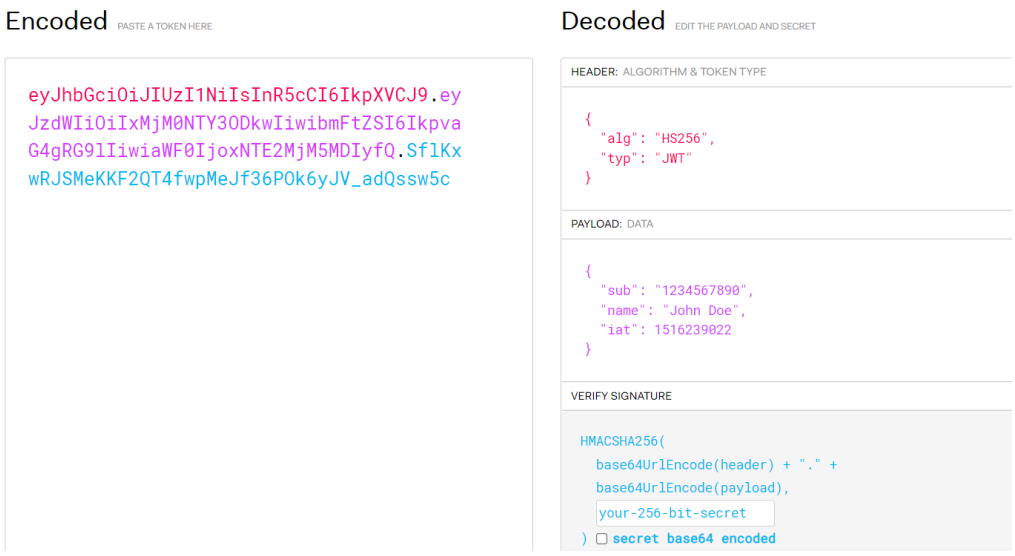


***Ilustración 63 - jwt.io***

#### 5.4.3.1 Expiración del Token: Obtención de uno nuevo

Como se ha comentado anteriormente, la propiedad *exp* permite determinar el momento de expiración de un token, a partir del cual ya no es válido su uso. Cuando esto ocurre, es necesario obtener un nuevo token para poder seguir usando los servicios de la API REST. Esta API está configurada de manera que el tiempo de vida del token que devuelve el servicio `/login()` es de dos minutos.

Antes de realizar la llamada a cualquiera de los servicios de la API REST, lo primero que se comprueba es si el token ha expirado. Esta comprobación se realiza de forma local haciendo uso de la mencionada propiedad *exp*. Para ello, se hace uso del método `now()` del componente `Date` con el objetivo de comprobar si el momento de expiración del token ha llegado ya o no.

En el caso de que ese momento aún no haya llegado -lo que quiere decir que el token no ha expirado- no se realiza ninguna acción y se procede a la llamada del servicio solicitado de la API.

Por el contrario, si el token ya ha expirado se procede entonces a la obtención de uno nuevo haciendo uso del servicio `/login()`. Para ello, se recupera del almacenamiento seguro de Expo las credenciales del usuario que ha iniciado sesión en la aplicación y se invoca el servicio `/login()` para obtener un nuevo token, que será de nuevo guardado de forma segura sustituyendo al anterior.

A continuación, se ejemplifica todo este procedimiento con un fragmento de código.

```
const isTokenExpired = async () => {
  const token = await authStorage.getToken();
  const decoded = jwtDecode(token);
  const username = await authStorage.getUsername();
  const password = await authStorage.getPassword();

  if (decoded.exp < Date.now() / 1000) {
    //Token caducado - Obtenemos uno nuevo
    const response = await authApi.login(username, password);

    //Gestionamos posibles errores de respuesta de la API
    if (response.data.error?.status == (401 || 500)) {
      Alert.alert("Error", response.data.error.message, [
        {
          text: "Aceptar",
        },
      ]);
    } else {
      await authStorage.storeToken(response.data["token"]);
    }
  } else {
    //Token Actualizado - No es necesario refrescarlo
  }
};
```

*Ilustración 64 - Fragmento de código de la obtención de un nuevo token*

# 6 INTERFAZ DE USUARIO

*El diseño no es solo lo que se ve y lo que se siente.*
*El diseño es cómo funciona*

*Steve Jobs*

El contenido de este apartado se va a dedicar a mostrar las diferentes pantallas que componen la interfaz de usuario de la aplicación desarrollada.

## 6.1 Pantalla de Bienvenida

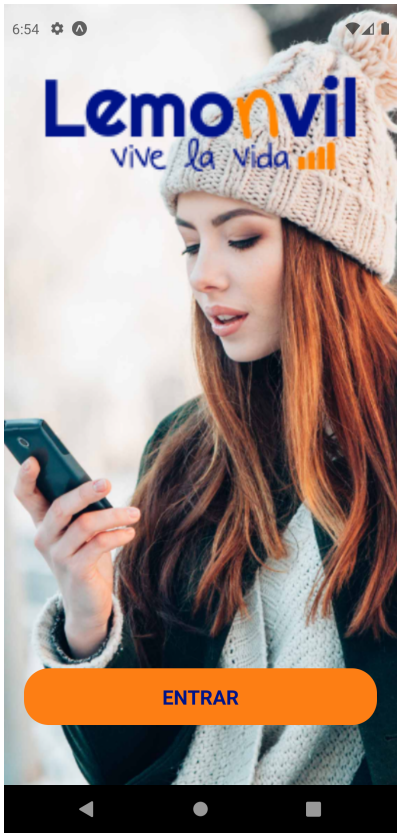


*Ilustración 65 - Pantalla de Bienvenida*

La primera pantalla que se observa al abrir la aplicación es la pantalla de “Bienvenida” que se muestra en la Ilustración 65. En ella se puede observar el logotipo de Lemonvil, una imagen de fondo y un botón que permite entrar a la aplicación el cual conducirá al usuario a la pantalla de “Login” o inicio de sesión.

## 6.2 Pantalla de Inicio de Sesión

Como se observa en la Ilustración 66, esta pantalla muestra el formulario que se debe rellenar para que un usuario pueda iniciar sesión en la aplicación. Los campos requeridos son el nombre de usuario y su contraseña. También se ha incluido un botón que permite ocultar y mostrar la contraseña.

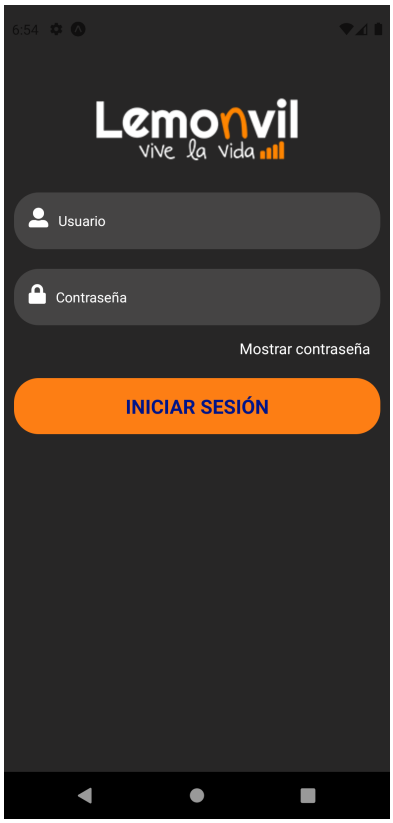


*Ilustración 66 - Pantalla de Inicio de Sesión*

## 6.3 Menú Inicial

Una vez que un usuario ha iniciado sesión correctamente, se le conducirá a la pantalla principal de la aplicación. Desde esta pantalla, llamada pantalla de "Inicio", se podrá acceder al resto de funcionalidades de la aplicación.


*Ilustración 69 - Pantalla de Inicio I*

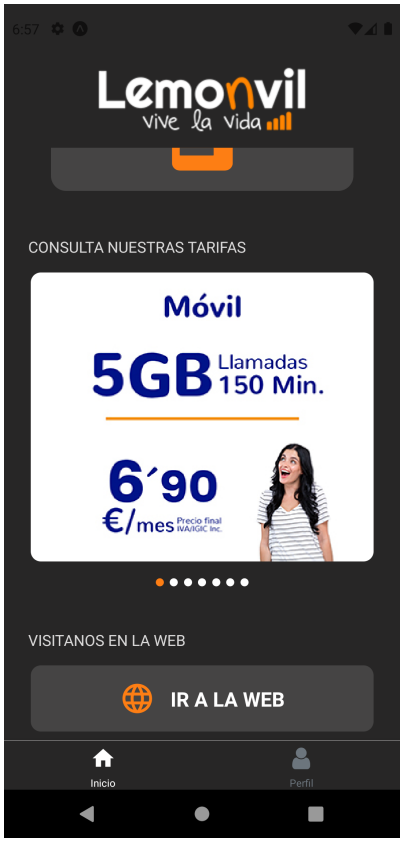


*Ilustración 68 - Pantalla de Inicio II*

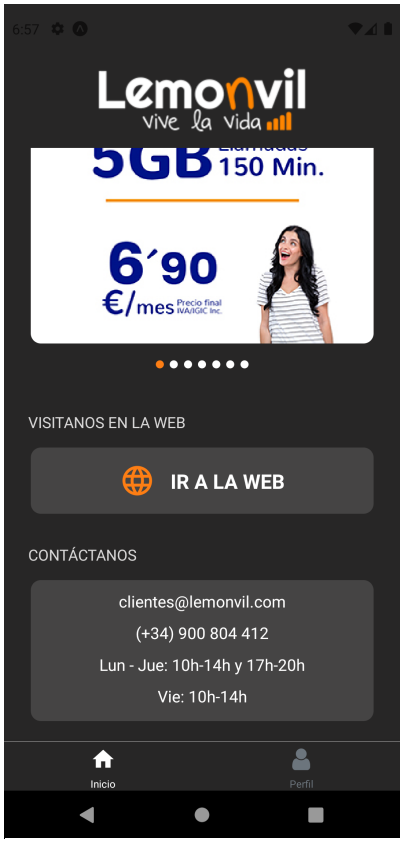


*Ilustración 67 - Pantalla de Inicio III*

Como se puede observar en las Ilustraciones 67, 68 y 69, la pantalla de "Inicio" se trata de una pantalla en la que es posible deslizar hacia abajo para ver más contenido.

Lo primero que se observa es el logotipo de Lemonvil seguido de un saludo con el nombre del usuario que ha iniciado sesión en la aplicación. Debajo, encontramos dos botones que nos conducirán a las pantallas de gestión de líneas y facturas, las cuales se mostrarán posteriormente.

Como se observa en la Ilustración 68, la sección *"Consulta nuestras tarifas"* se trata de una serie de imágenes extraídas de la página web de Lemonvil mostrando las tarifas móviles disponibles, y justo debajo, se encuentra la sección de *"Visítanos en la web"*. Esto se trata de un botón que abrirá en el navegador del dispositivo la página web de Lemonvil.

Como se observa en la Ilustración 69, la sección de *"contáctanos"* muestra información de contacto de Lemonvil, como un correo electrónico, un número de teléfono y el horario de atención al cliente.

## 6.4 Facturas

A la pantalla de facturas se llega pulsando el botón “Facturas” en la pantalla de “Inicio”. Como se ha comentado anteriormente, solo en el caso del rol cliente será posible consultar las facturas. Por ello, se distinguirán entre dos pantallas a mostrar dependiendo del rol que inicie sesión en la aplicación.

### 6.4.1 Facturas - Rol Cliente


***Ilustración 70 - Factural Rol Cliente***

Como se puede observar en la Ilustración 70, la pantalla de facturas está compuesta por una serie de botones en la parte superior y justo debajo, un listado con las facturas que se han emitido.

El botón superior *“ver estadísticas”* permite consultar las estadísticas de las facturas emitidas en los últimos meses. Los botones con los títulos *“ordenar por fecha”* y *“ordenar por importe”* permiten ordenar el listado de todas las facturas por fecha e importe en orden ascendente o descendente y, finalmente, debajo de esta serie de botones encontramos el listado de las facturas que se han emitido.

### 6.4.2 Detalle de factura

A la pantalla “Detalle de factura” se llega al pulsar sobre una factura en la pantalla de “Facturas”. Esta pantalla permite ver en detalle los datos de la factura que se ha seleccionado.


*Ilustración 71 - Pantalla Detalle de Factura*

Como se observa en la Ilustración 71, en esta pantalla se muestra de una forma más cómoda el número de la factura que se ha seleccionado, su importe, la fecha en la que se ha emitido, y un botón que permite visualizarla y descargarla en formato PDF.

### 6.4.3 Estadísticas de las facturas

A la pantalla de "Estadísticas de las facturas" se llega al pulsar el botón *"ver estadísticas"* situado en la pantalla de "Facturas" que se mostraba en la Ilustración 70.


***Ilustración 72 - Pantalla de Estadísticas de Facturas***

Como se puede observar en la Ilustración 72, el contenido de esta pantalla consta de una gráfica que muestra el importe de las facturas de los últimos cinco meses y el gasto medio en ese período de tiempo.

### 6.4.4 Facturas - Rol Usuario

El rol usuario no tiene acceso a consultar las facturas. Por tanto, cuando se inicia sesión como usuario en la aplicación y se pulsa el botón *Facturas* en la pantalla de "Inicio", la pantalla que se mostrará será la de la Ilustración 73.


***Ilustración 73 - Facturas Rol Usuario***

El contenido de esta pantalla consta de una imagen animada y de un mensaje que indica que no se tiene permiso para poder ver las facturas.

## 6.5 Selección de Línea

A la pantalla de "Selección de Línea" se llega al pulsar el botón *"Gestión de Líneas"* que se encuentra en la pantalla de "Inicio" tal y como se observó en la Ilustración 67.

Desde esta pantalla se podrá seleccionar la línea que se quiere gestionar. Nuevamente, el contenido de esta pantalla será distinto dependiendo del rol que inicie sesión en la aplicación. A continuación, se muestra cada una de ellas.

### 6.5.1 Selección de línea - Rol Cliente

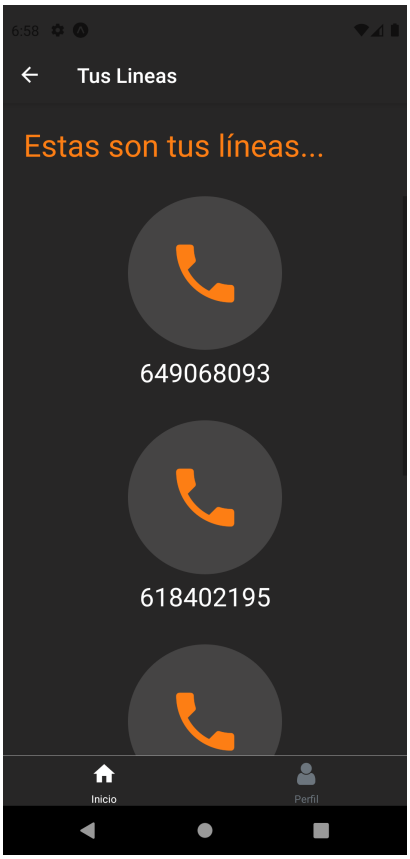


***Ilustración 74 - Selección de Línea Rol Cliente***

Como se observa en la Ilustración 74, el contenido de esta pantalla consta de una lista con todas las líneas de las que dicho cliente es titular. Desde aquí, se podrá seleccionar la línea que se desea gestionar, y una vez seleccionada, se le conducirá a la pantalla de "Gestión de Línea".

### 6.5.2 Selección de línea - Rol Usuario


***Ilustración 75 - Selección de Línea Rol Usuario***

En el caso del rol usuario -al solo usarse una única línea- es necesario configurar la aplicación con dicha línea. Para ello, hay que indicarla en el campo correspondiente y pulsar el botón *"continuar"*. Tras esto, si la línea introducida es correcta, se conducirá al usuario a la pantalla de "Gestión de Línea".

Como nota adicional, el campo para introducir la línea es de tipo numérico, por lo que solamente será posible escribir caracteres de este tipo.

## 6.6 Gestión de Línea

A la pantalla de “Gestión de Línea” se llega al seleccionar una de las líneas disponibles en la pantalla de “Selección de Línea” en el caso del rol cliente, o bien, al introducir de forma correcta una línea para el caso del rol usuario.


*Ilustración 76 - Pantalla de Gestión de Línea I*

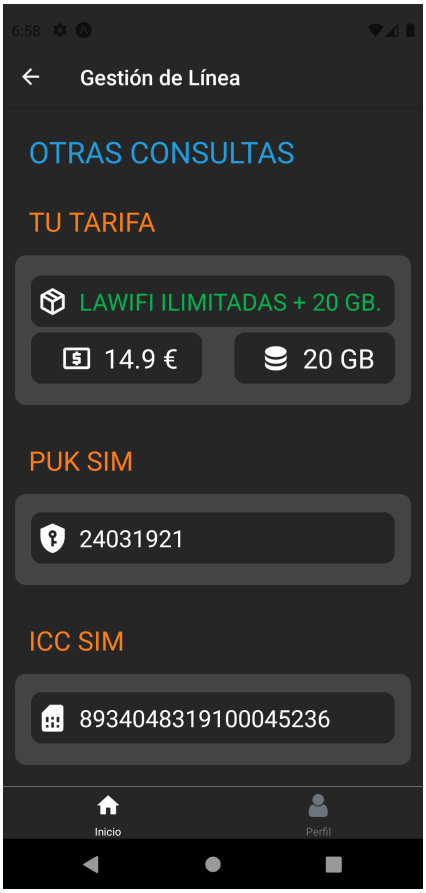


*Ilustración 77 - Pantalla de Gestión de Línea II*

El aspecto de la pantalla de “Gestión de Líneas” es el que se observa en las Ilustraciones 76 y 77. En ella es posible deslizar hacia arriba y abajo para ver más contenido.

Como se aprecia en la Ilustración 76, lo primero que aparece es el número de la línea a gestionar que se ha seleccionado o introducido dependiendo del rol. A continuación, encontramos una serie de botones que permiten consultar toda la información relativa al consumo de esa línea, tales como el consumo de datos, llamadas y SMS.

Como se observa en la Ilustración 77, en la sección de *“Otras Consultas”* se podrá consultar otra información relativa a la línea como la tarifa aplicada y los códigos PUK e ICC de la SIM.

El código PUK es un código de ocho dígitos que permite desbloquear una tarjeta SIM en caso de que el número PIN se haya introducido tres veces de forma incorrecta, y el código ICC es un número de 19 dígitos que identifica a nivel internacional una tarjeta SIM.

## 6.7 Consumo de Datos

A la pantalla de "Consumo de Datos" se llega al pulsar el botón *"consulta tus datos"* en la pantalla de "Gestión de Línea" que se ha comentado en el apartado anterior.


***Ilustración 78 - Pantalla de Consumo de Datos***

Como se puede observar en la Ilustración 78, encontramos una serie de botones en la parte superior y justo debajo, el listado con todo el consumo de datos realizado.

El botón superior con el título *"ver estadísticas"* permite consultar de una forma gráfica las estadísticas del consumo de datos que ha realizado hasta la fecha de la consulta.

Los botones con los títulos *"ordenar por fecha"* y *"ordenar por megas"* permiten ordenar el listado por fecha y consumo de megas en orden ascendente o descendente.

### 6.7.1 Estadísticas del consumo de datos

A la pantalla de "Estadísticas del consumo de datos" se llega al pulsar el botón *"ver estadísticas"* que se encuentra en la pantalla de "Consumo de Datos".

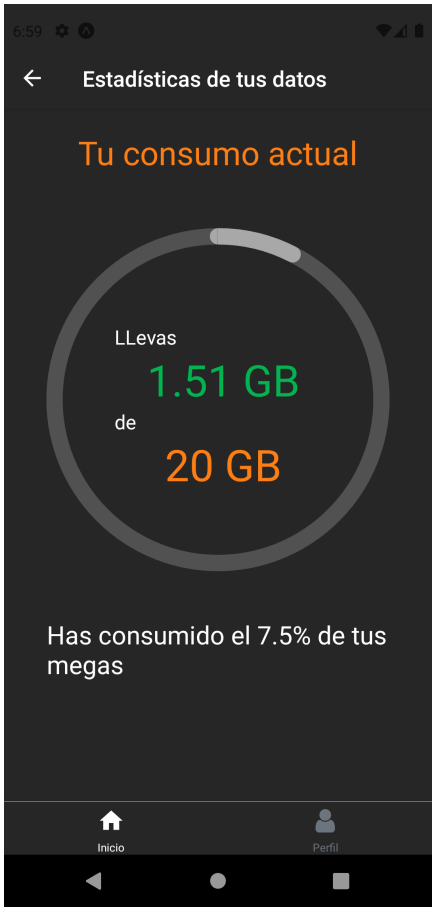


***Ilustración 79 - Pantalla de Estadísticas del consumo de datos***

El aspecto de esta pantalla es el que se muestra en la Ilustración 79. Como se puede observar, consta de una gráfica que muestra el consumo de datos realizado hasta la fecha de la consulta y el porcentaje que se lleva consumido.

## 6.8 Consumo de Llamadas

A la pantalla de "Consumo de Llamadas" se llega al pulsar el botón *"consulta tus llamadas"* en la pantalla de "Gestión de Línea".

El aspecto de esta pantalla es el que se muestra en la Ilustración 80. Nuevamente, el botón superior con el título *"ver estadísticas"* permite consultar las estadísticas del consumo de llamadas que ha realizado hasta la fecha de la consulta, y los botones con los títulos *"ordenar por fecha"* y *"ordenar por duración"* permiten ordenar el listado por fecha y duración de las llamadas en orden ascendente o descendente.


***Ilustración 80 - Pantalla de Consumo de Llamadas***

### 6.8.1 Estadísticas del consumo de llamadas

A la pantalla de "Estadísticas del consumo de llamadas" se llega al pulsar sobre el botón *"ver estadísticas"* situado en la pantalla de "Consumo de Llamadas".

En esta pantalla se muestra el tiempo total que se lleva hablado durante el mes actual a fecha de realizar la consulta. También se muestra el número total de llamadas que se llevan efectuadas durante el mes actual.

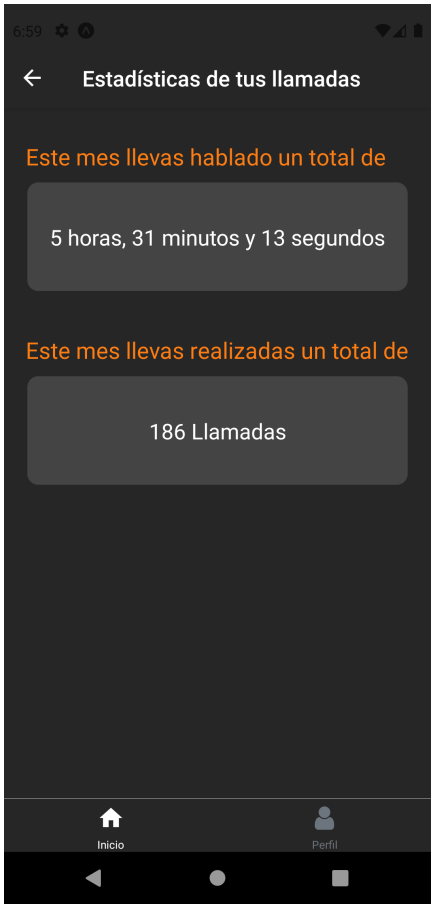


***Ilustración 81 - Pantalla de Estadísticas del consumo de Llamadas***

## 6.9 Consumo de SMS

A la pantalla de “Consumo de SMS” se llega al pulsar el botón *“consulta tus SMS”* situado en la pantalla de “Gestión de Línea”.

Esta pantalla se compone de una lista con todos los SMS enviados desde la línea seleccionada.

En esta ocasión no se han incluido botones para ordenar el listado ni para consultar las estadísticas. El principal motivo es que este canal se usa cada vez con menos frecuencia debido a la existencia de las aplicaciones de mensajería instantánea como WhatsApp o Telegram (entre otras) por lo que el consumo de SMS se estima mucho menor que hace unos años. Por ello, en el caso de que no se haya enviado ningún SMS en el momento en el que se realiza la consulta, se mostrará una animación.

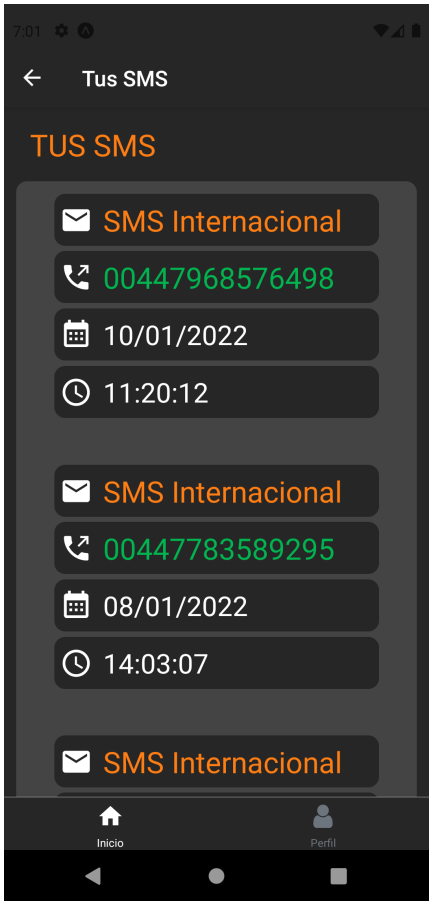


*Ilustración 82 - Pantalla de Consumo de SMS*

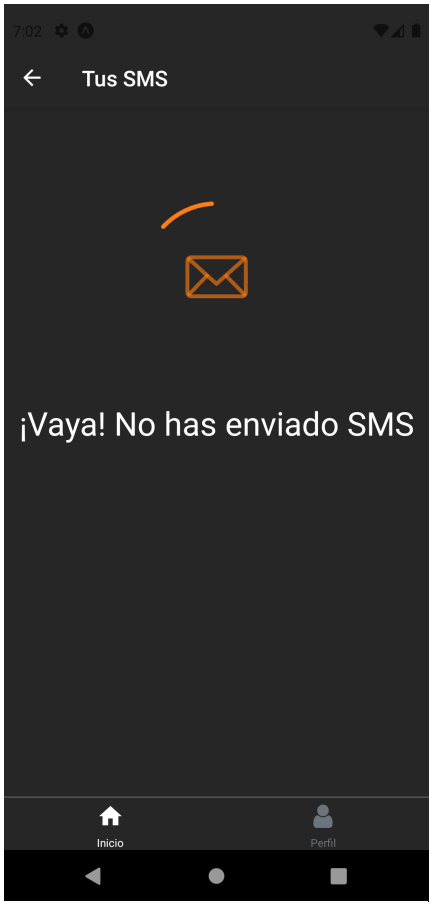


*Ilustración 83 - Pantalla de Consumo de SMS (Ningún SMS enviado)*

## 6.10 Perfil de Usuario

A la pantalla de "Perfil de Usuario" se accede al pulsar sobre el botón *"Perfil"* situado en la barra de navegación de la parte inferior de la pantalla. Desde aquí se podrá consultar información relativa al cliente/usuario que inice sesión en la aplicación.

En la parte superior encontramos una sección donde se especifica el nombre completo del usuario que ha iniciado sesión en la aplicación, el rol que presenta (cliente o usuario) y la posibilidad de añadir una foto de perfil.

El botón *"Cerrar Sesión"* permite cerrar la sesión del usuario actual en la aplicación. Al pulsar dicho botón, se conducirá al usuario a la Pantalla de "Bienvenida".

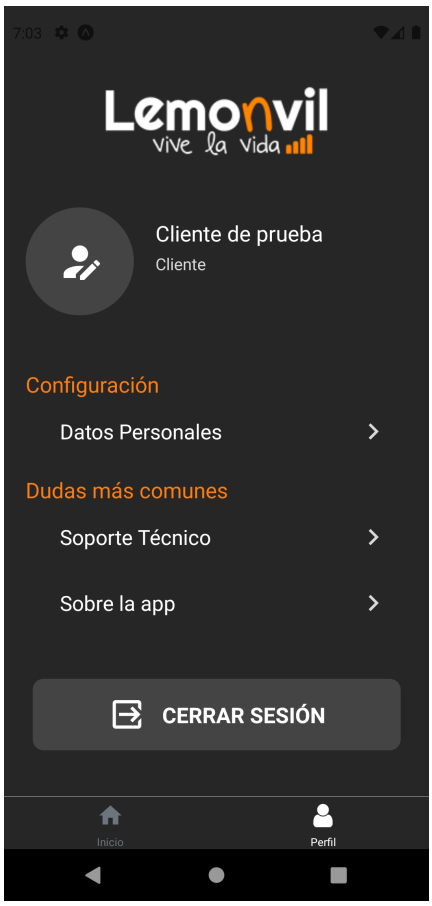


***Ilustración 84 - Pantalla de Perfil de Usuario***

### 6.10.1 Configuración – Datos Personales

La sección de "Configuración" tiene como opción disponible la de "Datos Personales". A ella se accede al pulsar sobre el botón *"Datos Personales"* en la pantalla de "Perfil de Usuario".

Como se puede observar en la Ilustración 85, en esta pantalla es posible seleccionar el número de la línea principal de contacto -solo en el caso del rol cliente- y un correo electrónico de contacto. En el caso del rol usuario, como se muestra en la Ilustración 87, no será posible seleccionar la línea principal de contacto puesto que solo dispone de la que se está usando.

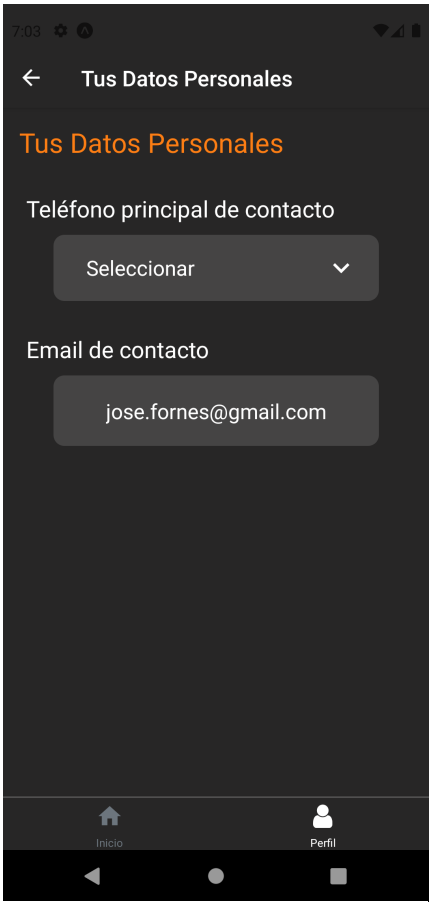


*Ilustración 85 - Pantalla de configuración - Datos Personales: Rol Cliente*

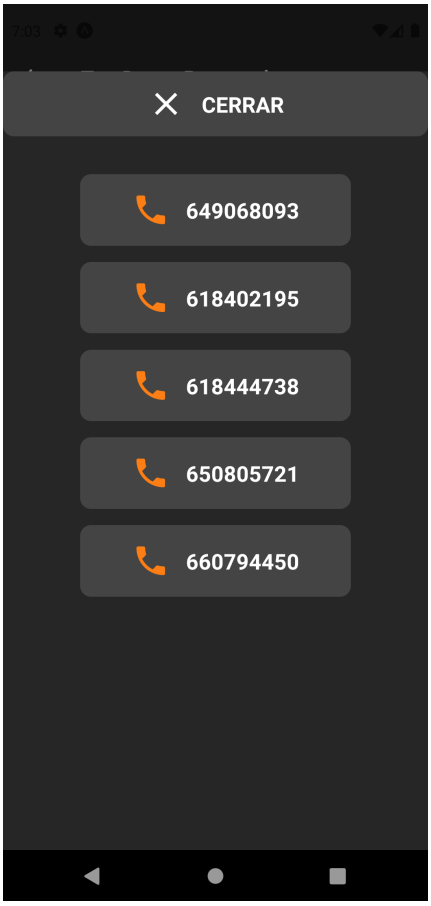


*Ilustración 87 – Pantalla de selección de la línea de contacto*

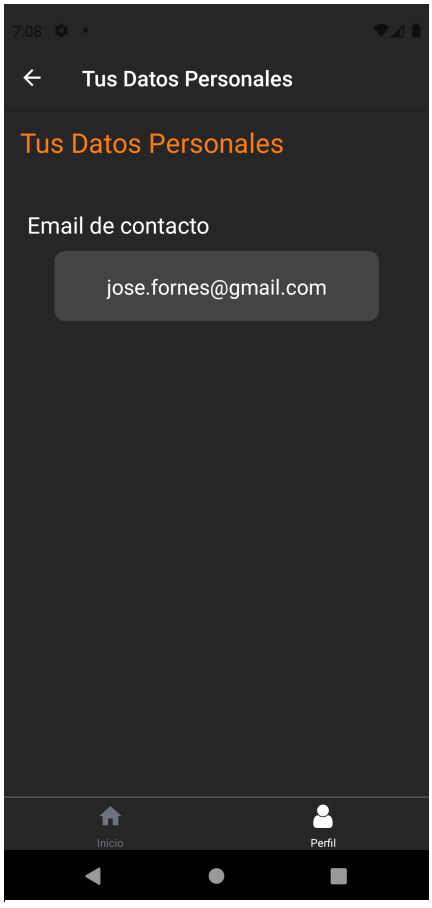


*Ilustración 86 - Pantalla de configuración - Datos Personales: Rol Usuario*

### 6.10.2 Dudas más comunes - Soporte Técnico

La sección de “Dudas más comunes” tiene dos opciones disponibles. La primera de ellas se trata de la de “Soporte Técnico”. Desde esta pantalla se podrá consultar información relativa al soporte técnico de Lemonvil en caso de que exista algún tipo de incidencia en sus servicios.

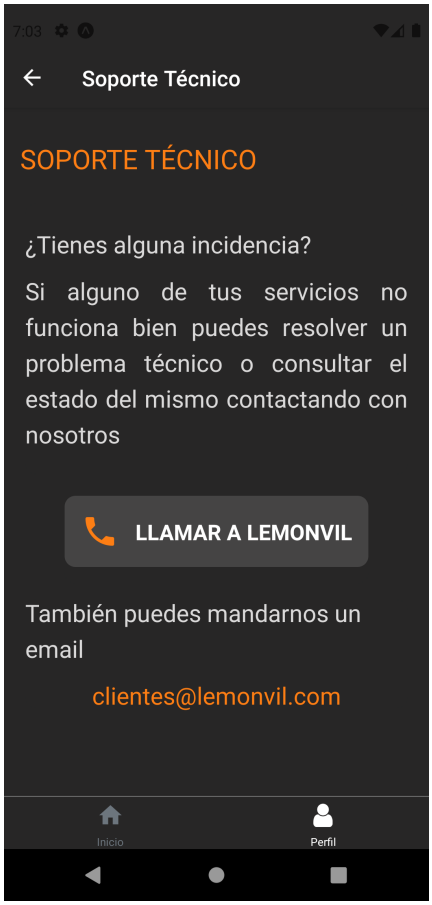


*Ilustración 88 - Pantalla de Soporte Técnico*

En esta pantalla se encuentra un botón que permite llamar directamente a Lemonvil, abriendo en el dispositivo la aplicación del teléfono y marcando automáticamente el número.

### 6.10.3 Dudas más comunes - Sobre la app

La otra opción disponible en la sección de “Dudas más comunes” se trata de la de “Sobre la app” o “acerca de la aplicación”. Desde esta pantalla, como se muestra en la Ilustración 89, se podrá consultar de forma resumida todas las funcionalidades que se pueden hacer con esta aplicación.

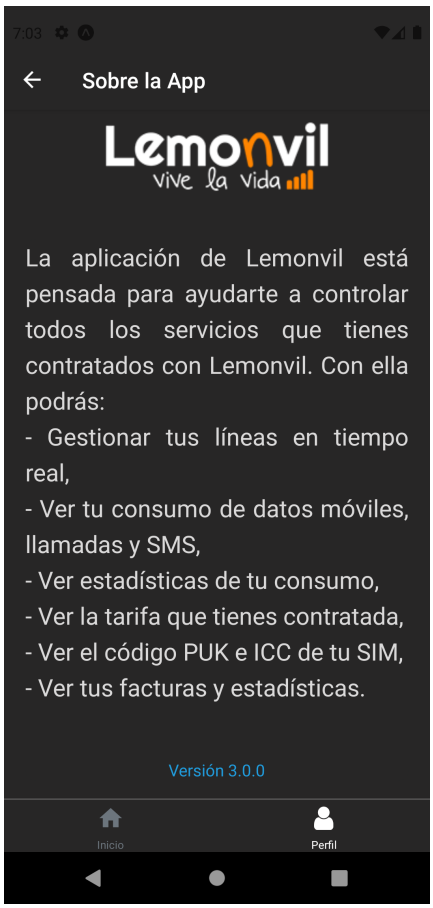


*Ilustración 89 - Pantalla de "Sobre la app"*

## 6.11 Recepción de una notificación

En la Ilustración 90 se muestra el aspecto de una notificación recibida en un dispositivo iOS.

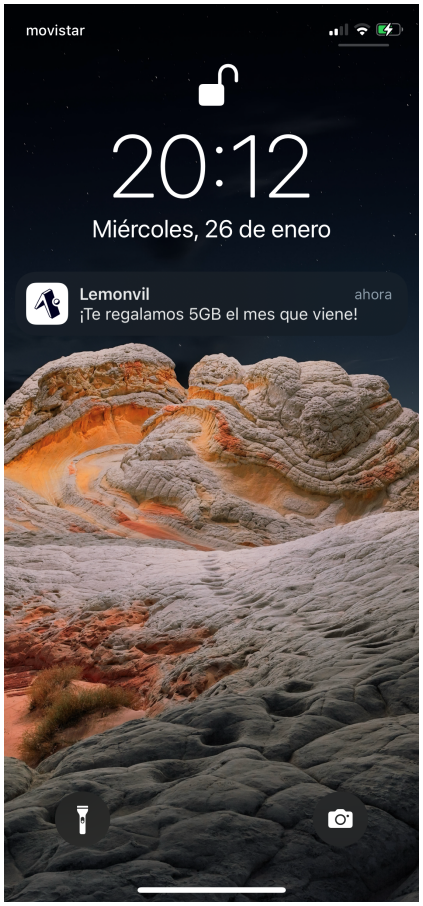


***Ilustración 90 - Ejemplo de notificación recibida***

## 6.12 Pantalla de aviso de pérdida de conexión a Internet

La aplicación cuenta con una funcionalidad que permite detectar si el dispositivo no tiene conexión a internet, para en ese caso, notificar al usuario mediante un mensaje en la parte superior de la pantalla tal y como se aprecia en la Ilustración 91. Este mensaje de alerta aparecerá en toda la aplicación mientras no se disponga de conexión a Internet, y desaparecerá automáticamente cuando el dispositivo recupere la conexión.


*Ilustración 91 - Aviso de pérdida de conexión a Internet*

## 6.13 Pantalla de Carga/Recarga

Cuando la aplicación se (re)carga en el dispositivo, por defecto aparece una pantalla en blanco. Esta pantalla se puede sustituir por otra que sea más acorde a la aplicación para ofrecer una mejor experiencia al usuario. En esta ocasión, se ha diseñado la pantalla que se muestra en la Ilustración 92.

Para hacer este cambio, basta con modificar la propiedad `splash` en el fichero `app.json` indicando la ruta a una nueva imagen que, normalmente, deberá estar situada en el directorio `assets` del proyecto.


***Ilustración 92 - Pantalla de carga/recarga***

# 7 PRUEBAS

*Si buscas resultados distintos, no hagas siempre lo mismo*

*Albert Einstein*

Durante el desarrollo del proyecto se han ido realizando pruebas tanto del lado del cliente como del lado del servidor para asegurar su correcto funcionamiento, tanto en dispositivos Android como iOS.

## 7.1 Pruebas de la aplicación móvil

En el caso de dispositivos Android, las pruebas de funcionamiento se han realizado utilizando el simulador de Android Studio. Concretamente, se ha usado un Google Pixel 4 con SO Android R (versión 11.0). Esta ha sido la principal via de pruebas de la aplicación mientras estaba en desarrollo, ya que Expo proporciona un mecanismo de *hot reloading* el cual permite visualizar automáticamente los cambios que se vayan realizando en un fichero.

Para el caso de dispositivos iOS, las pruebas se han realizado utilizando un terminal físico. En concreto se ha usado un iPhone XS con iOS 15.3. Para ello, se ha hecho uso de la aplicación "Expo Go" que se comentará en el apartado A.2.1 del Anexo A. Con ella, también se han podido observar los cambios en tiempo real cuando se modifica algún fichero.

## 7.2 Pruebas del uso de las API

Al igual que ocurre con la aplicación móvil, también se han realizado pruebas de funcionamiento de los servicios de las API SOAP y REST.

En concreto se ha usado el software *"Postman"* que se comentará en el Anexo B. Con esta herramienta, se ha podido observar la estructura de la respuesta de todos los servicios antes de hacer uso de ellos en la aplicación. El objetivo de esta tarea ha consistido en observar la salida de la llamada a los servicios de ambas API para crear un componente que cargue en pantalla los datos de forma correcta.

# 8 CONCLUSIONES Y MEJORAS

*La inspiración desbloquea el futuro*

*H. Miyazaki*

En este apartado se van a comentar las conclusiones extraídas tras la realización del proyecto. Para ello se irán comentando cada una de las tecnologías utilizadas en el desarrollo de la aplicación, así como el propio *framework* de React Native.

En primer lugar, que React Native use como lenguaje de programación JavaScript/TypeScript da un amplio abanico de posibilidades a la hora del desarrollo. JavaScript se trata de un lenguaje bastante extendido en la actualidad, con multitud de librerías, y además cuenta con una comunidad muy amplia, por lo que es fácil encontrar documentación y tutoriales de ayuda en la red.

React Native ha demostrado ser un potente *framework* para el desarrollo de aplicaciones móviles multiplataforma. Es fácil de aprender ya que básicamente es una librería de JavaScript y el haber trabajado anteriormente con desarrollo Android en Java, me ha permitido desde un primer momento conocer el uso de los componentes y las propiedades. Por todo ello, solo ha sido necesario familiarizarse con algunos aspectos propios de esta librería, tales como los estados, el contexto y la forma en la que se realiza el renderizado de los componentes cuando se cargan o recargan en pantalla.

Expo ha facilitado bastante algunos aspectos del desarrollo, en especial la compilación y las pruebas en tiempo real de la aplicación, por lo que lo convierten en el cliente perfecto para las personas que empiecen a desarrollar con React Native. Sin embargo, tiene bastantes limitaciones, en especial a la hora de utilizar librerías adicionales de React Native, ya que durante el desarrollo de la aplicación me he encontrado que muchas de ellas no son compatibles con Expo y no he podido usarlas. Sin duda, este es el mayor inconveniente que le encuentro, por lo que, si tuviera que desarrollar otra aplicación, al contar ya con cierta experiencia, usaría el cliente propio de React Native en lugar de Expo para evitar estas posibles limitaciones.

En cuanto a la aplicación móvil, he intentado hacerla lo más intuitiva y fácil de usar como fuera posible, cogiendo ideas de aplicaciones de otras operadoras telefónicas. El diseño de la aplicación que se ha presentado en este documento es totalmente distinto al que planteé en un primer momento, ya que inicialmente la consulta de todo lo que no fuera relativo al consumo, ya sea de datos, llamadas o sms, estaba separado en otra pantalla. Por lo que pensando en el usuario que tuviera más de una línea contratada, decidí cambiar el diseño de la aplicación y añadir una pantalla de selección de línea, para una vez seleccionada la que interese gestionar, poder consultar todo lo relativo a esa línea desde un mismo lugar.

Otro aspecto que ha cambiado con respecto al diseño inicial ha sido la inclusión de un tema oscuro en la aplicación. Como línea de mejora futura, podría darse al usuario la opción de elegir entre usar un tema claro u oscuro, como viene siendo cada vez más habitual en aplicaciones. Además, también podría implementarse un *widget* para la pantalla de inicio que permita consultar rápidamente el consumo de datos de la línea que el cliente desee.

Otras líneas de mejora futuras, siempre y cuando sea posible, podría ser la unificación de todos sus servicios. De esta manera, además de controlar el consumo de una línea móvil, también se podría, por ejemplo, controlar y gestionar la fibra contratada para el hogar.

Con respecto a las API SOAP y REST utilizadas, destaco el buen mecanismo de seguridad de la API REST. Considero que JWT proporciona un mecanismo de seguridad muy superior al mecanismo que usa la API SOAP, que no es más que una clave alfanumérica que se pasa como parámetro a cada servicio. JWT también me ha permitido comprobar la expiración del token sin necesidad de hacer una llamada a un servicio de la API REST. Simplemente usando la propiedad *exp* para comprobarlo de forma local sin necesidad de sobrecargar al servidor. Este mecanismo es muy distinto en la API SOAP, en la cual se hace rotundamente necesaria la llamada a un servicio para comprobar en la respuesta si la clave generada ya ha expirado.

Otro aspecto importante en el manejo de ambas API ha sido el formato de la respuesta de la llamada a un servicio. La API REST devuelve la respuesta en formato JSON, por lo que ha sido muy fácil trabajar con los datos. Esta tarea ha sido mucho más tediosa con la API SOAP, ya que para la descripción de sus servicios se emplea WSDL, y, por tanto, XML. Esto ha complicado bastante más acceder a los datos realmente útiles en la respuesta de un servicio, habiendo sido necesarias librerías externas para parsear la respuesta.

Personalmente considero que se han cumplido los objetivos marcados en la introducción de este documento, habiendo desarrollado una aplicación multiplataforma accesible y fácil de usar para todos los clientes que usen la aplicación móvil.

# ANEXO A: INSTALACIÓN DE LAS HERRAMIENTAS NECESARIAS Y DESPLIEGUE

El contenido de este anexo se va a destinar a explicar cómo instalar todas las herramientas necesarias para ejecutar aplicaciones móviles usando React Native y Expo. Se detallará paso a paso el procedimiento a seguir y la forma de ejecutar las aplicaciones tanto en dispositivos físicos como virtuales.

## A.1 Instalación de Node.js

Para poder ejecutar aplicaciones en local en nuestra máquina, en primer lugar, necesitamos hacer uso de Node.js y de npm, su sistema de gestión de paquetes por defecto.

La instalación de Node.js se realiza de forma sencilla accediendo a la web https://nodejs.org/es/download/ y seleccionando la versión apropiada para el sistema operativo que estemos utilizando, en este caso Windows.

Como podemos ver en la Ilustración 93, la descarga de Node.js incluye el gestor de paquetes npm.

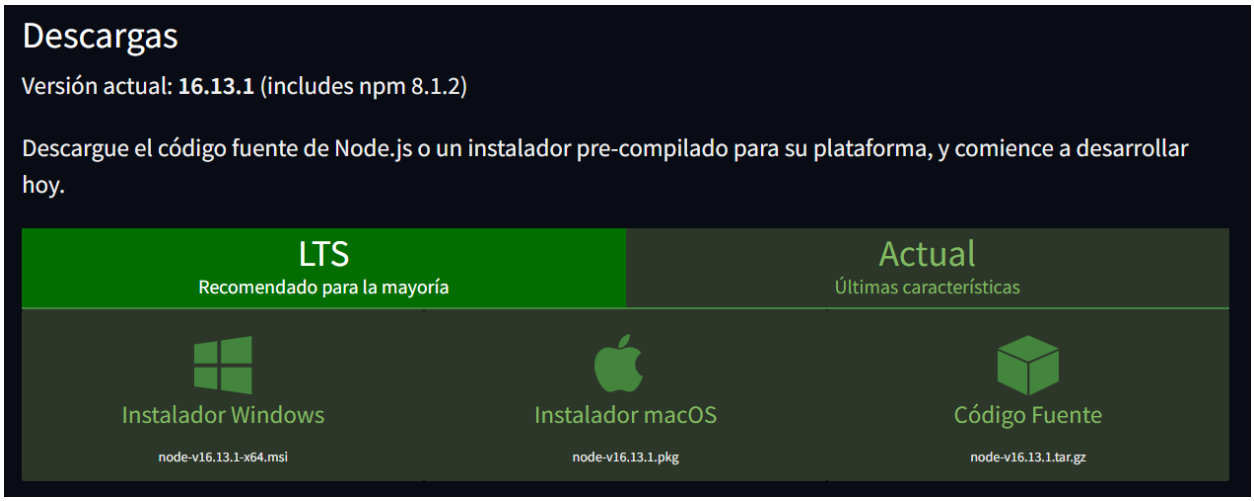


*Ilustración 93 - Web de node.js*

Una vez instalado, comprobaremos la existencia de Node.js y npm en nuestra máquina abriendo un terminal de comandos y ejecutando las siguientes líneas: `node -v` para comprobar la versión de Node.js y `npm -v` para comprobar la versión de npm. En la Ilustración 94 se muestra un ejemplo de ello.

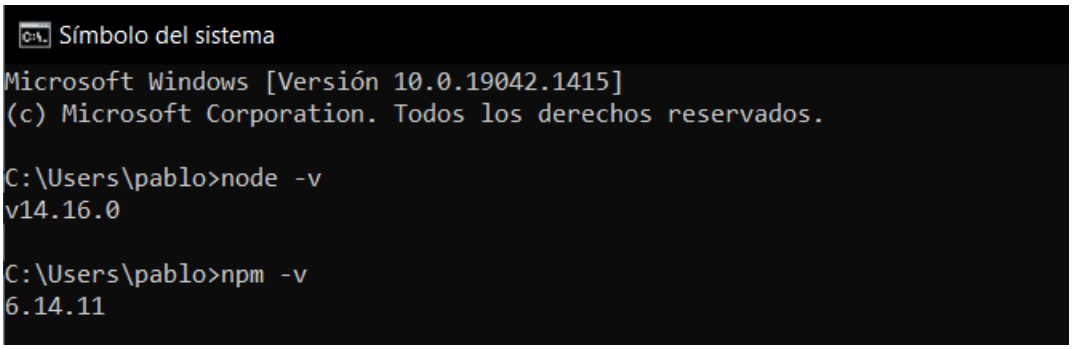


*Ilustración 94 - Versión de node.js y npm*

Es importante destacar que si ya teníamos instalado previamente Node.js en nuestra máquina, nos aseguremos que estemos utilizando al menos la versión 12.0.0 o posteriores, para evitar cualquier tipo de problemas e incompatibilidades.

## A.2 Instalación de Expo

Una vez hemos instalado Node.js, lo siguiente que necesitamos es el cliente Expo. Para su instalación [55] lo único que debemos hacer es ejecutar el siguiente comando en un terminal de comandos: `npm i -g expo-cli`
Es probable que la ejecución del comando dure unos minutos dependiendo de la conexión a Internet y de la potencia de la máquina.

Si estamos usando macOS o Linux, puede que sea necesario prefijar el comando anterior con `sudo` para evitar cualquier problema de permisos.

La ejecución del comando y su salida se muestra en la Ilustración 95, en este caso, se está utilizando la versión 5.0.3 de Expo. Se pueden apreciar varios mensajes de *Warning* que podemos obviar sin problemas, ya que solamente indican que algunas librerías de Expo están obsoletas o que no están soportadas para la versión instalada.

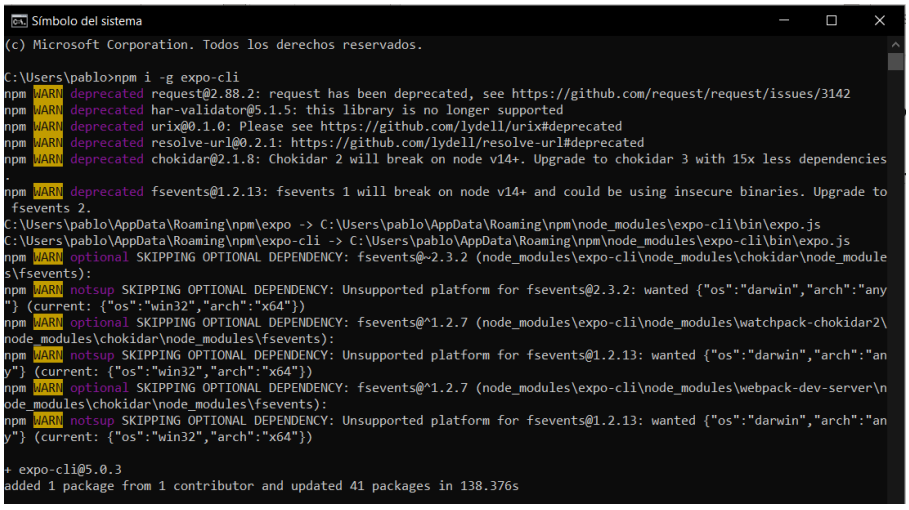


*Ilustración 95 - Instalación de Expo*

### A.2.1 Cliente Expo para el móvil – Expo Go

En el caso de que queramos ejecutar las aplicaciones en un dispositivo físico, es necesario instalar la aplicación de Expo en dicho dispositivo. Esta aplicación la podemos encontrar en la tienda *Google Play* de Android y en la *App Store* de iOS. Los enlaces a la aplicación para cada dispositivo se proporcionan a continuación:

- App para Android: https://play.google.com/store/apps/details?id=host.exp.exponent&referrer=www
- App para iOS: https://apps.apple.com/app/apple-store/id982107779

***Ilustración 96 - Logo de Expo Go***

## A.3 Creación de una nueva aplicación

En este apartado se van a detallar los pasos a seguir para crear un nuevo proyecto desde cero.

En primer lugar, es conveniente crear un nuevo directorio donde situaremos todos los ficheros que van a componer la aplicación. Una vez creado, abrimos un terminal de comandos en dicho directorio y ejecutamos el siguiente comando: `expo init NombreDeLaApp`

Este comando se encarga de crear en el directorio donde estamos situados, un nuevo directorio de nombre `NombreDeLaApp` con todos los ficheros necesarios para la creación de una aplicación usando Expo.

Tras ejecutar dicho comando, debería aparecer lo que se ve en la Ilustración 97.

```
C:\Users\pablo\Desktop\miApp>expo init AppConsumo
? Choose a template: » - Use arrow-keys. Return to submit.
    ----- Managed workflow -----
>   blank                a minimal app as clean as an empty canvas
    blank (TypeScript)   same as blank but with TypeScript configuration
    tabs (TypeScript)    several example screens and tabs using react-navigation and TypeScript
    ----- Bare workflow -----
    minimal              bare and minimal, just the essentials to get you started
```

***Ilustración 97 - creación de una nueva app***

Como se observa en la Ilustración 97, podemos elegir entre dos tipos de flujo de trabajo o *workflow*. Estos tipos son: *Managed Workflow* y *Bare Workflow*. La descripción de cada tipo de flujo de trabajo ya se comentó

en el apartado 4.2. En este caso, se ha optado por un flujo de trabajo *Managed Workflow.*

Dentro del flujo *Managed Workflow*, podemos observar que tenemos tres tipos de plantillas disponibles. En este caso, se ha optado por la primera de ellas: *blank*. La diferencia de esta plantilla con las otras reside en que con esta se empleará JavaScript como lenguaje de programación, mientras que, con las otras, se empleará TypeScript.

Una vez que hemos seleccionado la plantilla, comenzará la instalación de todas las librerías y dependencias de Expo necesarias para la creación de una nueva aplicación. El resultado se muestra en la Ilustración 98.

```
C:\Users\pablo\Desktop\miApp>expo init AppConsumo
√ Choose a template: » blank              a minimal app as clean as an empty canvas
√ Downloaded and extracted project files.
  Using npm to install packages.
√ Installed JavaScript dependencies.

  Your project is ready!

To run your project, navigate to the directory and run one of the following npm commands.

- cd AppConsumo
- npm start # you can open iOS, Android, or web from here, or run them directly with the commands below.
- npm run android
- npm run ios # requires an iOS device or macOS for access to an iOS simulator
- npm run web
```

***Ilustración 98 - Instalación de las dependencias necesarias de Expo***

Finalmente, nos situaremos en el nuevo directorio que se ha creado tras finalizar la instalación. Este directorio tendrá el nombre que se le dio al comando `expo init` que ejecutamos anteriormente. En el caso de la Ilustración 97 el directorio se llama *AppConsumo*. El contenido del directorio se muestra en la Ilustración 99.

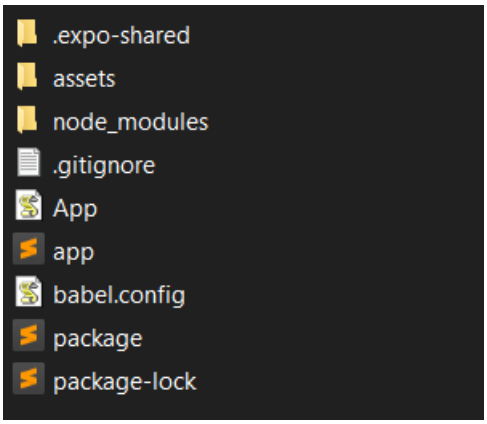


***Ilustración 99 - Ficheros del directorio de trabajo***

## A.4 Ejecución de la Aplicación

Una vez hemos creado un nuevo proyecto siguiendo los pasos comentados en el apartado anterior, se va a pasar a explicar los pasos a seguir para ejecutar la aplicación tanto en un dispositivo virtual como uno físico.

Para ello, abrimos un terminal de comandos y nos situamos en el directorio `NombreDeLaApp` y ejecutamos el siguiente comando: `npm start`

Este comando se encarga de ejecutar el proyecto en local en nuestra máquina. En la Ilustración 100 podemos ver un ejemplo de ello, donde el puerto que se está usando es el 19002.


***Ilustración 100 - Consola de comandos: Ejecución de una app***

La ejecución del comando anterior debería abrir de forma automática el navegador que tengamos asignado por defecto en nuestra máquina, en este caso, con la url http://localhost:19002, como podemos observar en la Ilustración 100. El número de puerto puede variar de una máquina a otra.

En el navegador encontraremos las herramientas de desarrollador, llamadas *Metro Bundler*, entre las cuales tenemos:

- Ejecutar la aplicación en un dispositivo Android físico o virtual,
- Ejecutar la aplicación en un simulador de iOS,
- Ejecutar la aplicación en el navegador web,
- Enviar por email el enlace de desarrollo,
- Publicar la aplicación en un repositorio de Expo,
- Seleccionar el tipo de conexión deseada (LAN por defecto),
- Código QR para un dispositivo físico,
- Ventana con logs de la aplicación

En la Ilustración 101 se puede ver un ejemplo de la interfaz de *Metro Bundler*.

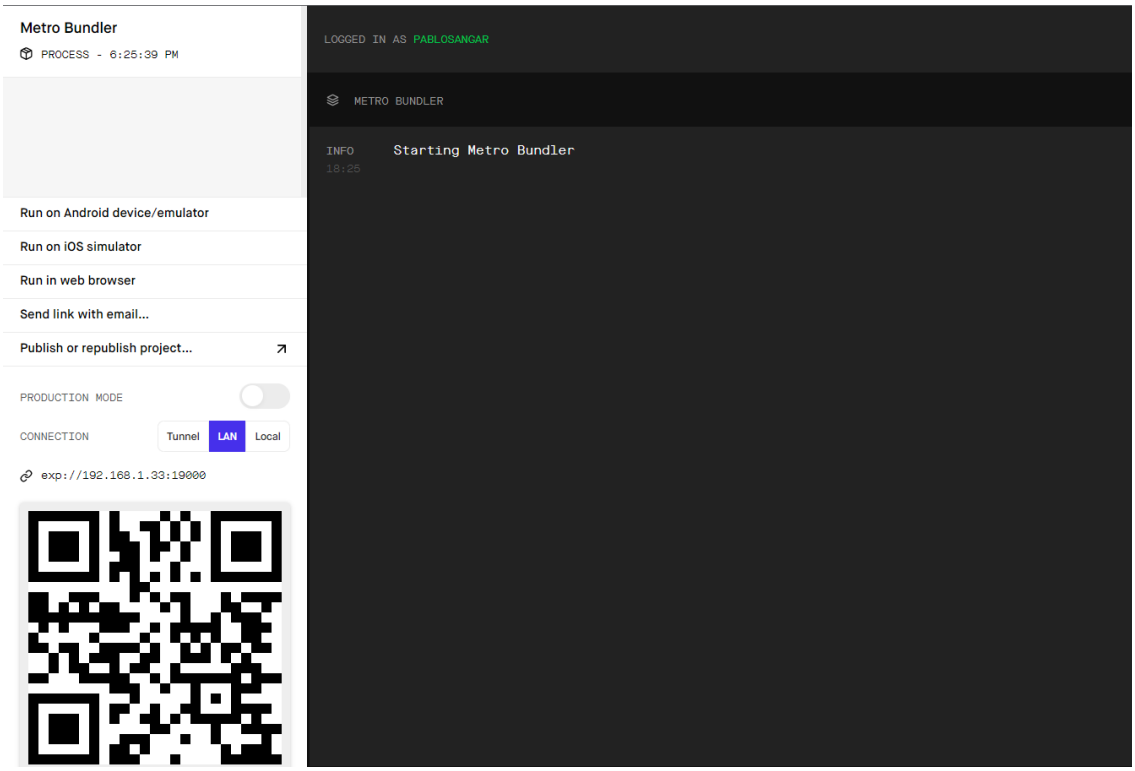


***Ilustración 101 - Metro Bundler***

### A.4.1 Ejecución de la aplicación en Android

Llegados a este punto, podemos optar entre ejecutar la aplicación en un dispositivo Android físico o uno virtual. Las diferentes formas de hacerlo se explican a continuación.

#### *Ejecución en un dispositivo Android físico*

Para ejecutar la aplicación en un dispositivo físico con Android, en primer lugar, será necesario tener instalada la aplicación de Expo en el dispositivo, como ya se comentó en el apartado A.2.1 de este anexo.

Una vez instalada, basta con conectar el dispositivo con el cable correspondiente al ordenador en el que se esté trabajando y pulsar el primer botón del lateral izquierdo llamado *"Run on Android device/emulator"* de las herramientas de desarrollo en la web.

La aplicación comenzará a instalarse en el dispositivo y una vez finalizada la instalación, se ejecutará de forma automática.

NOTA: una vez nos estamos moviendo por la aplicación, para salir de ella o recargarla, basta con agitar el dispositivo para abrir el panel de desarrollo donde encontraremos varias opciones.

### *Ejecución en un dispositivo Android virtual*

Para ejecutar la aplicación en un dispositivo Android virtual, es decir, en un simulador de Android, será necesario tener instalado Android Studio en la máquina en la que se esté trabajando.

Para instalar Android Studio, nos dirigimos a la web https://developer.android.com/studio y elegimos la versión para el sistema operativo en el que se esté trabajando.

Una vez finalizada la instalación, será necesario configurar un dispositivo virtual donde podamos ejecutar las aplicaciones. Para ello, abrimos el programa y seleccionamos la opción *Configure* que podemos encontrar en la parte inferior de la ventana. En la Ilustración 102 se muestra un ejemplo.

Aparecerá un menú desplegable, donde seleccionaremos la opción *AVD Manager* (Android Virtual Device Manager) tal como se muestra en la Ilustración 103.

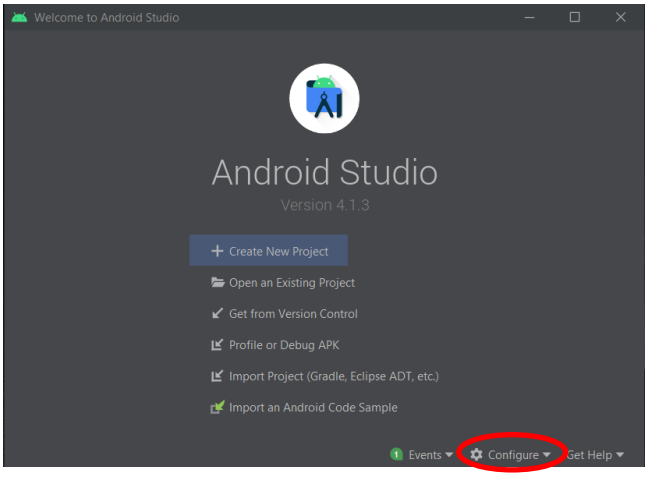


***Ilustración 102 - Ventana Inicial Android Studio***

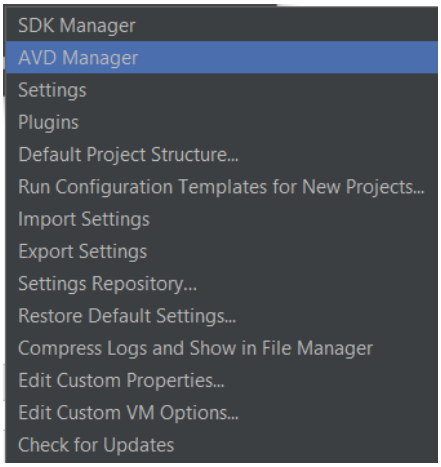


***Ilustración 103 - Menú desplegable Android Studio***

Una vez se ha seleccionado *AVD Manager* aparecerá una nueva ventana donde tendremos un listado de los simuladores que tenemos instalados, y la opción de configurar un nuevo dispositivo virtual, tal como se muestra en la Ilustración 104.

Para la creación de un nuevo dispositivo, pulsamos el botón *Create Virtual Device* y a continuación aparecerá un menú donde podremos configurar el dispositivo según convenga.

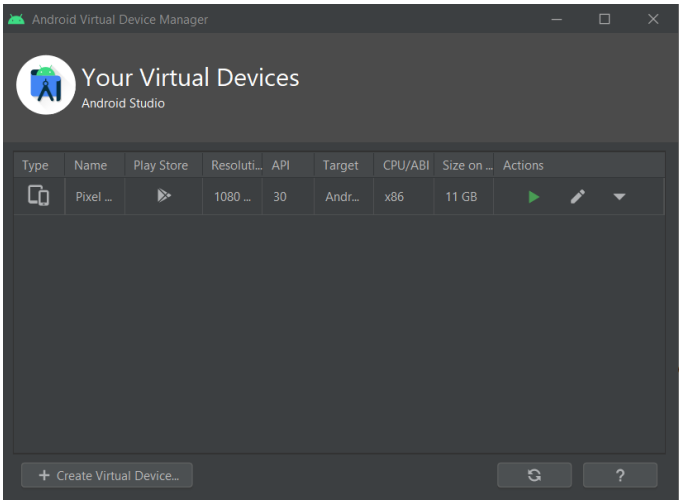


***Ilustración 104 - Android Studio: Dispositivos virtuales***

La primera opción para elegir será el hardware sobre el que se quiere trabajar, en este caso, se ha optado por un *Pixel 4*. A continuación, seleccionaremos la versión de Android que queramos tener instalada en el dispositivo. Es recomendable instalar la última versión estable del sistema, en este caso, *Android R.*

Por último, como se puede observar en la Ilustración 105, podremos poner un nombre al dispositivo que estamos configurando, ver las opciones que hemos elegido previamente a modo de resumen y seleccionar la orientación por defecto para el dispositivo, en este caso, vertical o *Portrait.* Para finalizar, pulsaremos sobre el botón *Finish.*

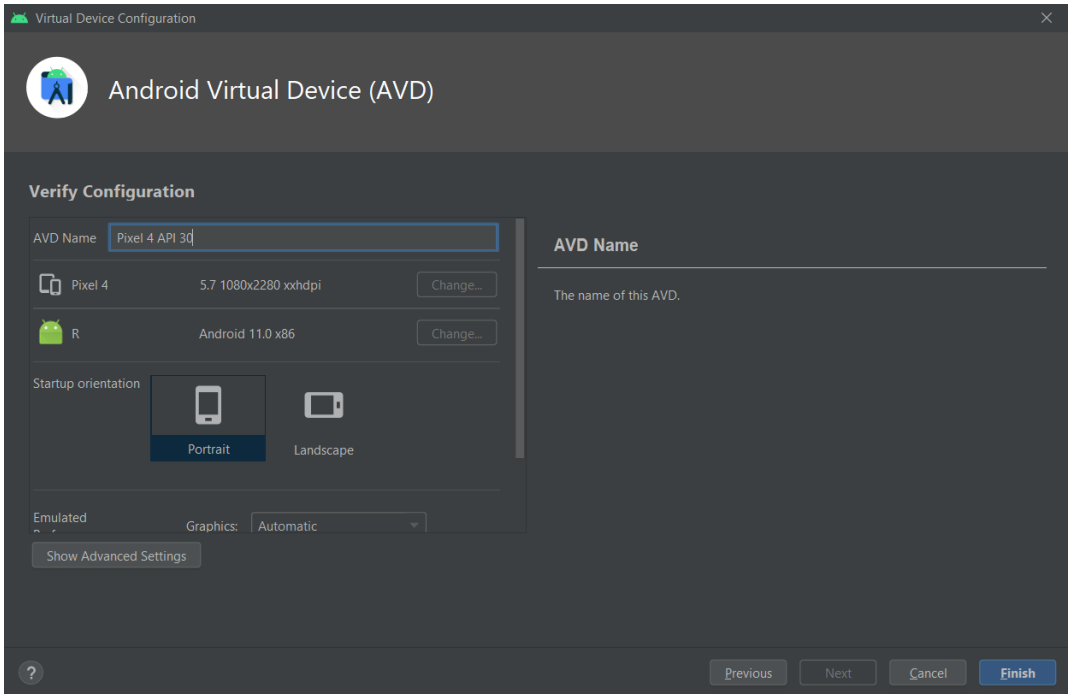


***Ilustración 105 - Creación de un dispositivo virtual***

Una vez se ha creado el dispositivo virtual, para ejecutarlo basta con presionar sobre el botón *Play* de la lista de dispositivos virtuales, tal como se apreciaba en la Ilustración 104.

Con el simulador en marcha, para ejecutar la aplicación, pulsaremos nuevamente sobre el botón *"Run on Android device/emulator"* de las herramientas de desarrollo en la web, al igual que se hizo para el caso de un dispositivo físico, y la aplicación comenzará a cargarse de forma automática en el simulador.

NOTA: en este caso, para abrir las herramientas de desarrollo dentro de la aplicación, tendremos que pulsar *CTRL + M* en Windows, o *Command + M* en macOS.

### A.4.2 Ejecución de la aplicación en iOS

Como se pudo observar anteriormente, las herramientas de desarrollo en la web, solo nos da la posibilidad de ejecutar la aplicación en un simulador de iOS. Sin embargo, existe una forma alternativa para poderla ejecutar en un dispositivo físico con iOS -iPhone o iPad- que se comentará posteriormente.

#### *Ejecución de la aplicación en un simulador de iOS*

Para ejecutar la aplicación en un simulador de iOS, necesitamos una máquina con macOS y tener instalado Xcode.

Xcode lo podemos obtener en la *App Store* de mac o bien, pulsando sobre el siguiente enlace https://apps.apple.com/es/app/xcode/id497799835?mt=12

***Ilustración 106 - Logo de Xcode***

Una vez instalado, para configurar un nuevo dispositivo virtual, pulsaremos sobre la pestaña *Window* en la parte superior, y elegiremos la opción *Devices and Simulators*, tal como se muestra en la Ilustración 107.

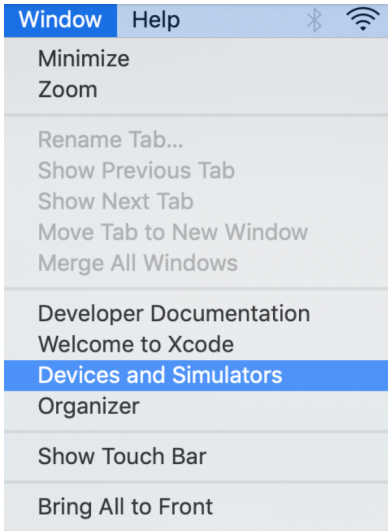


***Ilustración 107 - Creación de un dispositivo virtual iOS (I)***

Seleccionamos la pestaña *Simulators* y hacemos clic sobre el icono + en la parte inferior. Aquí podremos

configurar el modelo del dispositivo que queramos utilizar, así como la versión de iOS. En la Ilustración 108 se muestra un ejemplo de ello. Una vez seleccionado lo que nos interese, finalizamos haciendo clic en *Create.*


***Ilustración 108 - Creación de un dispositivo virtual iOS (II)***

Una vez tengamos configurado el dispositivo, lo pondremos en marcha pulsando sobre el botón *Play* situado en la barra superior.

En esta ocasión, para ejecutar la aplicación, tendremos que pulsar sobre el botón *Run on iOS simulator* en las herramientas de desarrollo en la web. Se instalará de forma automática la aplicación de Expo en el simulador, y se cargará y ejecutará nuestra aplicación.

### *Ejecución de la aplicación en un dispositivo iOS físico*

Como se ha comentado anteriormente, las herramientas de desarrollo en la web no dan la posibilidad directamente de ejecutar la aplicación en un dispositivo iOS físico. Sin embargo, podemos recurrir a otra de las opciones disponibles, que es haciendo uso del código QR situado al final de estas herramientas.

Para ello, lo primero que necesitamos será tener instalada la aplicación de Expo en nuestro dispositivo físico -iPhone o iPad- tal como se comentó en el apartado A.2.1 de este anexo.

Una vez instalada, será necesario situar el dispositivo en la misma red en la que esté la máquina con la que estemos trabajando, y tras ello, bastará con escanear el código QR para que nuestra aplicación se cargue y ejecute en la aplicación de Expo.

NOTA: esta forma de ejecutar la aplicación haciendo uso del código QR, también es válida para un dispositivo Android físico como alternativa a la forma presentada anteriormente.

## A.5 Generación de los ejecutables

En este apartado se va a explicar el procedimiento a seguir para generar los ficheros ejecutables para ser usados en dispositivos Android.

Una vez finalizado un proyecto, para generar el ejecutable *.apk*, lo primero que se debe hacer es modificar el fichero `app.json` y añadir la configuración para Android. Se deberán añadir al menos las propiedades `package` y `versionCode` las cuales permiten definir el nombre del paquete y el número de versión de la aplicación respectivamente. En la siguiente ilustración se muestra un ejemplo.

```
"android": {
    "package": "com.pabsangar.appTFG",
    "versionCode": 1
}
```

***Ilustración 109 - app.json: configuración de Android***

Una vez se ha actualizado este fichero, en una consola de comandos se deberá ejecutar lo siguiente: `expo build:android`. Este comando será el encargado de compilar la aplicación y generar el fichero *.apk*. En el proceso, se darán a elegir entre dos tipos de compilación: `apk` y `app-bundle`. En este caso, se ha elegido la opción `apk`, la cual genera el fichero ejecutable que se puede utilizar tanto en la tienda *Google Play* como directamente en un terminal. La otra opción, `app-bundle`, solamente genera un fichero para ser usado en la tienda de aplicaciones.

A continuación, se pedirán las credenciales o *keystore* para firmar la aplicación. Se podrá seleccionar entre utilizar una clave ya existente o dejar que Expo cree una nueva automáticamente. En este caso, se ha dejado que Expo cree una nueva seleccionando la opción *Generate new keystore*.

En la Ilustración 110 se observan los pasos comentados.

```
√ Choose the build type you would like: » apk
Checking if there is a build in progress...

Accessing credentials for pablosangar in project pruebasTFG
√ Would you like to upload a Keystore or have us generate one for you?
If you don't know what this means, let us generate it! :) » Generate new keystore
The `keytool` utility was not found in your PATH. A new Keystore will be generated on Expo servers
.

› Expo SDK: 41.0.0
› Release channel: default
› Workflow: Managed

Building optimized bundles and generating sourcemaps...
Starting Metro Bundler
Android Bundling complete 67526ms
```

***Ilustración 110 - Proceso de compilación de una app***

Una vez generada la clave, comenzará el proceso de compilación de la aplicación. Este proceso se lleva a cabo en los servidores de Expo, por lo que el tiempo necesario podrá variar en función del tráfico en ese momento.

Para monitorizar la compilación, Expo crea un enlace automáticamente para comprobar el estado, por lo que no es necesario dejar la consola de comandos abierta durante todo el proceso.

Siguiendo el enlace que genera, como se aprecia en la Ilustración 111, el proceso de compilación entra en una cola de espera hasta que algún servidor esté disponible.

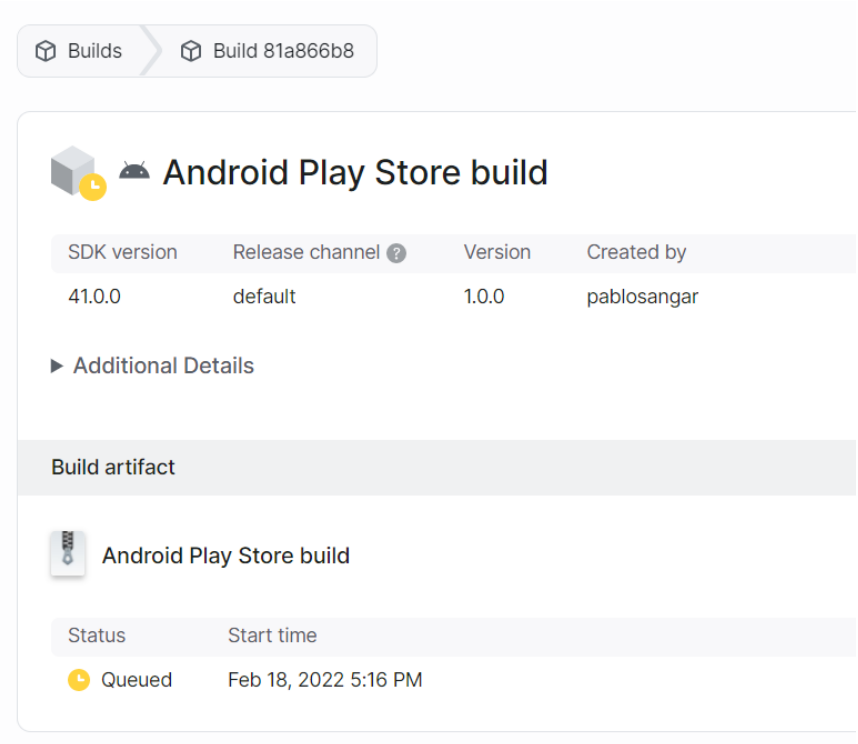


***Ilustración 111 - Compilación de la app***

Una vez que el proceso de compilación ha terminado, el estato (*status*) de la compilación pasará de "en cola" (*Queued)* a finalizado (*Finished).* En este momento, será posible descargar el fichero *.apk* que podrá ser instalado en cualquier terminal Android.

# ANEXO B: SOFTWARE UTILIZADO

En este anexo se van a comentar las herramientas software utilizadas en el desarrollo de la aplicación.

## B.1 VSCode

*Ilustración 112 - Logo de VSCode*

El editor de código que se ha utilizado para el desarrollo de la aplicación ha sido Visual Studio Code, o VSCode [27].

Se trata de un potente editor de código desarrollado por Microsoft y disponible para la mayoría de los sistemas operativos. Entre alguna de sus principales funcionalidades destacan la integración con Git, la personalización tanto de la interfaz de usuario como de los atajos del teclado, y el uso de extensiones para los lenguajes de programación que permiten mejorar la experiencia de desarrollo.

## B.2 Postman


*Ilustración 113 - Logo de Postman*

Postman [28] es un software que actúa como cliente HTTP[11] y permite realizar pruebas con una API. Esta herramienta es la que se ha usado para hacer pruebas con los servicios del *backend* de la aplicación.

Permite configurar ampliamente las peticiones y modificar aspectos como el uso de *tokens* de autorización, las cabeceras utilizadas, el cuerpo de la petición y otros aspectos como la verificación de certificados SSL[12].

## B.3 Git

***Ilustración 114 - Logo de Git***

Git [29] es un software de control de versiones ampliamente utilizado en el desarrollo de software y enfocado a la eficacia y al mantenimiento de las versiones durante el desarrollo de una aplicación.

Git posee una plataforma web llamada *GitHub* que permite, además de lo comentado, la colaboración de más desarrolladores en un proyecto. También ofrece una aplicación de escritorio llamada *GitHub Desktop* que cuenta con una interfaz de usuario, o GUI.

## B.4 Android Studio

***Ilustración 115 - Logo de Android Studio***

Android Studio [30] es el entorno de desarrollo de aplicaciones móviles para Android. Permite el uso de

[11] HTTP son las siglas de HyperText Transfer Protocol
[12] SSL son las siglas de Secure Sockets Layer

lenguajes como Java y Kotlin, además de Flutter, otra tecnología para el desarrollo de aplicaciones móviles multiplataforma que usa el lenguaje Dart.

La herramienta que se ha utilizado de este software es el simulador de Android, el cual permite probar las aplicaciones mientras se desarrollan.

# REFERENCIAS


[1] Instituto Nacional de Estadística, "Encuesta sobre Equipamiento y Uso de Tecnologías de Información Comunicación en los Hogares" 2020. [En línea]. Available: https://www.ine.es/prensa/tich_2020.pdf

[2] MDN Web Docs, "About JavaScript" Mozilla, 2022. [En línea]. Available: https://developer.mozilla.org/en-US/docs/Web/JavaScript/About_JavaScript?redirectlocale=en-US&redirectslug=JavaScript%2FAbout_JavaScript

[3] NodeJS, "Documentación NodeJS", 2022. [En línea]. Available: https://nodejs.org/es/docs/

[4] MDN Web Docs, "Introducing asynchronous JavaScript", Mozilla 2022. [En línea]. Available: https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Asynchronous/Introducing

[5] MDN Web Docs, "Callback Function", Mozilla 2022. [En línea]. Available: https://developer.mozilla.org/en-US/docs/Glossary/Callback_function

[6] MDN Web Docs, "Using Promises", Mozilla 2022. [En línea]. Available: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Using_promises

[7] MDN Web Docs, "Promise.Prototype.then()", Mozilla 2022. [En línea]. Available: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise/then

[8] MDN Web Docs, "async function", Mozilla 2022.[En línea]. Available: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/async_function

[9] React, [En línea]. Available: https://es.reactjs.org/docs/getting-started.html

[10] React, "Componentes y propiedades". React Documents 2022. [En línea]. Available: https://es.reactjs.org/docs/components-and-props.html

[11] React, "Introducing JSX", React Online Documentation 2022. [En línea]. Available: https://en.reactjs.org/docs/introducing-jsx.html

[12] React, "State and Lifecycle", React Online Documentation 2022. [En línea]. Available: https://en.reactjs.org/docs/state-and-lifecycle.html

[13] React, "ComponentDidMount" 2022. [En línea]. Available: https://es.reactjs.org/docs/react-component.html#componentdidmount

[14] React, "ComponentDidUpdate" 2022. [En línea]. Available: https://es.reactjs.org/docs/react-component.html#componentdidupdate

[15] React, "ComponentWillUnmount" 2022. [En línea]. Available: https://es.reactjs.org/docs/react-component.html#componentwillunmount

[16] React Native, "React Fundamentals", Facebook, Inc (Meta) 2022. [En línea]. Available: https://reactnative.dev/docs/intro-react

[17] React Native, [En línea]. Available: https://reactnative.dev/

[18] React Native, "Text Component", React Native Documentation. [En línea]. Available: https://reactnative.dev/docs/text

[19] React Native, "Styles", React Native Online Documentation. [En línea]. Available: https://reactnative.dev/docs/style

[20] React Native, "Introducing Hooks". [En línea]. Available: https://en.reactjs.org/docs/hooks-intro.html

[21] React Native, "Using the Effect Hook", 2022. [En línea]. Available: https://en.reactjs.org/docs/hooks-effect.html

[22] React Native, “Using the State Hook”, 2022. [En línea]. Available: https://en.reactjs.org/docs/hooks-state.html

[23] React, “useContext Hook”, 2022. [En línea]. Available: https://es.reactjs.org/docs/hooks-reference.html#usecontext

[24] Expo. [En línea]. Available: https://expo.dev/

[25] Expo, “Managed Workflow”, 2022. [En línea]. Available: https://docs.expo.dev/introduction/managed-vs-bare/#managed-workflow

[26] Expo, “Bare Workflow”, 2022. [En línea]. Available: https://docs.expo.dev/introduction/managed-vs-bare/#bare-workflow

[27] Visual Studio Code, Microsoft, 2022. [En línea]. Available: https://code.visualstudio.com/

[28] Postman, Postman API Platform, 2022. [En línea]. Available: https://www.postman.com/

[29] Git. [En línea]. Available: https://git-scm.com/

[30] Android Studio, Google, 2022. [En línea]. Available: https://developer.android.com/studio

[31] React Navigation, “Bottom Tabs Navigator”, 2022. [En línea]. Available: https://reactnavigation.org/docs/bottom-tab-navigator/

[32] React Navigation, “Stack Navigator”. [En línea]. Available: https://reactnavigation.org/docs/stack-navigator/

[33] Formik Docs, Formik, 2022. [En línea]. Available: https://formik.org/

[34] npm, Yup, 2022. [En línea]. Available: https://www.npmjs.com/package/yup

[35] Swagger -supported by Smartbear-, “Bearer Authentication”, 2022. [En línea]. Available: https://swagger.io/docs/specification/authentication/bearer-authentication/

[36] npm, “jwt-decode”, 2021. [En línea]. Available: https://www.npmjs.com/package/jwt-decode

[37] React Native, “ScrollView, 2022. [En línea]. Available: https://reactnative.dev/docs/scrollview

[38] React Native, “TouchableOpacity”, React Native Components 2022. [En línea]. Available: https://reactnative.dev/docs/touchableopacity

[39] GitHub, “React Native Swiper”, leecade, 2019. [En línea]. Available: https://github.com/leecade/react-native-swiper

[40] React Native, “Linking”. [En línea]. Available: https://reactnative.dev/docs/linking

[41] React Native, “FlatList”. [En línea]. Available: https://reactnative.dev/docs/flatlist

[42] Expo, “FileSystem”. [En línea]. Available: https://docs.expo.dev/versions/latest/sdk/filesystem/

[43] React Native, “Alert”. [En línea]. Available: https://reactnative.dev/docs/alert

[44] npm, “React Native Chart Kit”, 2022. [En línea]. Available: https://www.npmjs.com/package/react-native-chart-kit

[45] Lottie Files, “Free Animations”, 2022. [En línea]. Available: https://lottiefiles.com/featured

[46] Expo, “Lottie”, 2022. [En línea]. Available: https://docs.expo.dev/versions/latest/sdk/lottie/

[47] Expo, “ImagePicker”, 2022. [En línea]. Available: https://docs.expo.dev/versions/latest/sdk/imagepicker/

[48] React Native, “Modal”, 2022. [En línea]. Available: https://reactnative.dev/docs/modal

[49] Firebase, Firebase Cloud Messaging, Google, 2022. [En línea]. Available: https://firebase.google.com/docs/cloud-messaging

[50] AWS, Amazon Simple Notification Service, Amazon, 2022. [En línea]. Available: https://aws.amazon.com/es/sns/?whats-new-cards.sort-by=item.additionalFields.postDateTime&whats-new-cards.sort-order=desc

[51] Expo, Push Notifications Overview, 2022. [En línea]. Available: https://docs.expo.dev/push-notifications/overview/

[52] Expo, Notifications, 2022. [En línea]. Available: https://docs.expo.dev/versions/latest/sdk/notifications/

[53] Expo, Push Notifications Tool. [En línea]. Available: https://expo.dev/notifications

[54] GitHub, "React Native NetInfo", 2022. [En línea]. Available: https://github.com/react-native-netinfo/react-native-netinfo

[55] GitHub, "ApiSauce", infinitered, 2022. [En línea]. Available: https://github.com/infinitered/apisauce

[56] Expo, "Introduction to Expo", 2022. [En línea]. Available: https://docs.expo.dev/