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

BookStand: Aplicación Multiplataforma de Gestión de Bibliotecas Personales basada en React Native, Redux, Firebase y Cloud Messaging

Autor: Aurora Salvador Gutiérrez

Tutor: Teresa Ariza Gómez

Dpto. de Ingeniería Telemática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2021

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

BookStand: Aplicación Multiplataforma de Gestión de Bibliotecas Personales basada en React Native, Redux, Firebase y Cloud Messaging

Autor:
Aurora Salvador Gutiérrez

Tutor:
Teresa Ariza Gómez
Profesora titular

Dpto. de Ingeniería Telemática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2021

Trabajo de Fin de Grado: BookStand: Aplicación Multiplataforma de Gestión de Bibliotecas Personales basada en React Native, Redux, Firebase y Cloud Messaging

Autor: Aurora Salvador Gutiérrez

Tutor: Teresa Ariza Gómez

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

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2021

El Secretario del Tribunal

A mi familia

A mis maestros

Agradecimientos

Me gustaría dedicar unas líneas para agradecer a todas aquellas personas que han estado acompañándome a lo largo de estos meses tan extraños y difíciles, y que han hecho posible este proyecto.

Para empezar, a Teresa, por haberme guiado y orientado durante la realización de este proyecto.

A todos mis compañeros que me han apoyado día a día y con los que he compartido recorrido por la universidad.

Por último, pero no menos importante, a mi familia y a mis amigos. Por apoyarme de forma incondicional y estar ahí para darme fuerzas en los momentos más críticos. Por creer en mí y animarme a seguir esforzándome al máximo. Habéis sido un pilar fundamental en mi vida, no solo durante estos últimos meses, sino a lo largo de toda la carrera.

Gracias por formar parte de mi vida.

Aurora Salvador Gutiérrez
Sevilla, 2021

Resumen

Desde que los teléfonos inteligentes llegaron a nuestras vidas, hemos sido testigos de un cambio a la hora de relacionarnos con nuestro alrededor. Muchas de las tareas cotidianas han pasado a formar parte del mundo digital, al que accedemos a través de las aplicaciones móviles o "apps". Por este motivo, hoy en día, el desarrollo de aplicaciones se ha convertido en una necesidad para cualquier negocio que pretenda ofrecer algún servicio a sus usuarios.

Además, debido a la gran diversidad de sistemas operativos que existen actualmente en el mercado, los desarrolladores deben atender a cada una de las plataformas, lo que incrementa de forma considerable tanto el tiempo como los recursos necesarios para crear estas aplicaciones.

Afortunadamente, en los últimos años han aparecido nuevos marcos de desarrollo que permiten crear apps para distintos sistemas operativos a partir del mismo código fuente.

A lo largo de este documento se pretende analizar una de estas tecnologías, React Native, a través de la creación de una aplicación para las gestión de bibliotecas personales. De forma adicional, se analizará su integración con Firebase, un entorno de servidor ofrecido por Google pensado para este tipo de software.

Abstract

Since smartphones came into our lives, we have witnessed a change in the way that we interact with our surroundings. Many of the daily tasks have become part of the digital world, which we access through mobile applications. For this reason, nowadays, application development has become a necessity for any business that intends to offer a service to its users.

In addition, due to the great diversity of operating systems that currently exist on the market, developers must attend to each of the platforms, and therefore increase both the time and the resources required to create these applications.

In recent years new frameworks have emerged and allowed the development of apps for different operating systems from a single codebase.

The aim of this document is to analyze one of these technologies, React Native, through the creation of an application for the management of personal book libraries. In addition, we will also study its integration with Firebase, a server environment offered by Google, designed for this kind of software.

Índice

ÍNDICE DE TABLAS

ÍNDICE DE ILUSTRACIONES

1 INTRODUCCIÓN

1.1 Motivación

La llegada de los teléfonos inteligentes a nuestras vidas supuso una revolución en la forma que las personas interactuamos y nos relacionamos con nuestro entorno. Se han convertido en la forma favorita por los usuarios de llevar a cabo actividades de índole diversa, desde realizar listas de la compra o pagar facturas hasta pedir comida a domicilio.

Por motivos como este, las aplicaciones móviles son esenciales para cualquier negocio hoy en día. No solo mejoran la accesibilidad de los clientes al servicio, sino que también actúan como una herramienta muy poderosa de publicidad y una valiosa fuente de datos e información sobre los usuarios [1].

A la hora de desarrollar una aplicación móvil, o *app* (del inglés *Application*), nos interesa maximizar el número de usuarios finales que podrán hacer uso de ella. Observando el cupo de mercado de los sistemas operativos móviles en el mundo [2] que aparece en la Ilustración 1, resulta evidente que centrándose en el desarrollo de una aplicación para Android y otra para iOS conseguiríamos llegar casi a la totalidad de los usuarios.

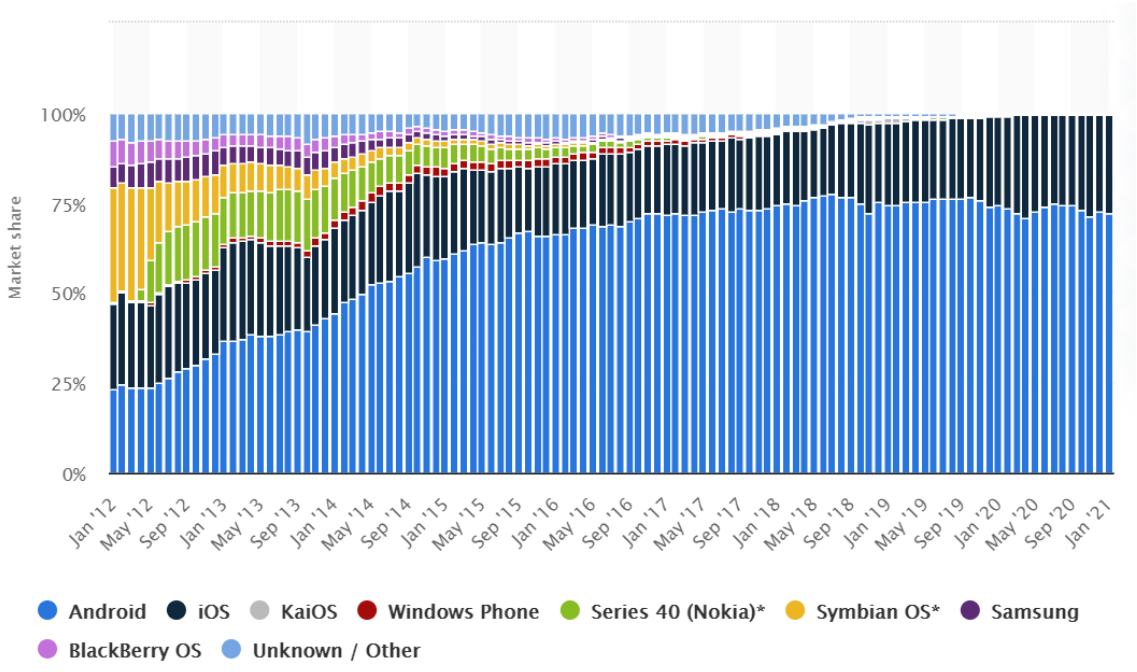

Ilustración 1 Cupo de mercado de los sistemas operativos móviles en el mundo desde 2012 a 2021 (Statista 2021)

Sin embargo, el desarrollo tradicional de aplicaciones móviles para conseguir que un servicio esté disponible en ambos sistemas operativos conlleva la creación de una aplicación diferente para cada uno de ellos. Es indudable que tener que crear múltiples aplicaciones con plataformas y entornos de desarrollo dispares supone un aumento en los costes y recursos de un negocio que en muchas ocasiones no son asumibles.

Afortunadamente, en los últimos años hemos visto una proliferación de nuevos marcos de desarrollo que facilitan la creación aplicaciones para varias plataformas a partir de un mismo código. Comienza la era del desarrollo multiplataforma.

Actualmente encontramos un amplio mercado de entornos de desarrollo multiplataforma para aplicaciones móviles, siendo las opciones más relevantes las siguientes:

- Flutter, la propuesta de Google basada en el lenguaje Dart.
- Xamarin, la propuesta de Microsoft basada en el lenguaje C#.
- Apache Cordova, la propuesta de la fundación Apache basada en HTML, CSS y JavaScript.
- React Native, la propuesta de Facebook basada en JavaScript.

En este documento se explora esta última alternativa, React Native, así como su integración con el entorno de servidor ofrecido por Google, Firebase.

1.2 Objetivos

En base a lo expuesto anteriormente, los objetivos con los que se realiza este proyecto se basan en conocer las capacidades de React Native para desarrollo de aplicaciones móviles de uso real, teniendo en cuenta factores como:

- Facilidad de aprendizaje
- Determinar la capacidad para realizar tareas avanzadas, como la integración de técnicas de Machine Learning (escaneo/reconocimiento de códigos de barras)
- Conocer el nivel de integración con el entorno de servidor de terceros Firebase (base de datos, autenticación, almacenamiento de imágenes, reglas de seguridad, integración con Machine Learning …)
- Eficiencia de la multiplataforma. ¿Qué cambios hay que realizar para adaptar la aplicación a ambos sistemas operativos (Android y iOS)?
- Ventajas e inconvenientes de trabajar con herramientas multiplataformas. ¿Cómo afecta este desarrollo a la calidad de la aplicación? ¿Qué diferencias hubiese habido si, en lugar de realizar la aplicación en React Native, la hubiésemos programado para Android o para iOS?

Por otro lado, dado que se pretende utilizar Firebase como parte del servidor, conviene analizar las características de este servicio:

- Facilidad de instalación y uso
- Diversos módulos usados, con especial énfasis en:
    - Autenticación
    - Machine Learning (escaneo de códigos de barras)
    - Base de Datos (Firestore)
    - Funciones de Cloud (eventos)
- Plan de pago

1.3 Descripción de la solución

Para poder probar las capacidades de este marco de desarrollo, se propone una aplicación pensada para el inventariado y la organización de los libros físicos de una persona: BookStand. Además, como elemento principal de la aplicación, se permite llevar un control de los libros que un usuario decide prestar a otro, mediante notificaciones y mensajes enviados entre dichos usuarios. En la Ilustración 2 que aparece a

continuación se muestra la arquitectura básica de la app:

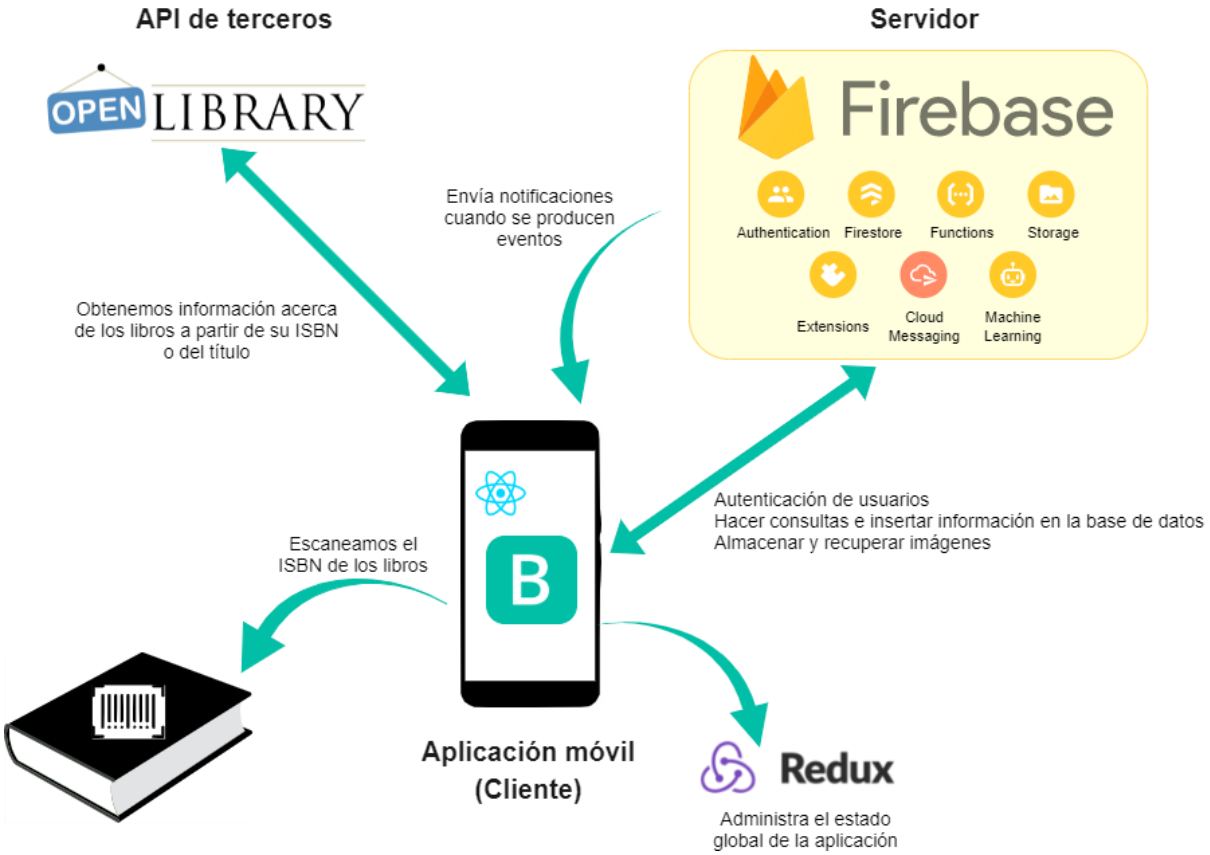

Ilustración 2 Arquitectura general de la aplicación

Como se puede apreciar, se trata de una aplicación móvil compleja, que incorpora las siguientes características:

- Llamadas a una API (del inglés Application Programming Interface, o Interfaz de Programación de Aplicaciones en español) para obtener datos
- Uso de Firebase como servidor, lo que nos permite, entre otras cosas:
  - Almacenar y obtener datos de una base de datos
  - Almacenar multimedia
  - Inicio de sesión con credenciales de Google
  - Enviar notificaciones y mensajes entre dispositivos
- Uso de Redux como patrón para administrar el estado de la aplicación
- Acceso a la cámara del dispositivo
- Lector de códigos de barras
- Notificaciones de recordatorio

1.4 Estructura de la memoria

A continuación, me dispongo a comentar la estructura de este documento para que sirva de guía al lector:

1. Introducción. Se trata del capítulo en el que nos encontramos. Pretende ser un preámbulo en el que se expongan las ideas generales que se tratarán en este documento.

2. Tecnologías utilizadas. En este capítulo se exponen las principales tecnologías en las que se apoya la aplicación móvil a desarrollar, con el objetivo de proporcionar el contexto suficiente al lector para comprender las bases de la aplicación.
3. Herramientas y recursos utilizados. Este capítulo presenta, de forma breve, todas las herramientas y recursos utilizados durante el desarrollo de la aplicación.
4. Arquitectura y análisis. El objetivo de este capítulo es facilitar al lector un conjunto de diagramas y casos de uso para ilustrar la funcionalidad y la estructura de la aplicación móvil.
5. Interfaz de usuario y funcionalidad. En este capítulo exploraremos la aplicación desde el punto de vista del usuario final, mostrando las pantallas y los elementos con los que interaccionará el usuario.
6. Pruebas. Tal y como su nombre indica, el capítulo cubrirá el procedimiento que se ha seguido para realizar pruebas tanto en los emuladores como en los dispositivos físicos.
7. Líneas de mejora y conclusiones. Finalmente, este capítulo abarcará los resultados y conclusiones extraídos tras haber trabajado con las tecnologías mencionadas anteriormente. Asimismo, se pretende ser crítico y analizar las posibles mejoras que se podrían realizar.

2 TECNOLOGÍAS UTILIZADAS

Cualquier tecnología suficientemente avanzada es indistinguible de la magia.

Arthur C. Clarke

2.1 React Native

Ilustración 3 Logo de React Native

React Native [3] es un marco de desarrollo para la creación de aplicaciones móviles multiplataforma elaborado por Facebook. Está basado en la librería React [4], creada por la misma empresa. Para poder comprender mejor el funcionamiento de React Native es necesario comentar las principales características de React para luego explicar cuáles son las diferencias entre ambos marcos.

2.1.1 React

React es una librería JavaScript pensada para construir interfaces de usuario web de forma declarativa. Esto quiere decir que nuestro código se basará en describir qué se quiere obtener en lugar de ocuparse del cómo se obtienen.

Para poder describir de forma declarativa, React divide la interfaz de usuario en elementos independientes llamados componentes. Estos componentes nos permiten aislar y abstraer los elementos de nuestra interfaz, de forma que podamos reutilizarlos en otras partes de nuestro proyecto.

Para describir los componentes, React usa la sintaxis JSX, una extensión de JavaScript basada en etiquetas y pensada para describir la interfaz de usuario (IU). En la Ilustración 4 podemos ver un ejemplo de esta notación para escribir un componente que representa un encabezado de nivel 1.

```
const elemento = <h1>¡Hola mundo!</h1>
```

Ilustración 4 Sintaxis JSX

Un componente puede hacer referencia a otro o a varios componentes en su salida, lo que nos proporciona diversos niveles de abstracción: podemos crear un componente que represente una imagen, otro componente que represente una cadena de texto y un tercer componente que represente a ambos cuando lo invoquemos. Esto lo podemos apreciar en la Ilustración 5:

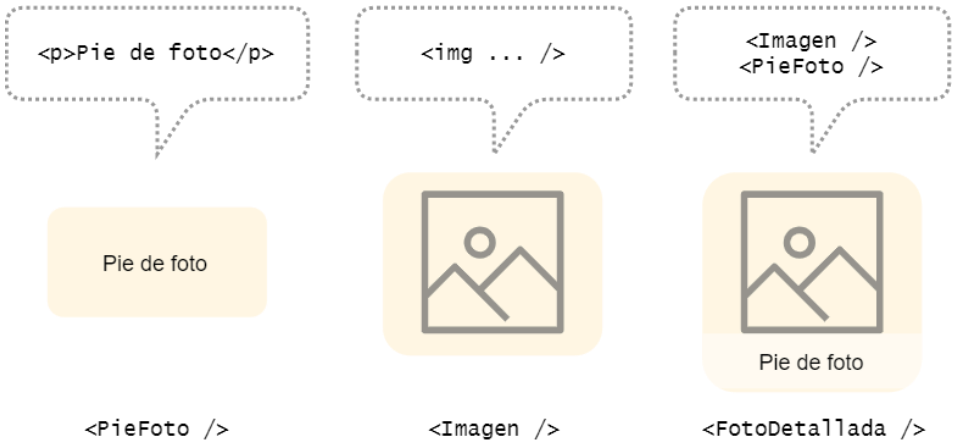

Ilustración 5 Componentes React

De forma conceptual, los componentes no son más que simples funciones JavaScript que aceptan una serie de propiedades o *props* y devuelven el elemento React que es el que finalmente se representará. Existen dos formas de definir los componentes: mediante funciones y mediante clases; y se denominan respectivamente componentes funcionales o sin estado y componentes de clase. En la Ilustración 6 y la Ilustración 7 podemos ver la sintaxis de ambos tipos de componentes, respectivamente.

```
function Bienvenida(props) {
  return <h1>Hola, {props.nombre}</h1>;
}
```

Ilustración 6 Componente funcional

```
class Bienvenida extends React.Component {
  render() {
    return <h1>Hola, {this.props.nombre}</h1>;
  }
}
```

Ilustración 7 Componente de clase basado en ES6 [5]

Desde el punto de vista de React, ambos componentes son capaces de realizar las mismas funciones, pero la forma de trabajar con ellos es diferente, principalmente a la hora de manejar el estado y el ciclo de vida.

React se originó con los componentes de clase e introdujo los funcionales más tarde. Aunque se pueden usar ambos tipos de componentes en una misma aplicación, en este proyecto me centraré en los funcionales, por ser los más modernos y presentar una sintaxis más sencilla.

Como se introdujo en líneas anteriores, los componentes funcionales también se denominan componentes sin estado. Esto se debe porque, a diferencia de los componentes de clase, carecen de un objeto de estado que almacene los valores de las propiedades del componente y que actualice su representación. En base a esto, ¿cómo somos capaces de manejar el estado de un componente sin utilizar clases? Mediante los denominados *Hooks*.

Los Hooks (del inglés *hook*, que podemos traducir como "gancho") son funciones que permiten relacionar ("enganchar") comportamiento que depende del estado a los componentes funcionales. Los principales hooks

que proporciona React son `useState` y `useEffect`. En la Ilustración 8 se puede ver un ejemplo del uso de estos hooks:

```
import React, { useState, useEffect } from 'react';

function EjemploHooks() {
  // Declaración de una variable de estado que llamaremos "contador"
  const [contador, setContador] = useState(0);

  // De forma similar a componentDidMount y componentDidUpdate
  useEffect(() => {
    // Actualiza el título del documento usando la API del navegador
    document.title = `Has hecho click ${contador} veces`;
  });

  return (
    <div>
      <p>Has hecho click {contador} veces</p>
      <button onClick={() => setContador(contador + 1)}>
        Haz click
      </button>
    </div>
  );
}
```

Ilustración 8 Ejemplo de uso de los hooks

El hook `useState` [6] nos permite definir propiedades como estado de un componente, de forma que, al cambiar de valor, hacen que el componente se vuelva a actualizar con el nuevo valor de la propiedad.

Por su parte, el hook `useEffect` [7] es una función a la que indicamos diversas acciones o efectos que tiene que realizar el componente después de renderizarse. Este hook nos permite definir qué tendrá que hacer el componente cuando se haya montado, desmontado o actualizado; de forma que controlemos su ciclo de vida.

Cabe mencionar que el estado de un componente se designa como local o encapsulado [8]. Esto significa que no es accesible desde otro componente excepto para aquel que lo posee y lo asigna. Sin embargo, un componente puede pasar su estado como props a sus componentes hijos, de forma que el flujo de datos en una aplicación es siempre descendiente o unidireccional: de padres a hijos.

En ocasiones, esta estrategia a la hora de organizar el estado de una aplicación puede resultar demasiado tediosa cuando trabajemos con datos que deban ser accesibles desde varias partes de una aplicación. Para estos casos, en lugar de tener que pasar el estado como propiedades a cada uno de estos componentes, se utiliza un contenedor de estado para la aplicación, proporcionado por la librería Redux. Trataremos más acerca de este tema más adelante.

2.1.2 Descripción y características generales

Una vez que conocemos las ideas principales de React, podemos definir React Native como una extensión de React que utiliza componentes nativos en lugar de componentes web para construir las aplicaciones.

Los componentes nativos [9] son las vistas que componen las aplicaciones, tanto en Android como en iOS. Cada plataforma tiene sus propios elementos que son los que se generan cuando escribimos código en Java o Kotlin para Android o en Swift u Objective-C para iOS. Podemos ver esta idea en la Ilustración 9:

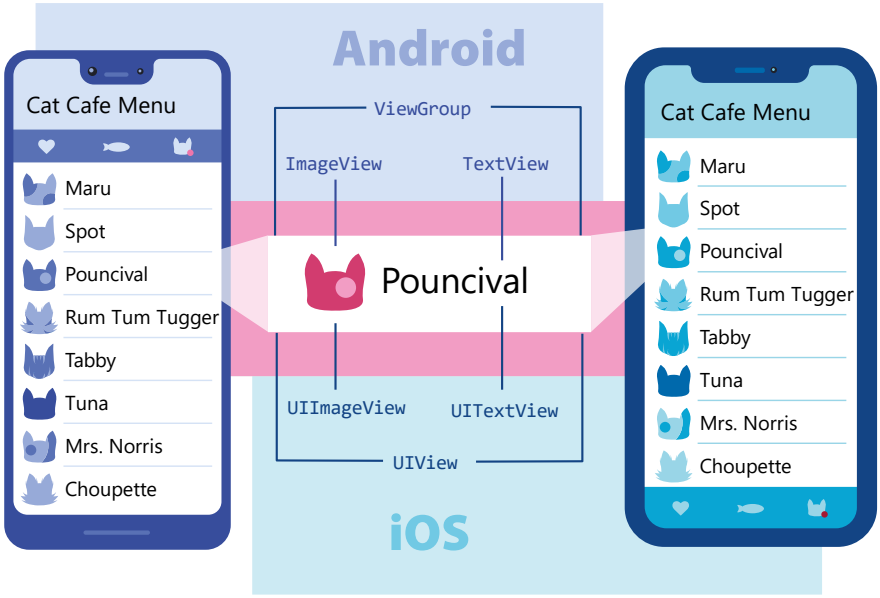

Ilustración 9 Ejemplo de diferentes vistas usadas en Android y iOS (React Native, 2021)

Con React Native, a través de JavaScript invocamos las vistas correspondientes para cada plataforma a través de componentes React. Estos componentes pueden ser los que nos proporciona directamente React Native (llamados componentes del core), los que nos proporcione la comunidad o los que definamos nosotros mismos.

2.1.3 Sintaxis y lenguaje

Para escribir una aplicación en React Native, se usa la sintaxis JSX tanto con JavaScript como TypeScript [10], un superconjunto de JavaScript que proporciona tipos estáticos.

Pese a que JavaScript es la opción más rápida y sencilla, he decidido usar TypeScript en esta aplicación por las ventajas que aporta a la hora de crear un código más robusto y que permite detectar errores antes de ejecutar el código.

El código escrito con JSX se almacena en ficheros con extensión `.jsx` para JavaScript y `.tsx` para TypeScript.

A continuación, en la Ilustración 10, muestro un extracto del código de la aplicación en el que se puede apreciar el uso de TypeScript en conjunto con JSX para crear un componente. Además, podemos observar cómo modificamos el estilo del componente mediante la propiedad "`style`" del componente nativo "`Image`".

```tsx
import React from 'react';
import {View, Image, StyleSheet} from 'react-native';
import {Usuario} from '../types';

// Componente que representa la foto de perfil de un usuario
export function Foto(props: {usuario: Usuario; size: string}) {
    return (
        <View>
            {/* Si el usuario tiene una foto definida, mostrarla */}
            {props.usuario?.foto && (
                <Image
                    style={props.size === 'large' ? styles.large : styles.small}
                    source={{uri: props.usuario?.foto?.toString()}}
                />
            )}
            {/* Si el usuario no tiene una foto, mostrar la foto por defecto*/}
            {!props.usuario.foto && (
                <Image
                    style={props.size === 'large' ? styles.large : styles.small}
                    source={require('../assets/images/avatar.png')}
                />
            )}
        </View>
    );
}

/* Para modificar la apariencia de los elementos utilizamos un objeto StyleSheet
que almacena el estilo de los componentes. Las propiedades suelen coincidir con
las que se usan en CSS, con la diferencia de que están escritas en camelCase */
const styles = StyleSheet.create({
    small: {
        width: 40,
        height: 40,
        borderRadius: 250,
    },
    large: {
        width: 250,
        height: 250,
        borderRadius: 250,
    },
});
```

Ilustración 10 Ejemplo de sintaxis JSX con TypeScript. Fichero `Foto.tsx`.

2.1.4 Instalación de módulos

Como comentábamos unas líneas antes, para escribir el código de nuestra aplicación podemos utilizar componentes React proporcionados por terceros o por la comunidad. Por lo general, estos componentes los instalamos a través de npm [11] (Node Package Manager)[1], un gestor de paquetes para Node.

Esto nos permite añadir funcionalidad a nuestra aplicación de forma rápida y sencilla, utilizando componentes y librerías externas para resolver problemas de diversa índole como la navegación por las pantallas (React Navigation [12]), manejar el estado global (Redux [13]) o conectar la aplicación con el servidor de Firebase (React Native Firebase [14]).

[1] De forma alternativa, podemos usar Yarn, otro gestor de paquetes compatible.

2.2 Firebase

Ilustración 11 Logo de Firebase

Firebase [15] es un servicio ofrecido por Google que aporta un conjunto de herramientas para la creación, el mantenimiento y el control tanto de aplicaciones móviles como web.

2.2.1 Descripción y características generales

Firebase ofrece un conjunto de módulos y funcionalidades [16] para conseguir diferentes objetivos: autenticar usuarios, envío de mensajes entre dispositivos o almacenar archivos e información en sus bases de datos. Todo esto lo convierte en una opción muy atractiva para los desarrolladores a la hora de crear sus aplicaciones móviles *serverless*[2].

Todos estos módulos y la configuración global del proyecto se controlan desde la consola web de Firebase:

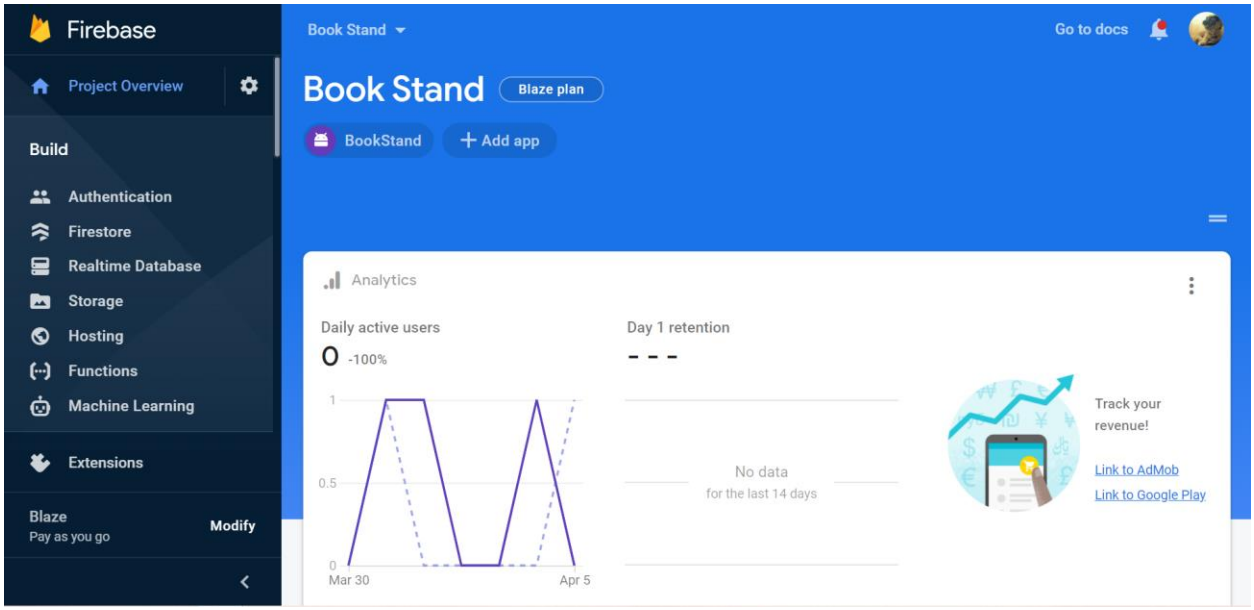

Ilustración 12 Consola de Firebase

La mayoría de estas herramientas se ofrecen de forma gratuita a través de su plan Spark, pero para tener acceso a algunos módulos como las funciones de Cloud, es necesario actualizar a un plan de pago.

En este proyecto se usa el plan Blaze [17], un plan de pago por uso en el que se cobra al desarrollador una vez se ven superados unos límites mensuales. Estos límites han resultado ser lo suficiente generosos como para que, a final de mes, el precio por uso del servicio siga siendo gratuito.

Para poder usar Firebase en la aplicación, se ha hecho uso del ya mencionado paquete React Native Firebase [14].

2.2.2 Módulos utilizados

De todos los módulos y productos que proporciona Firebase, en este proyecto se han usado los siguientes:

[2] La arquitectura serverless o "sin servidor" no quiere decir que no existan servidores, sino que permite a los desarrolladores centrarse en el código del cliente, puesto que los servidores se manejan de forma automática, en este caso por Google.

Ilustración 13 Módulos y productos de Firebase utilizados

2.2.2.1 Authentication

Este módulo proporciona una solución de identidad de extremo a extremo para crear sistemas de autenticación seguros y que logren mejorar la experiencia del usuario final. Permite crear y autenticar usuarios en el sistema mediante diversos métodos como a través de correo y contraseña, números de teléfono, cuentas de Google o de terceros (Twitter, GitHub o Facebook entre otros) [18].

En este proyecto se ha hecho uso de los métodos de inicio por correo y contraseña y de acceso con cuenta de Google para autenticar a los usuarios.

2.2.2.2 Firestore

Firestore, también conocido como Cloud Firestore, es una base de datos de documentos NoSQL que permite el almacenamiento, la sincronización y la consulta de datos desde aplicaciones móviles o la web. Los datos se estructuran de forma jerárquica en colecciones y documentos sobre los que se pueden realizar consultas simples o de complejas [19].

Este producto viene equipado con un conjunto de reglas de seguridad que permiten controlar el acceso a los datos que se almacenan.

Este módulo se ha usado como base de datos para almacenar toda la información de usuario de la aplicación. En el capítulo 4 de este documento podrán encontrarse más detalles acerca de la estructura de los datos que se ha almacenado en Firestore.

2.2.2.3 Cloud Functions

Como hemos comentado, Firebase proporciona una arquitectura “sin servidores” en nuestra aplicación. Sin embargo, con frecuencia necesitaremos escribir cierta lógica fuera de nuestro cliente, ya sea por motivos de seguridad o para descargar la aplicación cliente de procesamiento. Las funciones de Cloud resuelven este problema permitiéndonos escribir un conjunto de funciones que se ejecutan en respuesta a diversos eventos, como cambios en la base de datos o inicios de sesión, que alberguen esa lógica [20].

En el proyecto se han utilizado para actualizar datos en Firestore cuando el usuario realiza ciertas acciones o para enviar notificaciones a otros usuarios como consecuencia de dichas acciones.

2.2.2.4 Cloud Storage

Este módulo permite almacenar y procesar archivos como imágenes o vídeos de los usuarios en un servidor [21]. En concreto, con este módulo ha sido posible almacenar las imágenes de perfil que los usuarios suben a la aplicación.

2.2.2.5 Extensions

Las extensiones de Firebase son un conjunto de soluciones para problemas comunes que han sido probadas y pretenden agilizar el desarrollo de las aplicaciones [22]. Mediante las extensiones podemos realizar tareas como acortar automáticamente URLs, enviar mensajes automáticos para activar los correos electrónicos de los usuarios o eliminar los datos de un usuario de forma recursiva en Cloud Firestore.

En este proyecto he hecho uso de la extensión para eliminar los datos de usuario cuando un usuario decide eliminar su cuenta.

2.2.2.6 Firebase Machine Learning

Este módulo permite al desarrollador introducir características de Machine Learning en sus aplicaciones. Las posibilidades que ofrece incluyen alojar e implementar modelos personalizados (por ejemplo, algunos que se hayan creado con TensorFlow), entrenar modelos o utilizar alguna de las APIs con modelos ya entrenados para resolver problemas comunes [23].

La aplicación que he desarrollado hace uso de una de esas APIs, en concreto la API para escaneo del códigos de barras, proporcionada por ML Kit [24].

Desde el teléfono, se escanea el código de barras de un libro para obtener su ISBN. Con este identificador seremos capaces de extraer la información acerca de los libros de un usuario. Podrá encontrar más detalles acerca de este proceso en el apartado relativo a Open Library.

Cabe mencionar que el uso de esta API es indirecto, ya que es un módulo npm para controlar la cámara del dispositivo (React Native Camera) el que lo usa. En el capítulo de Arquitectura y análisis se detalla en profundidad cómo se configura.

2.2.2.7 Cloud Messaging y Arquitectura FCM

Firebase Cloud Messaging (o FCM) es una solución de mensajería multiplataforma que permite enviar mensajes de notificación a las aplicaciones de los usuarios [25].

La arquitectura FCM precisa de dos componentes principales para enviar y recibir datos: un entorno de servidor que genere y oriente los mensajes y una aplicación cliente que los reciba mediante el servicio de transporte específico para la plataforma (iOS, Android o la web).

En nuestro caso, nuestro entorno servidor estará formado por un conjunto de funciones de Cloud y los clientes serán los dispositivos físicos que ejecuten la aplicación BookStand. El siguiente diagrama ilustra la arquitectura usada:

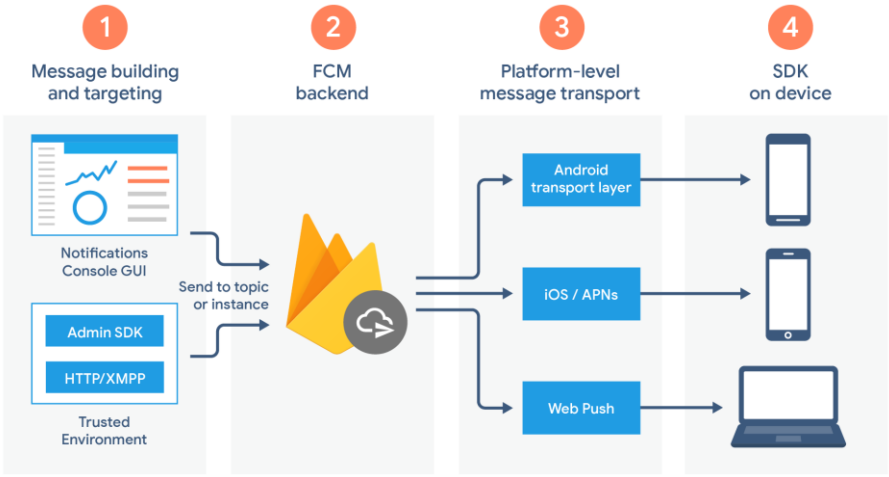

Ilustración 14 Arquitectura FCM

Como hemos mencionado las funciones de Cloud conformarán el entorno de confianza (1) que creará los mensajes y envía una solicitud de mensaje al *backend* de FCM (2). Este, tras recibir la solicitud del mensaje, es el encargado de enviarlo a la capa de transporte específica de la plataforma (3). Posteriormente, el mensaje se recibe en la aplicación del cliente (4) [26].

En el capítulo 4 de esta memoria se precisarán los detalles de la implementación de esta arquitectura.

2.3 Redux

Ilustración 15 Logo de Redux

2.3.1 Descripción y características generales

En apartados anteriores comentábamos que en React y en React Native, el estado de los componentes tiene un flujo descendiente desde el componente padre al componente hijo, lo que hace que sea complejo acceder a los datos desde varias partes de la aplicación.

Redux es un patrón software y una librería pensada para manejar y actualizar el estado de una aplicación a través de eventos llamados "acciones" [27]. Funciona como un almacén de estado centralizado para la aplicación que se asegura de que dicho estado solo se puede actualizar de forma predecible.

Bajo este patrón, tenemos el **estado** de nuestra aplicación, unas **vistas** que describen de forma declarativa la interfaz de usuario en base al estado y un conjunto de **acciones** o eventos que ocurren en nuestra aplicación y desencadenan actualizaciones en el estado. La Ilustración 16 representa de forma gráfica este patrón, enfatizando el flujo unidireccional de los datos:

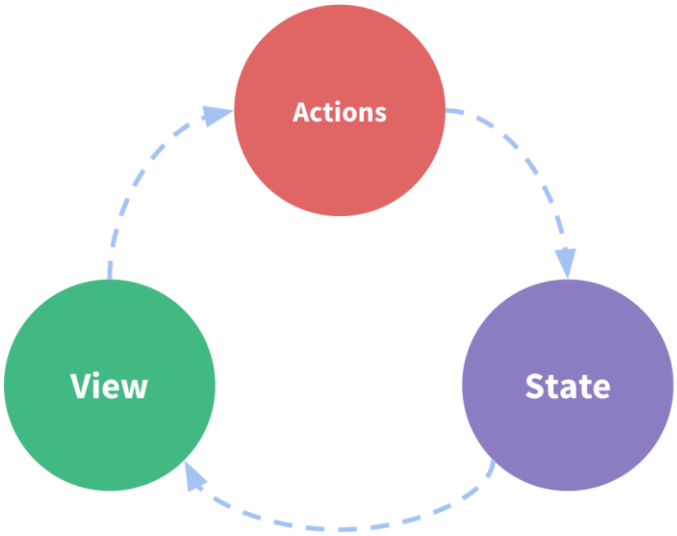

Ilustración 16 Flujo de datos Redux

Otra de las funcionalidades que permite Redux es hacer persistente el estado global de la aplicación, de forma que, tras cerrar la aplicación, apagar o reiniciar el dispositivo no se pierdan los datos.

2.3.2 Uso en la aplicación

Para usar este patrón en la aplicación tendremos que declarar varios elementos:

En primer lugar, definimos cuáles son los datos que almacenaremos en el *store* de Redux. En nuestro caso, la información que se incluye en el almacén serán los datos básicos del usuario de la aplicación y la información de los libros que están en la biblioteca. El lector podrá encontrar más información acerca de estos datos en el capítulo 4 de esta memoria.

Una vez que tenemos el estado definido, debemos crear reductores o *reducers* que son funciones que reciben el estado actual de la aplicación y una acción, deciden cómo actualizar el estado de ser necesario y lo devuelven. Podemos pensar en ellos como *listeners*[3] que manejan los eventos en función de las acciones recibidas. Son el único lugar donde se puede cambiar el estado de la aplicación.

En la Ilustración 17 Creación de *reducer* y acciones para los datos de usuario. Fichero `authSlice.ts`Ilustración 17 podemos apreciar el código necesario para crear un reducer con sus acciones.

[3] Un *listener* es un método que se ejecuta cuando ocurre algún evento que se le especifica previamente.

```
/* Creamos el reducer con la función createSlice que acepta el estado inicial,
un objeto con un conjunto de reducers y el nombre del slice y genera
automáticamente las acciones correspondientes a los reducers y al estado.

En nuestro caso, tenemos dos acciones: iniciar sesión y cerrar sesión. */
export const authSlice = createSlice({
    name: 'auth',
    initialState,
    reducers: {
        login: (state: Usuario, action: PayloadAction<Usuario>) => {
            // Cuando iniciamos sesión, asociamos el usuario
            state.uid = action.payload.uid;
            state.correoVerificado = action.payload.correoVerificado;
            state.correo = action.payload.correo;
            state.foto = action.payload.foto;
            state.nombre = action.payload.nombre;
            state.registroCorreo = action.payload.registroCorreo;
            state.token = action.payload.token;
        },
        logout: (state: Usuario) => {
            // Cuando cerramos sesión, devolvemos el estado al valor inicial
            state.uid = initialState.uid;
            state.correoVerificado = initialState.correoVerificado;
            state.correo = initialState.correo;
            state.foto = initialState.foto;
            state.nombre = initialState.nombre;
            state.registroCorreo = initialState.registroCorreo;
            state.token = initialState.token;
        },
    },
});

// Exportamos el reducer
export default authSlice.reducer;

// Exportamos las acciones
export const {login, logout} = authSlice.actions;
```

Ilustración 17 Creación de *reducer* y acciones para los datos de usuario. Fichero `authSlice.ts`

Para actualizar el estado se hace uso del método `dispatch` del almacén de estado, al que se le pasa como parámetros un objeto de acción. Esto desencadena el evento y provoca que se ejecute el código de los reducers, consiguiendo los cambios pertinentes en el estado.

Por último, tenemos los *selectores*: funciones que extraen información específica del almacén de estado de la aplicación. En la Ilustración 18 podemos ver un ejemplo de la creación de dichos selectores:

```
// Selector que nos permitirá acceder a todos los datos del usuario
export const selectUsuario = (state: RootState) => state.authReducer;

// Selector que nos permite acceder al identificador del usuario
export const selectUID = (state: RootState) => state.authReducer.uid;
```

Ilustración 18 Selectores para la información de usuario. Fichero `authSlice.ts`

Una vez que hemos hecho esto, podemos crear el almacén y usarlo en nuestra aplicación. Esto podemos verlo

en la Ilustración 19 que se muestra a continuación:

```
/* Combinamos los reductores que usará nuestra aplicación en uno solo
Uno contiene los datos del usuario (authReducer) y otro los datos de los libros
(librosReducer) */
const appReducers = combineReducers({ authReducer, librosReducer })

// Definimos un objeto con la configuración de persistencia deseada
const persistConfig = {
  key: 'root',
  storage: AsyncStorage,
  whitelist: ['authReducer', 'librosReducer'],
}

// Aplicamos la configuración a los reducers de la aplicación
const persistedReducer = persistReducer(persistConfig, appReducers)

// Creamos el store de la apliación con los reducers y el middleware necesario
export const store = createStore(
  persistedReducer,
  applyMiddleware(createLogger(), Redux)
)

// Definición del tipo de nuestro almacén global
export type RootState = ReturnType<typeof store.getState>

// Exportamos el store persistido
export const persistedStore = persistStore(store);
```

Ilustración 19 Creación del almacén de estado global. Fichero `store.ts`

2.4 Open Library

Ilustración 20 Logo de Open Library

2.4.1 Descripción y características generales

Open Library [28], es una iniciativa de Internet Archive [29], una organización sin fines de lucro, que construye una biblioteca digital de sitios de Internet y otros artefactos culturales en formato digital.

Ofrece un conjunto de APIs para obtener información acerca de diferentes obras de forma gratuita. Las usadas en el proyecto son las siguientes:

- Open Library Books API [30]: proporciona un método programático para la consulta de información acerca de libros usando JavaScript. De las múltiples opciones que tenemos para encontrar la información acerca de un libro, nos interesa la que usa los ISBN de los libros, ya que es la información que obtenemos al escanear el código de barras.
- Open Library Covers API [31]: proporciona un método programático para acceder a las imágenes de portada de los libros. En este caso también accedemos a la información a través del ISBN del libro.

Esta API tiene límites de 100 peticiones por IP cada 5 minutos, pero se han considerado que los límites son suficientes para nuestro uso en la aplicación.

- Open Library Search API [32], para realizar búsquedas de libros por título. En nuestra aplicación la usamos como un método opcional, en el caso en el que no se pueda encontrar una edición de ISBN en concreto o para casos en los que el usuario no pueda hacer uso de su cámara para escanear el código del ISBN.

2.4.2 Uso en la aplicación

En este proyecto usamos estas APIs para obtener la información relativa a los libros de la biblioteca del usuario: título, autores, géneros y cubierta del libro.

Esta información se obtiene mediante la Books API, a partir del ISBN [33], un identificador único de diez o trece dígitos asociado a un código de barras que podemos encontrar en la amplia mayoría de los libros publicados.

Para obtener los datos, construimos la URL con el formato apropiado e incluyendo el ISBN del libro del que queremos obtener la información. Como resultado, obtenemos una respuesta en formato JSON de la que extraemos los datos que necesitamos. En la Ilustración 21 se muestra el código para realizar esta tarea.

```
// Buscamos el libro en la API de OpenLibrary
fetch(`https://openlibrary.org/api/books?bibkeys=ISBN:${isbn}&format=json&jscmd=data`)
  .then(async (respuestaAPI) => {
    // Procesamos la respuesta de la API
    switch (respuestaAPI.status) {
      case 200:
        // Obtenemos el objeto JSON
        libroJSON = await respuestaAPI.json();

        /* El formato del objeto JSON incluye un campo de nombre "ISBN:" seguido
        del ISBN del libro concreto. Si este campo no existe, el libro no se ha
        encontrado */

        if (libroJSON[`ISBN:${isbn}`]) {
          // Si el libro se ha encontrado

          /* Añadimos los datos que nos interesan a nuestro libro: el isbn, el
          título, los autores y los géneros. */
          libro = {
            isbn: isbn,
            titulo: libroJSON[`ISBN:${isbn}`]?.title
              ? libroJSON[`ISBN:${isbn}`]?.title
              : undefined,
            autores: libroJSON[`ISBN:${isbn}`]?.authors
              ? json2array(libroJSON[`ISBN:${isbn}`]?.authors)
              : undefined,
            generos: libroJSON[`ISBN:${isbn}`]?.subjects
              ? json2array(libroJSON[`ISBN:${isbn}`]?.subjects)
              : undefined,
            esPropietario: true,
          };

          // Devolvemos el libro
          return libro;
        } else { /* El libro no se ha encontrado ... */ }
        break
      default:
        /* Se ha producido un error ... */
    }})
  .catch((error) => {/* Se ha producido un error ... */ })
```

Ilustración 21 Uso de Open Library Books API. Fichero `openLibrary.ts`

Para obtener las cubiertas de los libros se utiliza la ya mencionada Covers API construyendo la dirección URL apropiada con el ISBN del libro, tal y como se muestra en la Ilustración 22. Al acceder a dicha dirección obtendremos un archivo JPG con la imagen deseada o un error en el caso de no encontrarse la foto.

```
`http://covers.openlibrary.org/b/isbn/${props.estado.libro.isbn}-M.jpg?default=false`
```

Ilustración 22 URL necesaria para obtener la imagen de portada de un libro

Por último, para buscar libros a partir del título y obtener su información utilizaremos la Search API. Esta API nos devuelve un JSON con información acerca de todos los libros cuyo título contiene las palabras que introduzcamos en el campo de búsqueda. La URL a la que tenemos que acceder es la siguiente:

```
`http://openlibrary.org/search.json?title=${titulo}`
```

Ilustración 23 URL necesaria para obtener libros a partir del título

3 HERRAMIENTAS Y RECURSOS UTILIZADOS

Si tu única herramienta es un martillo, tiendes a tratar cada problema como si fuera un clavo.

Abraham Maslow

3.1 Herramientas para el desarrollo

En este apartado describiré las herramientas principales que se han utilizado durante la realización del proyecto en la fase de desarrollo y depuración de la aplicación.

3.1.1 Visual Studio Code

Ilustración 24 Logo de Visual Studio Code

Visual Studio Code [34] es un editor de código ofrecido por Microsoft disponible para Linux, Windows y macOS para numerosos lenguajes de programación, entre los que se encuentra TypeScript. Cuenta con diversas características para asistir al desarrollador como completar código, un entorno de depuración, herramientas para refactorización y soporte para el control de versiones con Git.

Por todas estas características, lo he elegido editor de código para desarrollar la aplicación en React Native. En la Ilustración 25 podemos observar la interfaz de este programa.

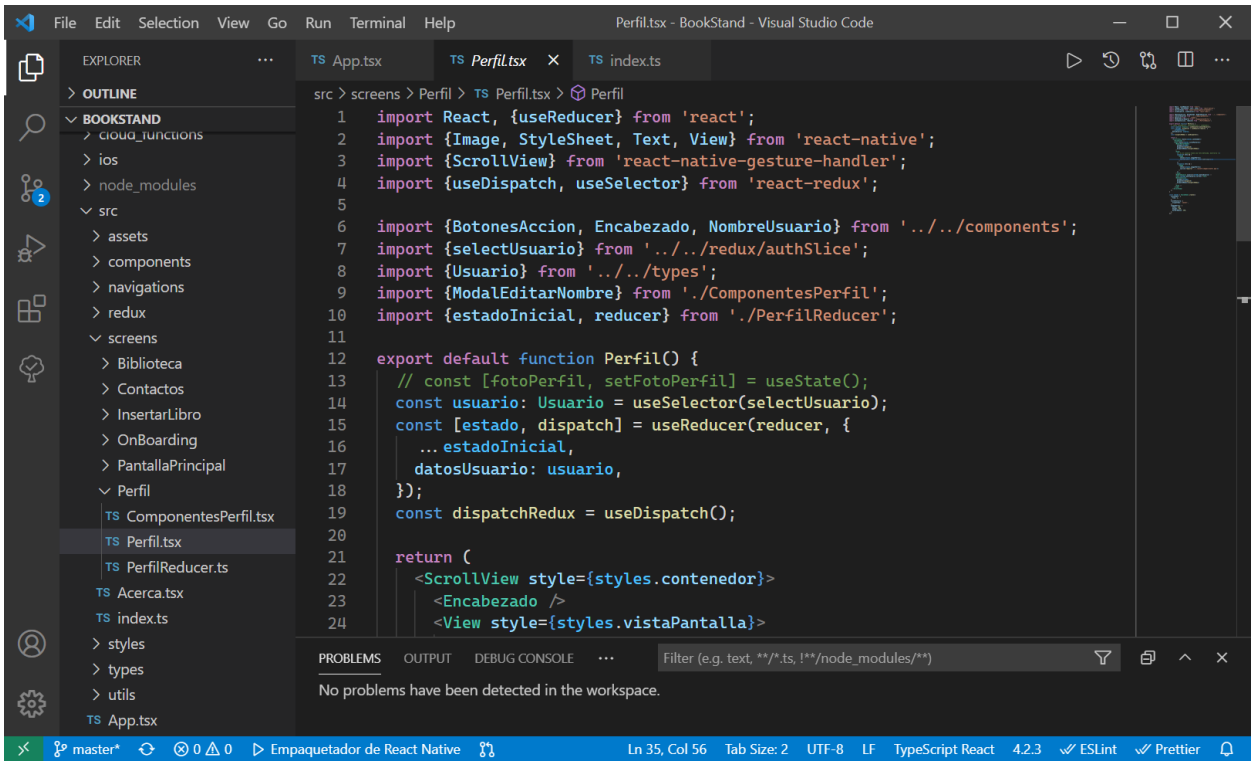

Ilustración 25 Interfaz de Visual Studio Code

3.1.2 Git y GitHub

Ilustración 26 Logo de GitHub

Para el desarrollo de cualquier aplicación o programa hoy en día, es imprescindible el uso de algún software para el control de versiones. En este proyecto se ha hecho uso tanto de Git como de GitHub para este propósito, ambos a través de sus correspondientes extensiones en Visual Studio Code.

Además, todo el código de la aplicación se encuentra disponible en un repositorio de GitHub público para que quien lo desee pueda consultarlo: repositorio de la App BookStand y código de Firebase Functions [35]

3.1.3 Consola de Firebase

Para interactuar con la parte del servidor se ha usado la consola que Firebase proporciona a través de una página web. Gracias a esta interfaz podemos visualizar el estado y la información que proporcionan los diferentes módulos de Firebase, así como gestionarlos. En las ilustraciones que aparecen a continuación se muestra la interfaz de la consola de Firebase:

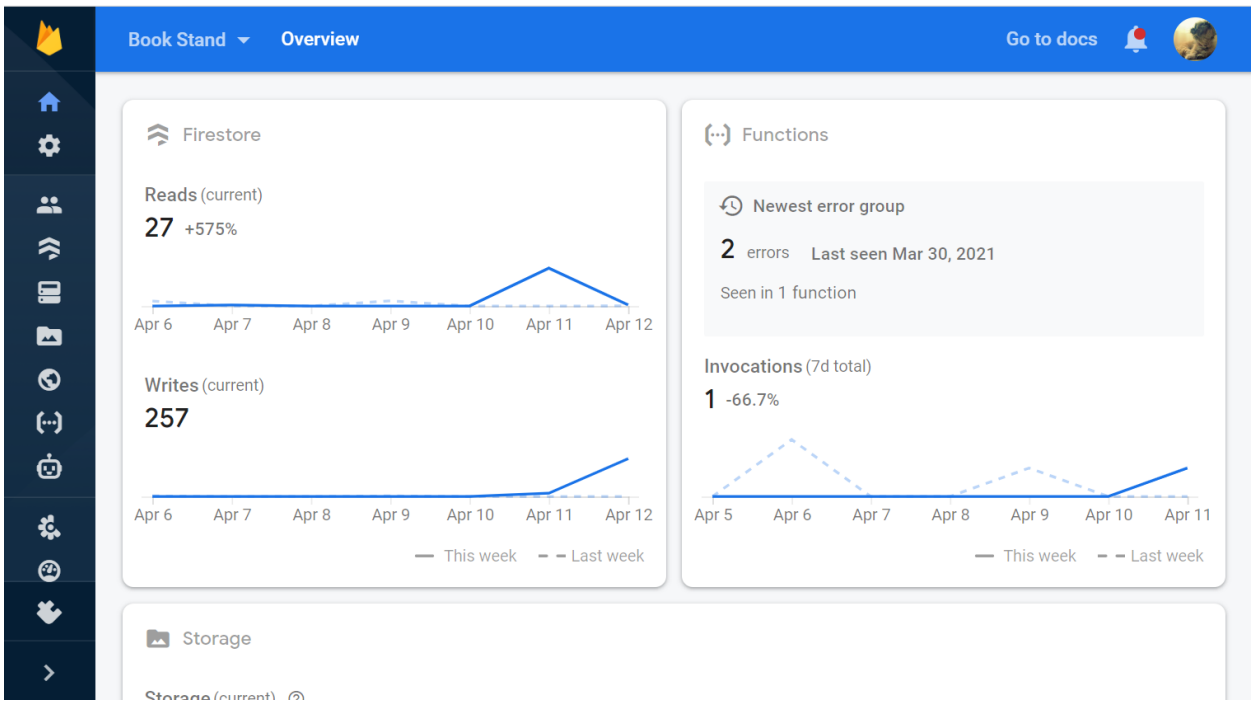

Ilustración 27 Página de inicio de la consola de Firebase

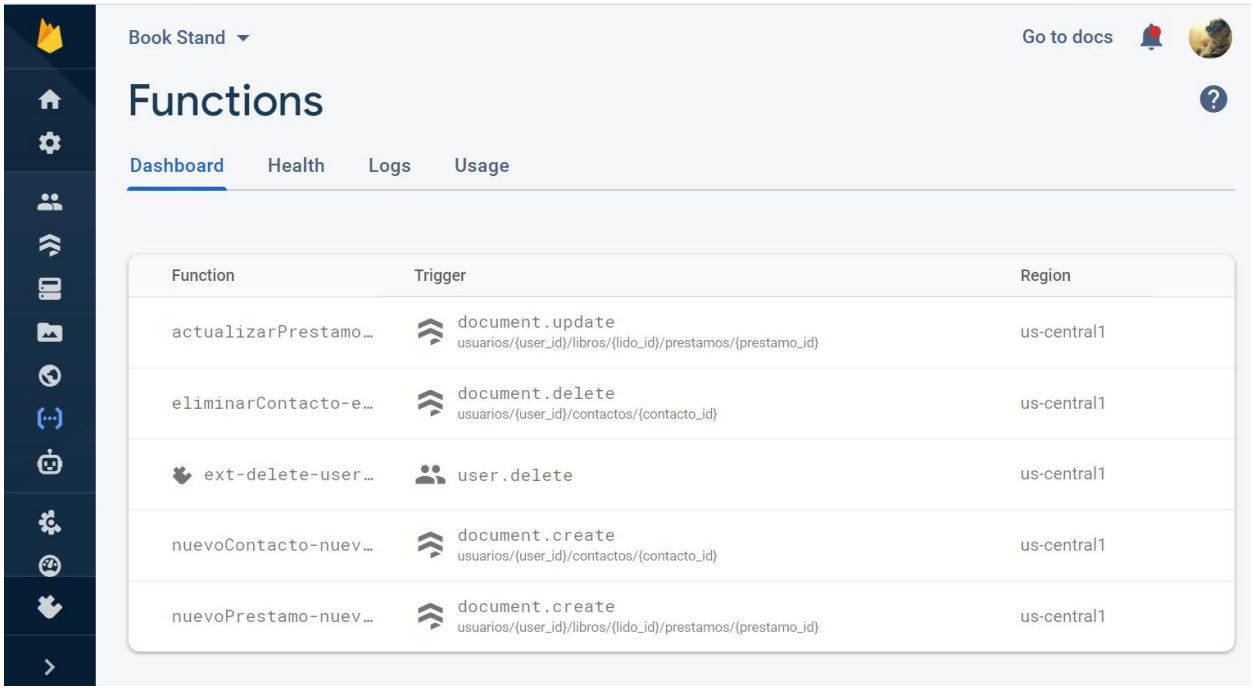

Ilustración 28 Página del módulo de Cloud Functions

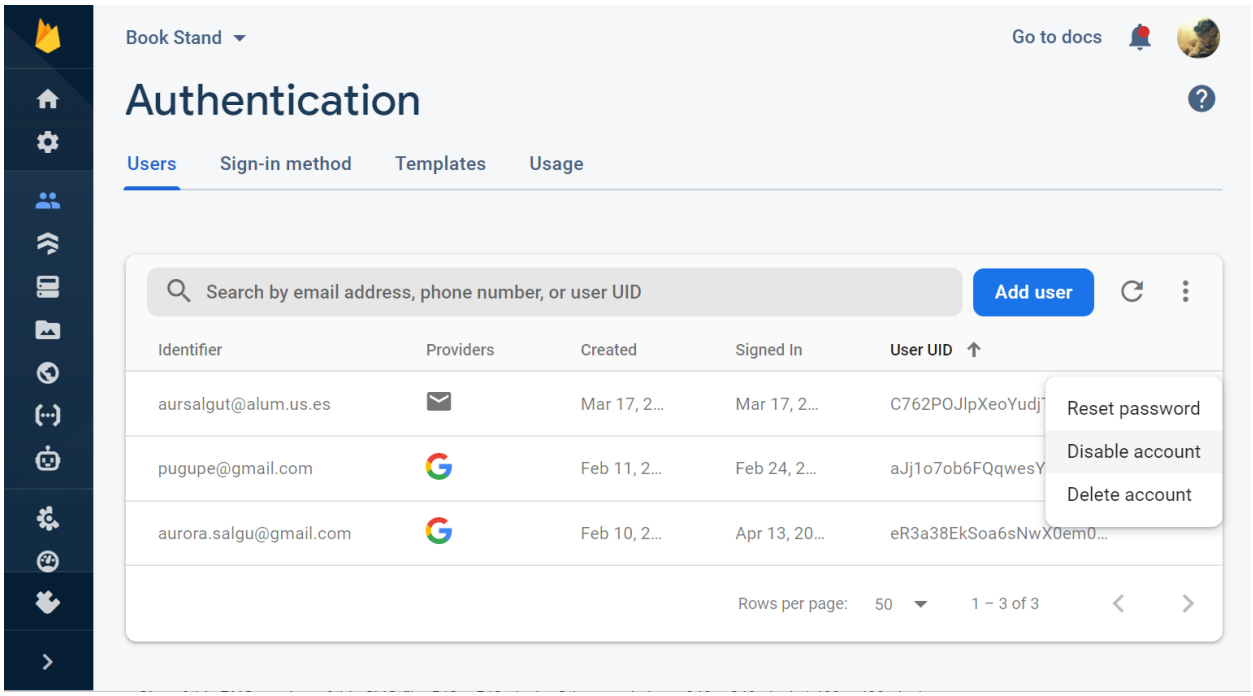

Ilustración 29 Página del módulo Authentication

3.1.4 Librerías y programas necesarios para desarrollar en React Native

Para poder desarrollar aplicaciones con React Native se requiere un conjunto de librerías y software específico que dependerá del sistema operativo para el que desarrollemos nuestra aplicación. En la tabla que aparece a continuación se muestran dichas herramientas:

Tabla 1 Herramientas para desarrollar aplicaciones en React Native

Herramientas comunes	Específico para Android	Específico para iOS
• Node • Interfaz de línea de comandos de React Native	• Java SE Development Kit (JDK) • Android Studio	• Xcode • CocoaPods • Watchman

3.1.4.1 Node (NodeJS)

Node [36] es un RTE (Runtime Enviroment, o sistema en tiempo de ejecución) para JavaScript sobre el que se apoya React Native para ejecutar sus aplicaciones.

3.1.4.2 Interfaz de línea de comandos de React Native

Para instalar React Native, utilizaremos NPM [11], un gestor de paquetes para Node que nos permite instalar módulos y librerías en nuestros proyectos a través del terminal.

En general, el comando que usamos para instalar dichos módulos es:

```
npm install <nombre_modulo>
```

Ilustración 30 Comando npm para la instalación de módulos

Y en particular, para instalar la librería de React Native, usamos:

```
npm install react-native
```

Ilustración 31 Comando para instalar React Native

En los siguientes apartados describimos las herramientas específicas que usamos para desarrollar la aplicación en cada una de las plataformas.

3.1.5 Entorno de Desarrollo para Android

Para desarrollar aplicaciones para Android con React Native se puede usar cualquier sistema operativo: macOS, Linux o Windows. En este caso, se ha utilizado un ordenador con Windows 10.

Para la instalación de las dependencias de React Native, crear y lanzar aplicaciones e instalar módulos a través de NPM es necesario utilizar un terminal. En este caso se ha usado Windows PowerShell a través de Windows Terminal.

Una vez instalado Node y el JDK (en la Ilustración 32 se muestra el comando necesario)Ilustración 1, se instala React Native tal y como se comentó más atrás.

```
choco install -y nodejs.install openjdk8
```

Ilustración 32 Comandos para instalar las dependencias

3.1.5.1 Android Studio

Android Studio [37] es el entorno de desarrollo integrado oficial para desarrollar aplicaciones en Android. Pese a que en este proyecto se ha decidido usar Visual Studio Code para el desarrollo de la aplicación, es necesario instalar este software para poder hacer uso de los simuladores de Android y de las herramientas del SDK (Software Development Kit).

En la Ilustración 33 podemos observar el emulador de Android ejecutando la aplicación BookStand:

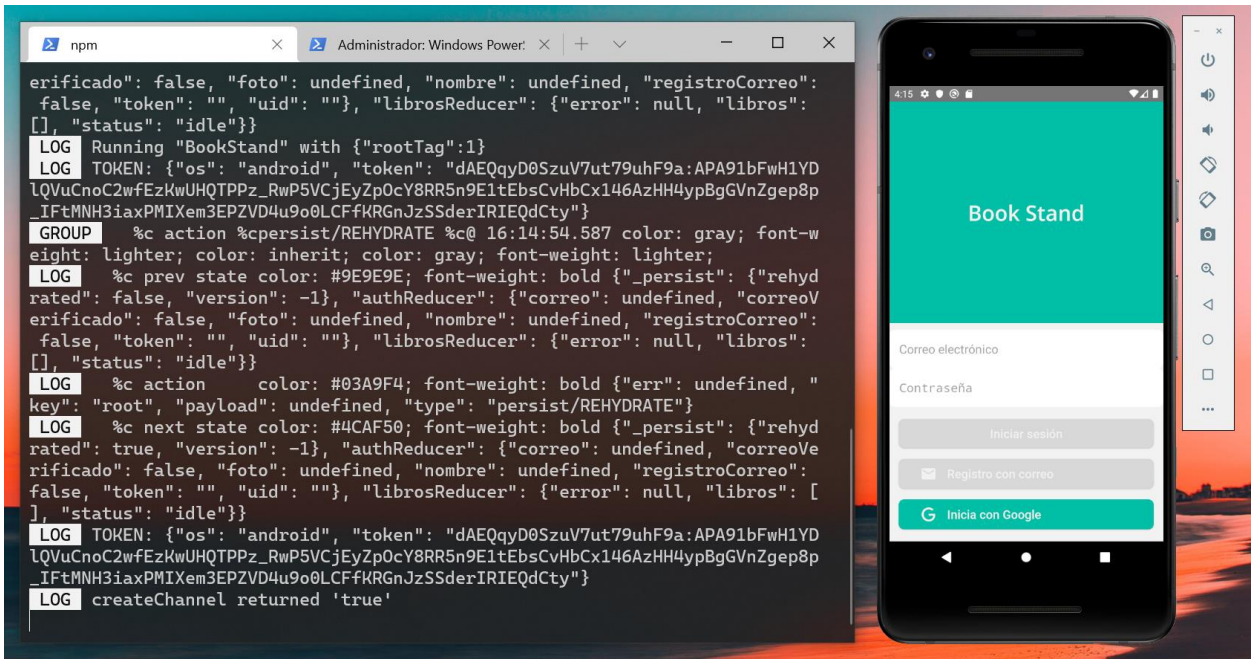

Ilustración 33 Emulador de Android

3.1.5.2 Dispositivos Android físicos

A la hora de realizar pruebas en una aplicación es necesario hacerlo no solo en simuladores, sino en dispositivos físicos, por poseer estos últimos características especiales como cámaras o porque algunas funcionalidades como los mensajes de Cloud de Firebase no se pueden probar en emuladores.

Además, otra ventaja que tiene el uso de los dispositivos físicos a la hora de desarrollar y depurar código es que permiten que el ordenador con el que se trabaje no use tantos recursos, lo que permite una experiencia a la hora de programar más rápida y fluida.

Para utilizar un equipo físico para el desarrollo, simplemente debemos activar la opción de depuración USB en el terminal y conectarlo al ordenador mediante un cable USB.

3.1.6 Entorno de desarrollo para iOS

Como mencionamos antes, React Native permite desarrollar aplicaciones multiplataforma, por lo que resulta interesante probar las capacidades de este *framework* en iOS además de en Android. Desafortunadamente, para desarrollar y probar aplicaciones compatibles con el sistema operativo de Apple, es necesario disponer de una máquina con macOS y usar herramientas específicas para la plataforma, como CocoaPods[4] y Xcode.

Para realizar esta tarea, y al no disponer de un ordenador de la marca Apple, he recurrido al uso de una máquina virtual que, pese a que proporciona una experiencia de desarrollo inferior, resulta suficiente para poder hacer pruebas sencillas con la aplicación.

El programa de virtualización que se ha utilizado es VMware Workstation 15 Player [38]. En él se ha creado una máquina virtual actualizada a la versión de macOS Catalina 10.15.7, tal y como se muestra en la Ilustración 34. Posteriormente se procede a la instalación del software necesario para el desarrollo: Xcode, Node, React Native, y las librerías Watchman[5] y CocoaPods.

[4] CocoaPods es un gestor de dependencias para los proyectos de Xcode.
[5] Watchman es una herramienta proporcionada por Facebook para mejorar el rendimiento a la hora de desarrollar en React Native.

Ilustración 34 Versión de macOS Catalina instalada

Exceptuando Xcode, que se descarga desde la App Store, podemos instalar estos programas desde el terminal usando los siguientes comandos:

```
brew install node
brew install watchman
brew install cocoapods
```

Ilustración 35 Instalación de las dependencias de React Native en macOS

Una vez instalado Node, para incluir la librería de React Native en nuestro sistema usamos el mismo comando npm que se incluyó en el apartado 3.1.4.2.

En la Ilustración 36 podemos observar el emulador de iOS proporcionado por Xcode ejecutando una aplicación de prueba en React Native.

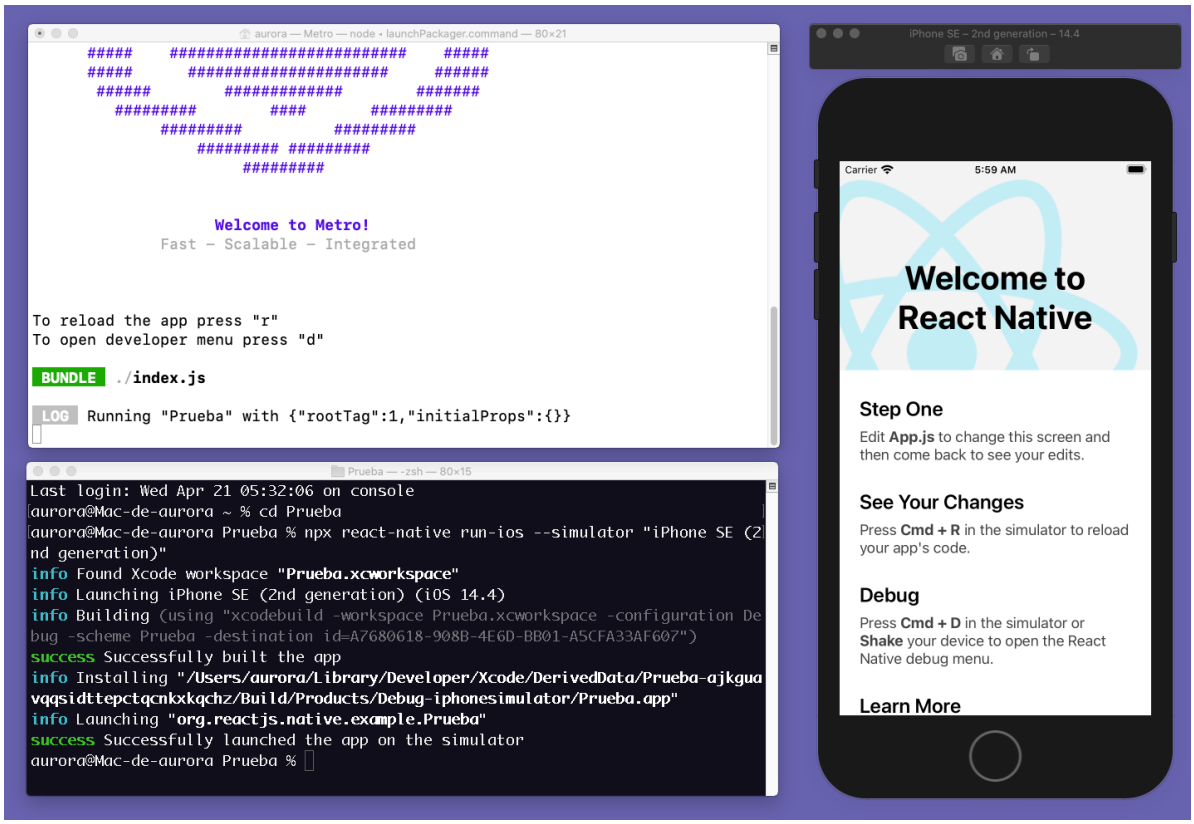

Ilustración 36 Ejecución de un proyecto de prueba en React Native en el entorno macOS

3.1.6.1 Xcode

Ilustración 37 Versión de Xcode instalada en la máquina virtual

Xcode [39] es el IDE ofrecido por Apple para el desarrollo de aplicaciones para sus dispositivos. Además de ser un editor de código, ofrece las herramientas de compilación y los simuladores necesarios para poder emular y probar dichas aplicaciones. React Native hace uso de estas herramientas para construir el ejecutable de la app, de forma que solo usaremos este programa de forma indirecta.

Para instalar este programa en nuestra máquina es necesario disponer de un ID de Apple [40], que lo podemos crear de forma gratuita desde su página web.

3.2 Herramientas para el prototipado, modelado y diseño

En este apartado aparecen un conjunto de recursos auxiliares que se han utilizado para realizar el diseño, modelado y prototipado de la aplicación.

3.2.1 Figma

Figma es una herramienta web para la edición y diseño de interfaces gráficas de usuario. En este proyecto ha sido utilizada para crear un prototipo inicial de la aplicación que sirva de modelo de referencia.

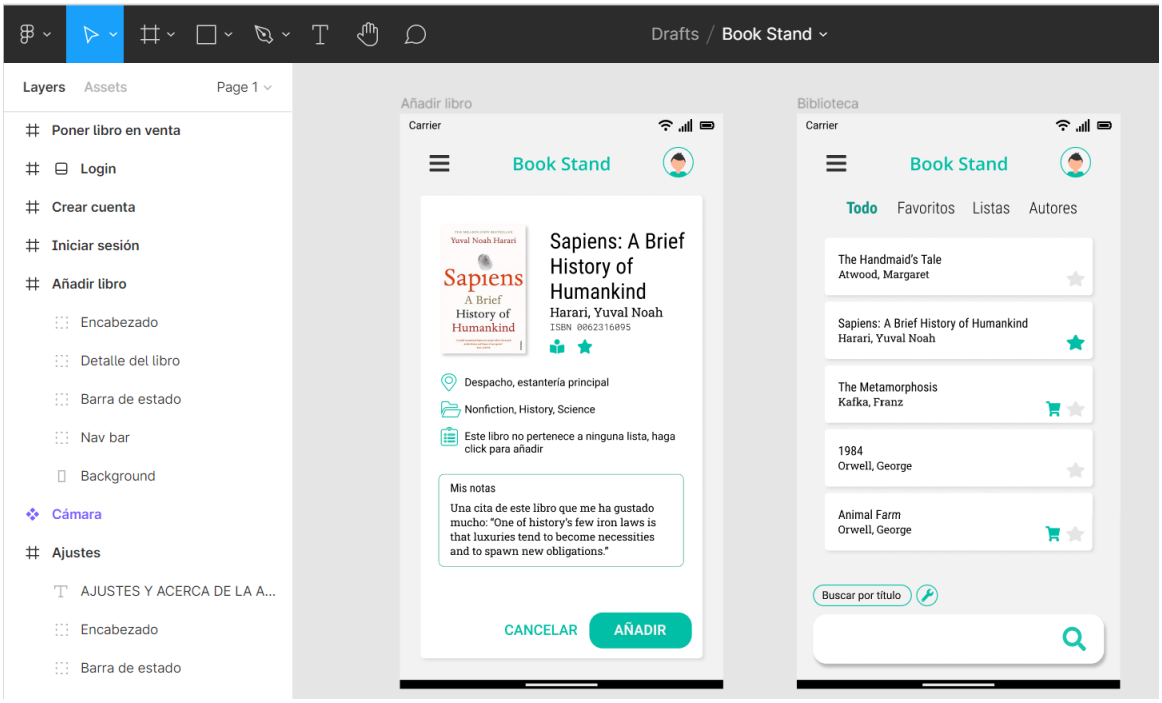

Ilustración 38 Pantallas para añadir un libro y ver la biblioteca en el prototipo

3.2.2 unDraw

unDraw [41] es una galería de ilustraciones de código abierto que permite el uso de imágenes en proyectos tanto comerciales como personales. En este proyecto se usan algunas de dichas ilustraciones en el diseño de la aplicación.

3.2.3 Diagrams.net

Ilustración 39 Logo de Diagrams.net

Diagrams.net [42] es una aplicación web gratuita que permite al usuario crear diseños y diagramas de cualquier tipo (diagramas UML[6], diagramas de flujo, diagramas de bases de datos, etc.) de forma rápida. Se ha usado a lo largo de esta memoria para crear algunos diagramas sencillos.

3.2.4 Visual Paradigm

Ilustración 40 Logo de Visual Paradigm

[6] Unified Modeling Language

Visual Paradigm [43] ofrece un conjunto de herramientas para diseñar software y gestionar proyectos. Una de sus principales herramientas son las relacionadas con el modelado de software que soportan, entre otros, diagramas UML y diagramas para bases de datos.

En este proyecto se ha usado esta herramienta para crear diagramas UML (casos de uso, diagramas de secuencia o diagramas de clase) y diagramas para ilustrar la estructura de la base de datos del proyecto.

4 ARQUITECTURA Y ANÁLISIS

Programar sin una arquitectura o diseño general en mente es como explorar una cueva con solo una linterna: no sabes dónde has estado, no sabes a dónde vas y no sabes muy bien dónde estás.

Danny Thorpe

En este capítulo nos centraremos en detallar la estructura y el funcionamiento de la aplicación a través de un conjunto de diagramas UML.

4.1 Diagrama de componentes

Comenzaremos por un diagrama de componentes que representa la arquitectura global de la aplicación (Ilustración 41):

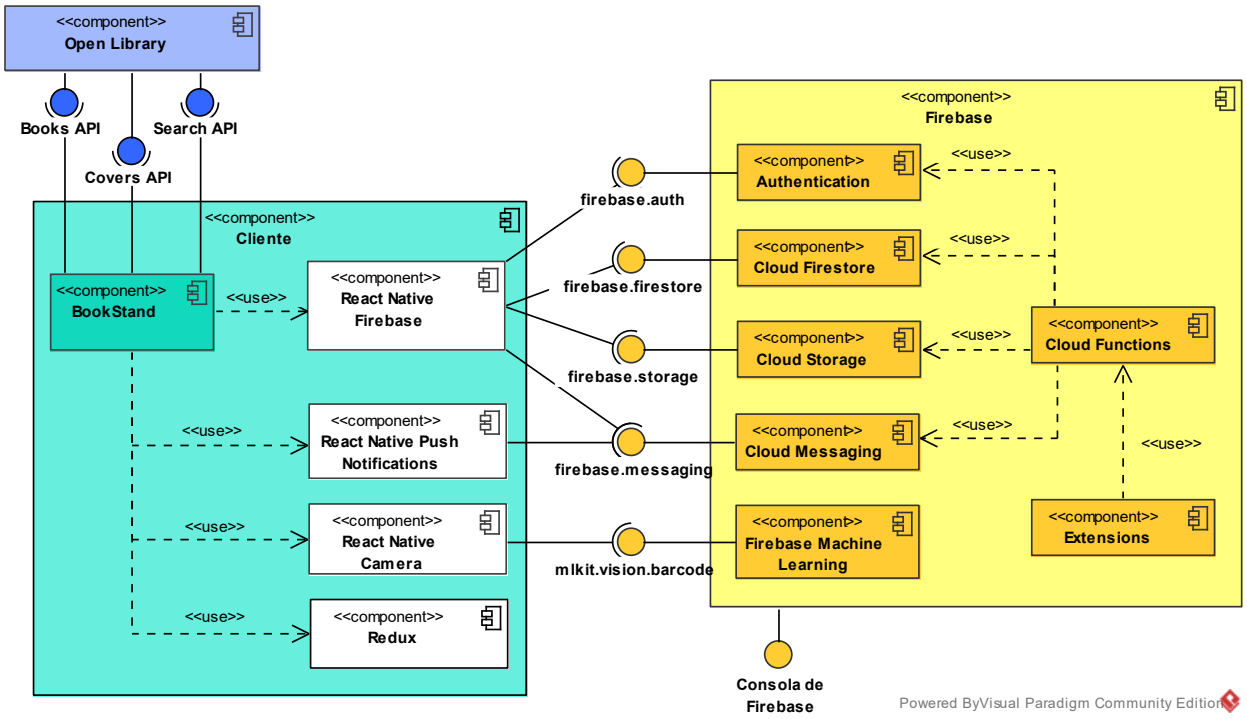

Ilustración 41 Diagrama de componentes de la arquitectura de la aplicación

En esta figura se ponen de manifiesto cada una de las partes que forman el proyecto, así como su relación con el resto de los elementos. Se aprecian las interfaces proporcionadas por cada uno de los módulos de Firebase, que serán utilizadas por la aplicación cliente mediante los paquetes pertinentes. Además, queda reflejada la relación entre los diferentes módulos de Firebase con las funciones de Cloud: en estas funciones se puede acceder y modificar la base de datos, enviar notificaciones o gestionar usuarios. También se recoge el hecho de

que las extensiones de Firebase no son más que un conjunto de funciones de Cloud predefinidas, que hacen uso del resto de módulos.

Por otra parte, el diagrama representa las APIs de Open Library a las que accede la app para obtener los datos de los libros.

Por último, se destaca la interfaz llamada “Consola de Firebase”, ya que se trata de la interfaz gráfica que nos permite administrar el servidor a través del navegador, tal y como se comentó en el apartado 3.1.3.

4.2 Casos de uso generales

En este apartado se presentan un conjunto de diagramas de casos de uso. Este tipo de diagramas sirven para describir las interacciones ente el sistema y uno o más actores, considerando el sistema como una caja negra [44], por lo que son idóneos para ilustrar la funcionalidad de la aplicación en desarrollo.

4.2.1 Autenticación de un usuario

A continuación, se detallan los procesos para registrar y autenticar a un usuario en la aplicación. Para ello, entran en juego dos actores: el usuario que pretende autentificarse y el módulo de Firebase Authentication, que se encarga de verificar la identidad del usuario en el servidor.

En la Ilustración 42 y en las tablas que aparecen a continuación se explican detalladamente estas operaciones:

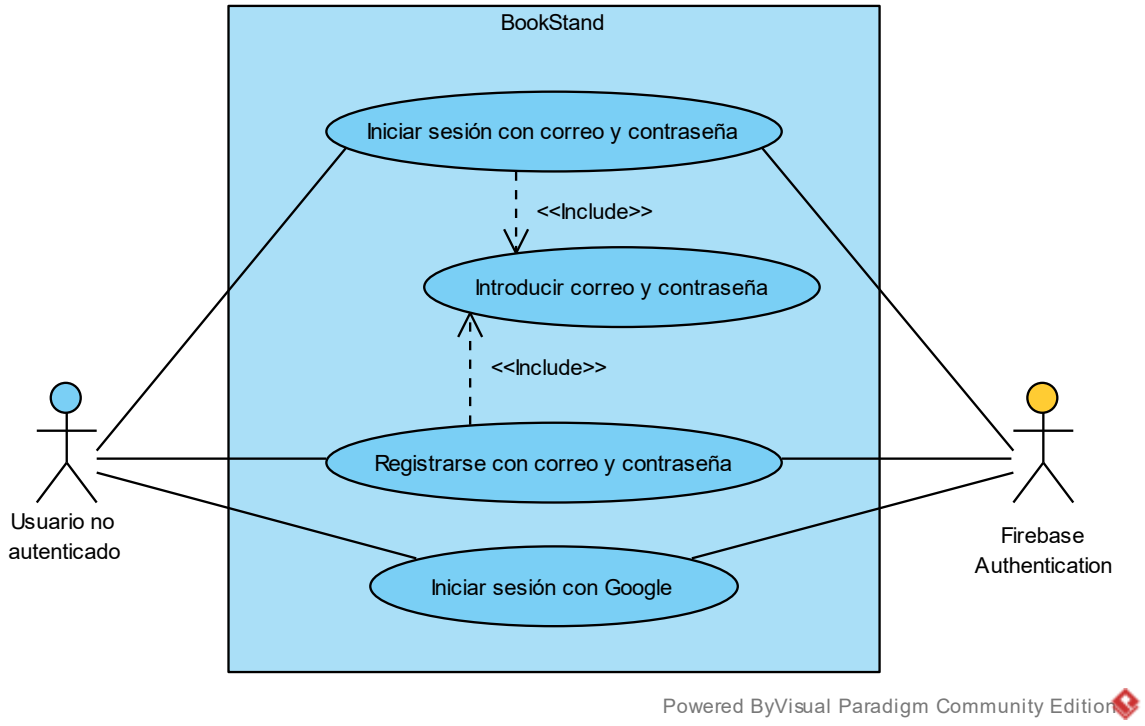

Ilustración 42 Casos de uso para autenticar usuarios

CU-01	**Iniciar sesión con correo y contraseña**	
Precondición	El usuario debe estar previamente registrado en la aplicación.	
Descripción	En este caso de uso se detalla el comportamiento de la aplicación para autenticar a un usuario que tiene una cuenta creada en la aplicación.	
Secuencia nominal	Paso	Acción

	1	El usuario introduce su correo electrónico y contraseña	
	2	El usuario solicita al sistema iniciar sesión	
	3	El módulo de autenticación de Firebase verifica correctamente las credenciales introducidas	
Postcondición	El usuario inicia sesión correctamente en la aplicación.		
Excepciones	Paso	Acción	
	1	El usuario introduce el correo o la contraseña de forma incorrecta o incompleta	
		E.1	La aplicación informa al usuario del error y le insta a modificar los datos
		E.2	Se cancela el caso de uso
	3	El usuario no se encuentra registrado en el servidor de autenticación	
		E.1	La aplicación informa al usuario del error y le insta a registrarse primero
		E.2	Se cancela el caso de uso
Comentarios	Ninguno.		

Tabla 2 CU-01 Iniciar sesión con correo y contraseña

CU-02	**Registrarse con correo y contraseña**		
Precondición	El usuario no debe estar previamente registrado en la aplicación.		
Descripción	En este caso de uso se detalla el comportamiento de la aplicación para registrar a un usuario en la aplicación.		
Secuencia nominal	Paso	Acción	
	1	El usuario introduce su correo electrónico y contraseña	
	2	El usuario solicita al sistema iniciar sesión	
	3	El módulo de autenticación de Firebase crea una cuenta para el usuario a partir de las credenciales introducidas	
Postcondición	El usuario queda registrado correctamente e inicia sesión en la aplicación.		
Excepciones	Paso	Acción	
	1	El usuario introduce un correo o contraseña inválido o incompleto.	
		E.1	La aplicación informa al usuario del error y le insta a modificar los datos

		E.2	Se cancela el caso de uso
	3	El usuario ya se encuentra registrado en el servidor de autenticación	
		E.1	La aplicación informa al usuario del error y le insta a iniciar sesión en su lugar
		E.2	Se cancela el caso de uso
Comentarios	Ninguno.		

Tabla 3 CU-02 Registrarse con correo y contraseña

CU-03	**Iniciar sesión con Google**		
Precondición	El usuario debe disponer de una cuenta de Google con la que poder autenticarse.		
Descripción	En este caso de uso se detalla el comportamiento de la aplicación para autenticar a un usuario en la aplicación a través de su cuenta de Google.		
Secuencia nominal	Paso	Acción	
	1	El usuario solicita al sistema continuar con una cuenta de Google	
	2	Se muestra al usuario una lista con las cuentas de Google que tenga registradas en el dispositivo para que seleccione o añada una.	
	3	El módulo de autenticación de Firebase verifica correctamente las credenciales introducidas	
Postcondición	El usuario inicia sesión correctamente en la aplicación.		
Excepciones	Paso	Acción	
	3	Se ha producido un error al autenticar al usuario frente al servidor	
		E.1	La aplicación informa al usuario del error
		E.2	Se cancela el caso de uso
Comentarios	Ninguno.		

Tabla 4 CU-03 Iniciar sesión con Google

4.2.2 Operaciones en la biblioteca

En el diagrama de casos de uso que aparece en la Ilustración 43Ilustración 1 se muestran los casos de uso relacionados con la gestión de la biblioteca personal del usuario:

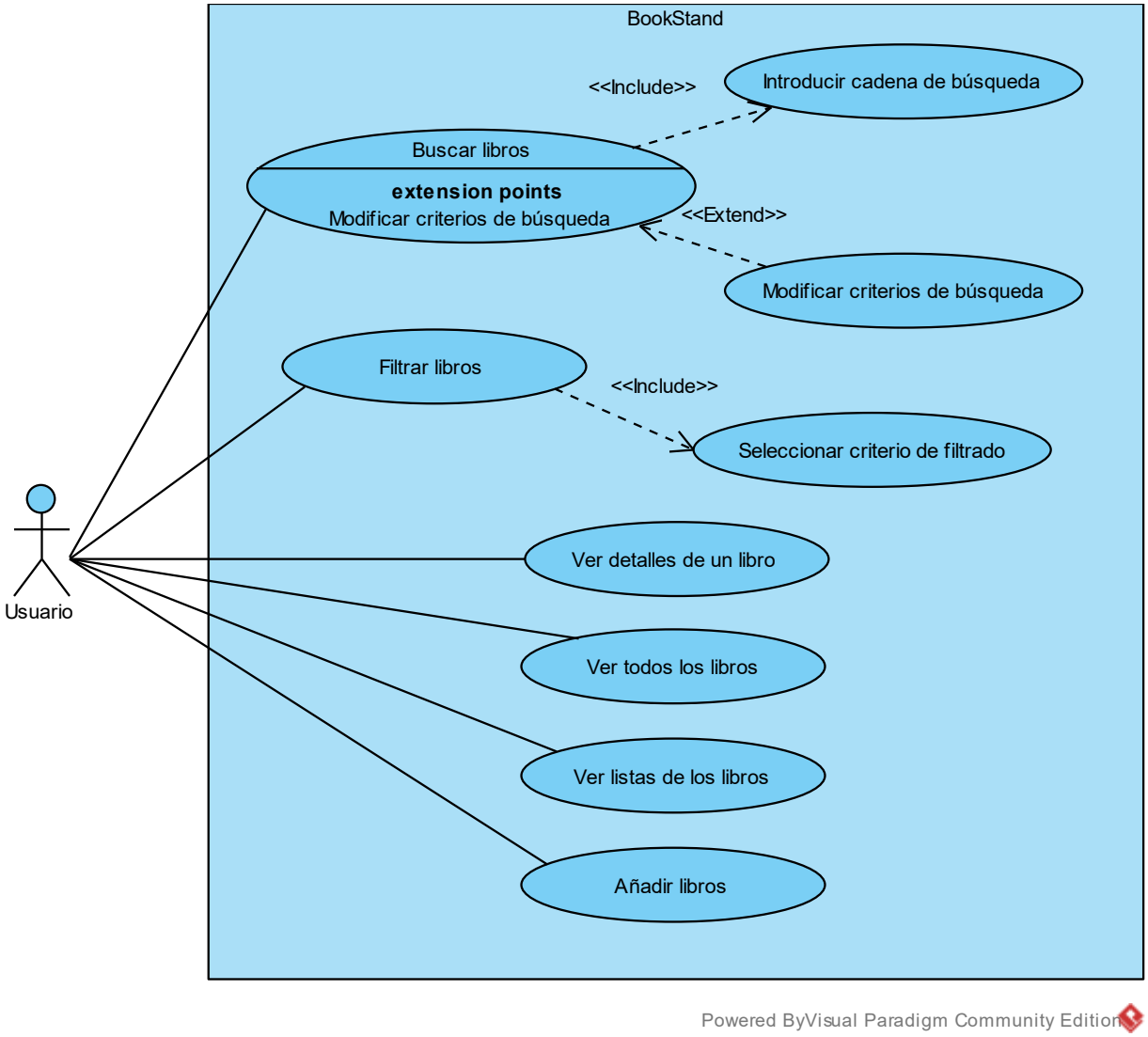

Ilustración 43 Diagrama de casos de uso para la gestión de la biblioteca

En las tablas que aparecen a continuación se recogen los detalles de los casos de uso más importantes que aparecen en el diagrama:

CU-04	**Buscar libro**	
Precondición	El usuario debe tener algún libro en su colección.	
Descripción	En este caso de uso se detalla el comportamiento de la aplicación para realizar búsquedas de libros dentro de su colección por diferentes criterios (título, ISBN, autores o géneros).	
Secuencia nominal	Paso	Acción
	0	Por defecto el sistema busca los libros por título, pero el usuario puede modificar el criterio con el que desea realizar la búsqueda (Se realiza el caso de uso *Modificar criterios de búsqueda*)
	1	El usuario introduce la cadena de búsqueda
Postcondición	El usuario obtiene una lista con todos los libros que cumplen dicha condición.	

Excepciones	Ninguna.
Comentarios	Si no se encuentran libros que cumplan la condición, no se produce error, simplemente se muestra la lista vacía.

Tabla 5 CU-04 Buscar libro

CU-05	**Filtrar libros**	
Precondición	El usuario debe tener algún libro en su colección.	
Descripción	En este caso de uso se detalla el comportamiento de la aplicación para mostrar los libros que cumplen una condición binaria: libros que el usuario ha leído, libros favoritos o libros que tienen un préstamo en curso.	
Secuencia nominal	Paso	Acción
	1	El usuario selecciona un criterio de filtrado
Postcondición	El usuario obtiene una lista con todos los libros que cumplen dicha condición.	
Excepciones	Ninguna.	
Comentarios	Si no se encuentran libros que cumplan la condición, no se produce error, simplemente se muestra la lista vacía.	

Tabla 6 CU-05 Filtrar libros

Debido a que los casos de uso para *Añadir un libro* y para *Ver detalles de un libro* son más complejos, aparecen detallados en los apartados 4.2.3 y 4.2.4 respectivamente.

4.2.3 Operaciones para añadir un libro

En este apartado se describen los casos de uso para añadir un libro al sistema en la Ilustración 44 y la Tabla 7:

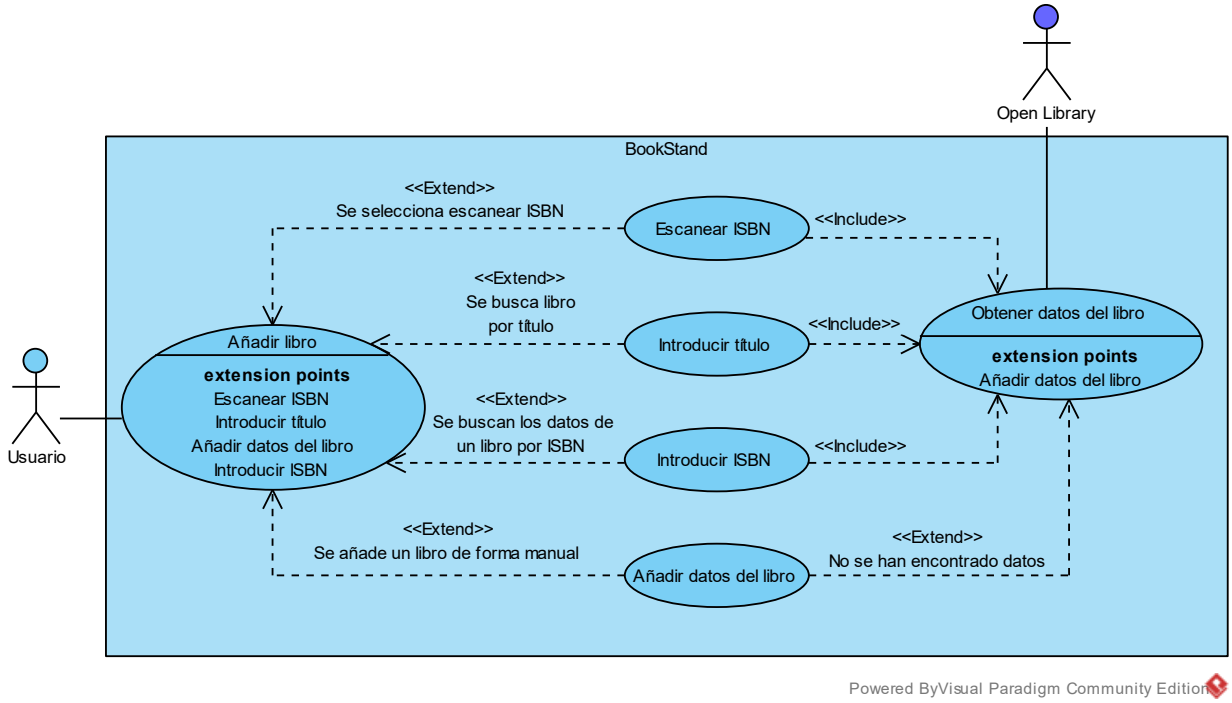

Ilustración 44 Diagrama de casos de uso para añadir un libro

CU-06	**Añadir un libro**	
Precondición	El usuario debe disponer de conexión a internet.	
Descripción	En este caso de uso se detalla el comportamiento de la aplicación para añadir libros a la biblioteca personal de	
Secuencia nominal	Paso	Acción
	1	El usuario solicita al sistema añadir un libro.
	2	Se selecciona el método para añadir los datos del libro
		2.a El usuario selecciona escanear el ISBN del libro, se realiza el caso de uso *Escanear ISBN*
		2.b El usuario selecciona buscar libro por título, se realiza el caso de uso *Introducir título*
		2.c El usuario selecciona introducir el ISBN de forma manual, se realiza el caso de uso *Introducir ISBN*
		2.d El usuario selecciona añadir datos del libro, se realiza el caso de uso *Añadir datos del libro*
	3	Se obtienen los datos del libro
		3.a Si se han seleccionado las opciones *Escanear ISBN*, *Introducir título* o *Introducir ISBN*; se obtienen los datos del libro a través de la API de Open Library

		3.b	Si se ha seleccionado la opción *Añadir datos del libro*, es el usuario el que ha proporcionado la información
	4	Se añaden los datos del libro a la base de datos	
Postcondición	El nuevo libro queda registrado en el sistema para ese usuario.		
Excepciones	Paso	Acción	
	3a	No se encuentran datos para el libro especificado en Open Library	
		E.1	El sistema informa al usuario y le ofrece la posibilidad de introducir los datos de forma manual.
		E.2	Si el usuario acepta introducir los datos de forma manual, se realiza el caso de uso *Añadir datos del libro*
Comentarios	Ninguno.		

Tabla 7 CU-06 Añadir un libro

4.2.4 Operaciones con un libro

En la Ilustración 45se detallan los casos de uso que reflejan las operaciones que se pueden realizar con un libro dentro de la aplicación BookStand. Además, en las tablas que aparecen más adelante, se describen con mayor precisión los casos de uso más importantes.

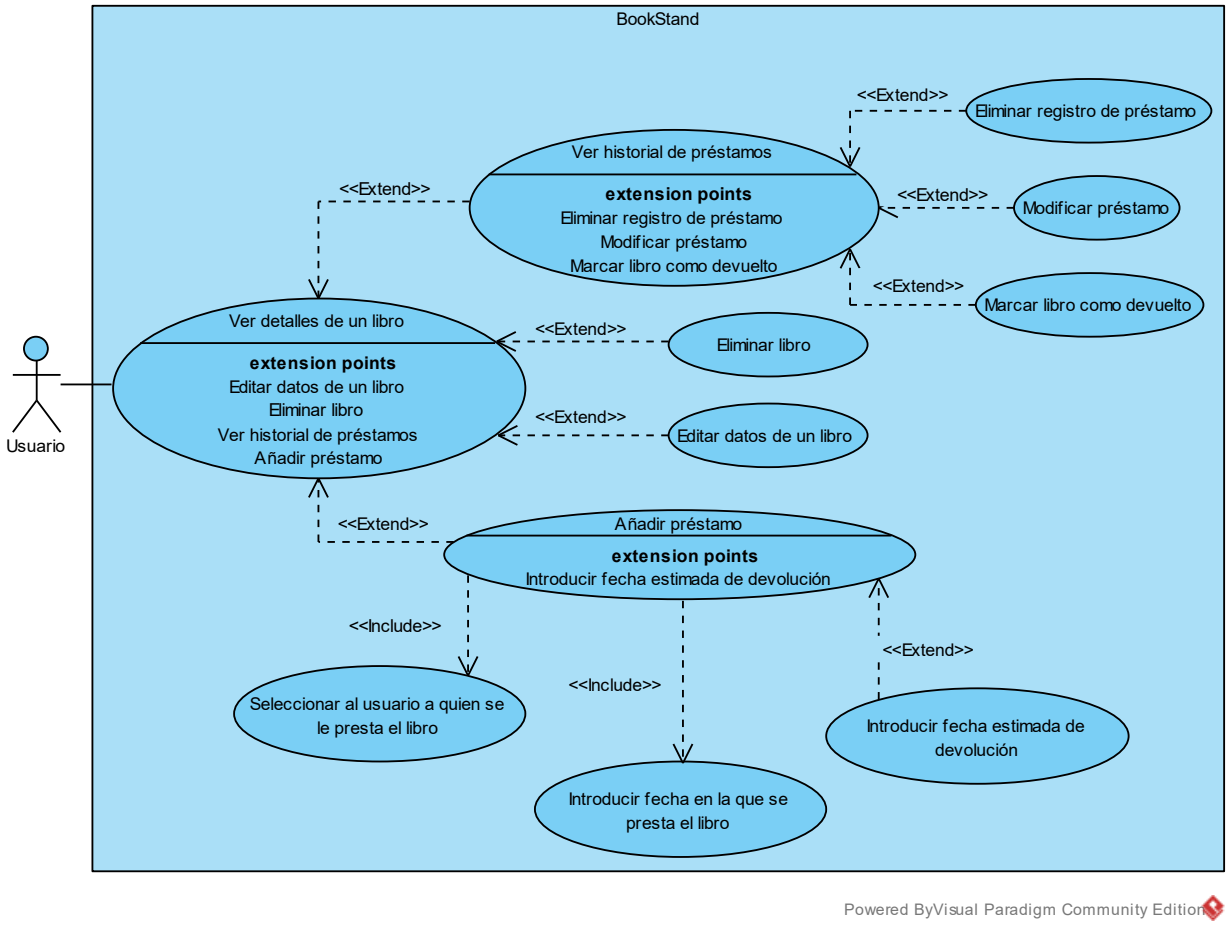

Ilustración 45 Diagrama de casos de uso para realizar operaciones con un libro

CU-07	**Ver detalles de un libro**	
Precondición	El usuario debe disponer de algún libro en su biblioteca.	
Descripción	En este caso de uso se detalla el comportamiento de la aplicación a la hora de ver los detalles de un libro, así como las operaciones que se permiten hacer con él.	
Secuencia nominal	Paso	Acción
	1	El usuario selecciona un libro de su biblioteca para ver los detalles.
Postcondición	Se muestra al usuario una pantalla con la información más relevante acerca del libro, así como las opciones que puede realizar con él: *Ver historial de préstamos*, *Eliminar libro*, *Editar datos de un libro* o *Añadir préstamo*.	
Excepciones	Ninguna.	
Comentarios	Ninguno.	

Tabla 8 CU-07 Ver detalles de un libro

CU-08	**Ver historial de préstamos**	
Precondición	El usuario debe disponer de algún libro en su biblioteca.	
Descripción	En este caso de uso se detalla el comportamiento de la aplicación para ver y gestionar el historial de préstamos de un libro.	
Secuencia nominal	Paso	Acción
	1	El usuario solicita al sistema ver el historial de préstamos para un libro concreto
Postcondición	Se muestra una lista con información acerca de todos los préstamos que se han hecho de ese libro. Se ofrece al usuario la posibilidad de *Eliminar registro del préstamo*, *Modificar el préstamo* o *Marcar el libro como devuelto.*	
Excepciones	Ninguna.	
Comentarios	Si no existe ningún préstamo para el libro seleccionado, simplemente se mostrará una lista vacía.	

Tabla 9 CU-08 Ver historial de préstamos

CU-09	**Añadir un préstamo**
Precondición	• El usuario debe disponer de algún libro en su biblioteca • Se debe tener algún contacto agregado en la aplicación. • El libro que se quiera prestar deberá ser propiedad del usuario.

	• El libro que se quiera prestar no puede estar siendo prestado a otro usuario. • Sólo puede iniciar el proceso del préstamo el usuario que va a prestar el libro.	
Descripción	En este caso de uso se detalla el comportamiento de la aplicación para añadir un préstamo al sistema.	
Secuencia nominal	Paso	Acción
	1	El usuario solicita al sistema añadir un préstamo
	2	El usuario selecciona a quién presta el libro (Se realiza el caso de uso *Seleccionar al usuario a quien se presta el libro*)
	3	Se añade la fecha en la que se hace el préstamo (Se realiza el caso de uso *Introducir fecha en la que se presta el libro*)
	4	Si el usuario lo desea, establece una fecha aproximada para la devolución del libro (En este caso, se realiza el caso de uso *Introducir fecha estimada de devolución*)
Postcondición	El préstamo queda registrado en el sistema y se envía una notificación al usuario deudor para que acepte el préstamo.	
Excepciones	Ninguna.	
Comentarios	Si no se cumple alguna de las precondiciones, se informa al usuario y se cancela el caso de uso.	

Tabla 10 CU-09 Añadir préstamo

4.3 Modelo de datos de BookStand

Este apartado tiene como objetivo explicar el modelo de datos utilizado en la aplicación. Antes de esto, conviene mencionar que en el desarrollo de la aplicación no se ha utilizado orientación a objetos, sino que se han definido los elementos principales de la aplicación a través de alias de tipos [45]. En la Ilustración 46[7] se muestran los principales tipos utilizados:

[7] En este diagrama, los campos cuyo nombre termina con '?' son opcionales.

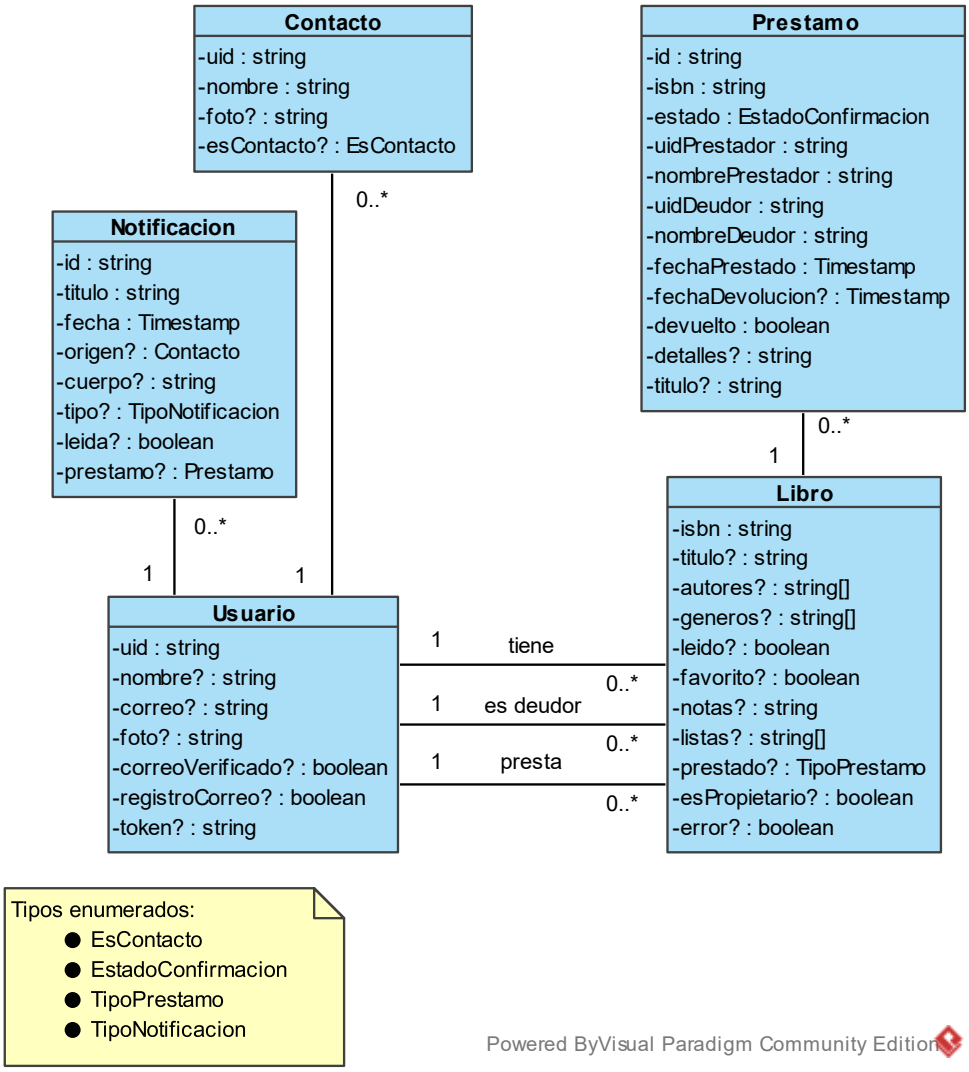

Ilustración 46 Tipos de objetos del modelo de datos

En las siguientes líneas se describe brevemente qué representan los elementos mostrados en el diagrama anterior:

- `Usuario`. Tal y como su nombre indica, estos objetos representarán a los usuarios de un sistema y recogerán información básica acerca de los mismos, como su nombre (campo `nombre`) o su foto de perfil (campo `foto`). Es interesante comentar dos de los campos más importantes de estos objetos: `uid` y `token`.

  El campo `uid` representa el identificador único que crea el módulo de Firebase Authentication cuando se registra un nuevo usuario en la aplicación, y se utilizará para identificar de forma unívoca a cada usuario en el sistema.

  Por otra parte, el campo `token` almacenará un identificador asociado a la instancia que permitirá la recepción de mensajes de Cloud Messaging.

- `Libro`. Este objeto almacenará toda la información relativa a un libro en nuestra aplicación, como el título, los autores o los géneros a los que pertenece. Los libros se identifican por su ISBN, un identificador de libros comerciales que identifican de forma única a cada edición de un libro.

- `Prestamo`. Los objetos de este tipo almacenan todos los datos relativos al préstamo de un libro entre dos usuarios. Cada usuario se identifica dentro de un préstamo como "deudor", si es la persona a la que se presta el libro o "prestador" si es quien lo presta. Estas relaciones quedan reflejadas en los campos `uidDeudor` y `uidPrestador` respectivamente.

De estos objetos se destaca el campo `estado` que, como su propio nombre indica, sirve para reflejar el estado en el que se encuentra un préstamo en el sistema. Los valores que puede tomar se muestran en la Ilustración 47. En el apartado 4.5.3 se verá de forma más detallada su utilidad en el sistema.

```
export enum EstadoConfirmacion {
    pendienteCreacion = 'PEND_CREACION',
    confirmadoCreacion = 'OK_CREACION',
    denegadoCreacion = 'DEN_CREACION',
    pendienteModificacion = 'PEND_MODIFICACION',
    confirmadoModificacion = 'OK_MODIFICACION',
    degenadoModidicacion = 'DEN_MODIFICACION',
    pendienteDevolucion = 'PEND_DEVOLUCION',
    confirmadoDevolucion = 'OK_DEVOLUCION',
    denegadoDevolucion = 'DEN_DEVOLUCION',
    ocioso = 'OCIOSO',
}
```

Ilustración 47 Tipo enumerado que representa el estado de un préstamo

- `Contacto`. Los objetos de este tipo representan las personas que un usuario ha agregado a su aplicación. Cuando un usuario es contacto de otro, pueden prestarse libros entre ellos.
- `Notificacion`. Estos objetos representan las notificaciones de Cloud Messaging que reciben los usuarios. Hay diferentes tipos de notificaciones (de solicitud de amistad, informativas, actualización de présamo, etc.) y estos quedan reflejados en el campo `tipo`.

De la misma forma, recalcaremos las relaciones que existen entre los objetos de la Ilustración 46:

- Un **usuario** puede tener varios **contactos** agregados
- Un **usuario** tendrá un conjunto de **notificaciones** que recibe
- Un **usuario** tendrá un conjunto de **libros** en su biblioteca
    - De esos libros, algunos podrán **ser prestados** por otros usuarios, y su relación con ellos será de “deudor”.
    - Otros libros los **estará prestando** a sus contactos, su relación en este caso será de “prestador”.
- Un **préstamo** almacena todos los datos relativos a un **libro** que está siendo prestado
- Un **libro** puede ser **prestado** varias veces (a distintos usuarios o al mismo), pero nunca puede ser prestado a varios usuarios al mismo tiempo.

Para acabar este apartado, es necesario explicar dónde se encuentra la lógica de la aplicación que maneja estos objetos ya que, como comentábamos al inicio de este apartado, no se han usado clases para representar el modelo de datos. En este caso, se ha hecho uso de módulos de funciones y clases auxiliares que realizan operaciones utilizando directamente con los objetos del modelo. En la Ilustración 48 se detallan las relaciones ente ellos:

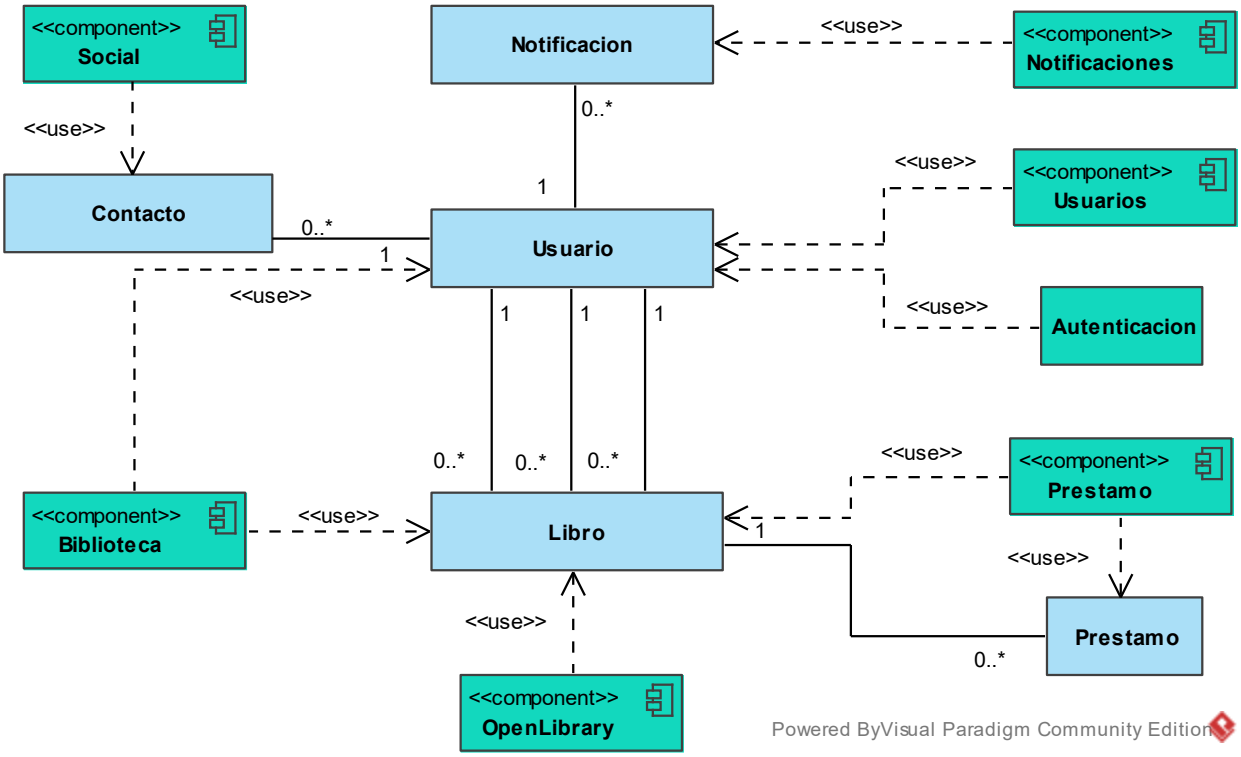

Ilustración 48 Relación de los módulos y las clases auxiliares con los objetos del modelo de datos

En las imágenes que se muestran a continuación se detallan las funciones y los métodos que forman parte de cada uno de los módulos y clases auxiliares. Por motivos de simplicidad se omiten los parámetros de las funciones.

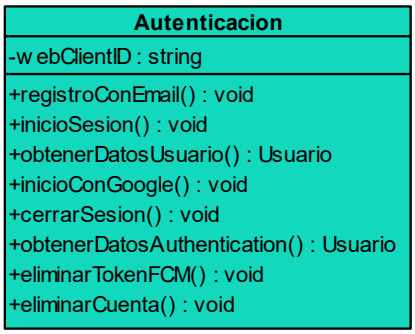

Ilustración 49 Clase Autenticación

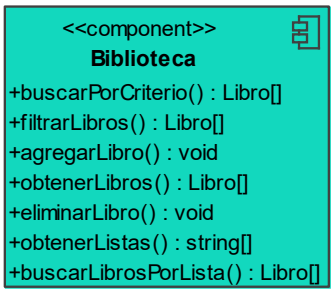

Ilustración 50 Módulo Biblioteca

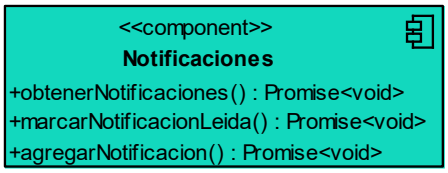

Ilustración 51 Módulo Notificaciones

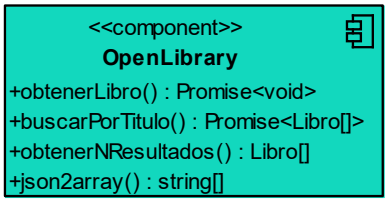

Ilustración 52 Módulo OpenLibrary

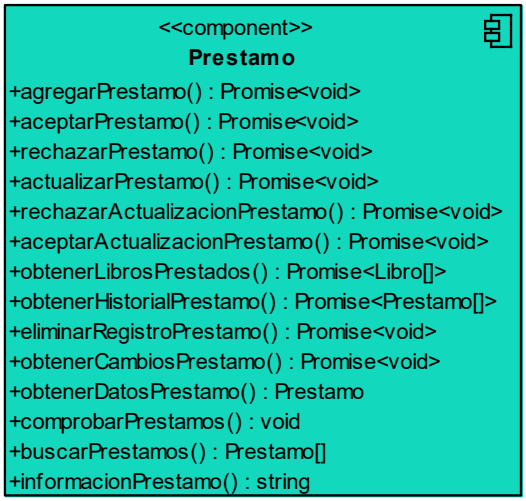

Ilustración 53 Módulo Préstamo

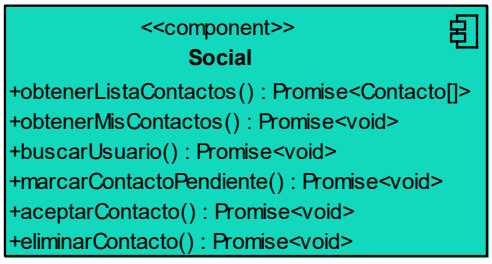

Ilustración 54 Módulo Social

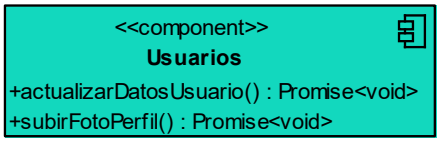

Ilustración 55 Módulo Usuarios

4.4 Diagrama de la base de datos

En este apartado describiremos cómo trasladamos el modelo de datos a nuestra base de datos Cloud Firestore. Antes de continuar, resulta necesario describir ciertas peculiaridades de este tipo de bases de datos.

Como comentábamos en el apartado 2.2.2.2, Firestore es una base de datos no relacional que almacena los datos en colecciones y documentos [19]:

Un documento es la unidad básica de almacenamiento. Un documento es un registro que usa pocos recursos y contiene campos con valores asignados. Cada documento se identifica con un nombre. Se organizan contenedores llamados colecciones. Dentro de un documento pueden crearse subcolecciones: colecciones asociadas a un documento.

Este tipo de bases de datos están pensadas para ser fácilmente escalables, rápidas y flexibles, por lo que son idóneas para las aplicaciones móviles.

Sin embargo, al estructurar los datos de forma jerárquica, las bases de datos NoSQL solo permiten la realización de consultas relativamente simples, en comparación con las consultas en SQL. Como consecuencia, habrá veces que para obtener información de nuestra base de datos necesitemos realizar varias consultas en cadena, lo que aumentaría los costes del servicio. Por este motivo, las bases de datos no relacionales suelen presentar información redundante, ya que así podemos reducir el número de lecturas.

Un lector agudo habrá podido darse cuenta de esta redundancia al observar el modelo de datos presentado en el capítulo anterior, por ejemplo, a la hora de añadir tanto los identificadores como los nombres de los usuarios que participan en un préstamo o el ISBN y el título de un libro. Los nombres los podríamos haber obtenido a partir del identificador de los usuarios; el título del libro se puede obtener a partir del ISBN. Al incluir todos estos datos en el mismo documento, cuando representemos el préstamo en nuestra aplicación bastará con realizar una única consulta, en lugar de cuatro.

Una vez aclarado esto, en la Ilustración 56 se muestra el diagrama de la base de datos utilizada en el proyecto:

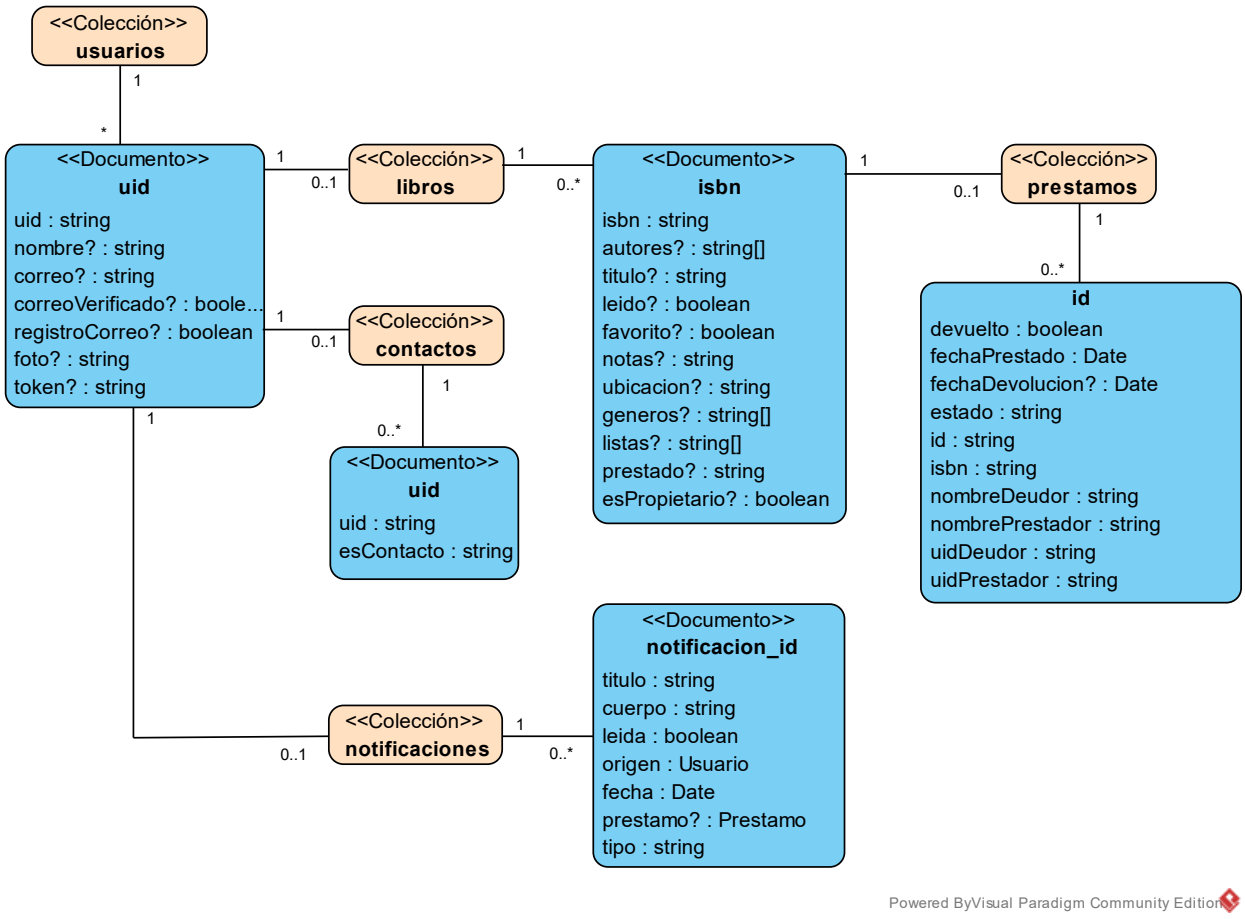

Ilustración 56 Diagrama de la base de datos no relacional Cloud Firestore

Observando el diagrama podemos concluir:

- La colección “usuarios” almacena cada usuario del sistema como un documento identificado por el uid que se obtiene de Firebase Authentication.
- Cada usuario del sistema tendrá una colección llamada “libros” donde almacenará como documentos los datos de los libros que incluya en su biblioteca. Estos documentos se identificarán por un ISBN.
    - Cada libro tendrá una colección llamada “prestamos” en la que se almacenarán documentos con cada préstamo que se haga de ese libro. Estos documentos se identifican con un id de préstamo.
- Cada usuario del sistema tendrá una colección llamada “contactos” donde se almacenará en documentos identificados por un uid a todos los usuarios a los que ha agregado.
- Cada usuario tendrá una colección llamada “notificaciones” en la que se almacenarán como documentos las notificaciones que se le envíen al usuario. Estos documentos se identifican con un id de notificación.

4.5 Diagramas de secuencia

Una vez que hemos definido tanto el modelo de datos como los componentes que forman parte del sistema, estamos en condiciones de ver cómo interactúan entre ellos para conseguir realizar las diferentes tareas de la aplicación. Para ello, en este apartado se proponen varios diagramas de secuencia para ilustrar tres de las tareas más importantes que se realizan en la aplicación: registrar a los usuario en el sistema, obtener los datos de un libro a través de Open Library y realizar el préstamo de un libro.

4.5.1 Registrar un usuario en la aplicación

En el diagrama de secuencia que se representa en la Ilustración 57 se detalla cómo se registra y autentica un usuario en la aplicación. Los elementos que intervienen en este diagrama son los siguientes:

- Usuario: usuario de la aplicación. Es la persona que quiere registrarse en el sistema.
- Autenticación (en color verde). Clase de BookStand definida para controlar la interacción de la app con el módulo de autenticación de Firebase. En la Ilustración 49 del apartado Modelo de datos de BookStand se describe con más detalle.
- Redux. Módulo que controla el estado global de la aplicación bajo el patrón Redux.
- Authentication (en color amarillo). Módulo de Firebase que actúa de servidor de autenticación.
- Cloud Firestore. Módulo de Firebase que actúa de base de datos.

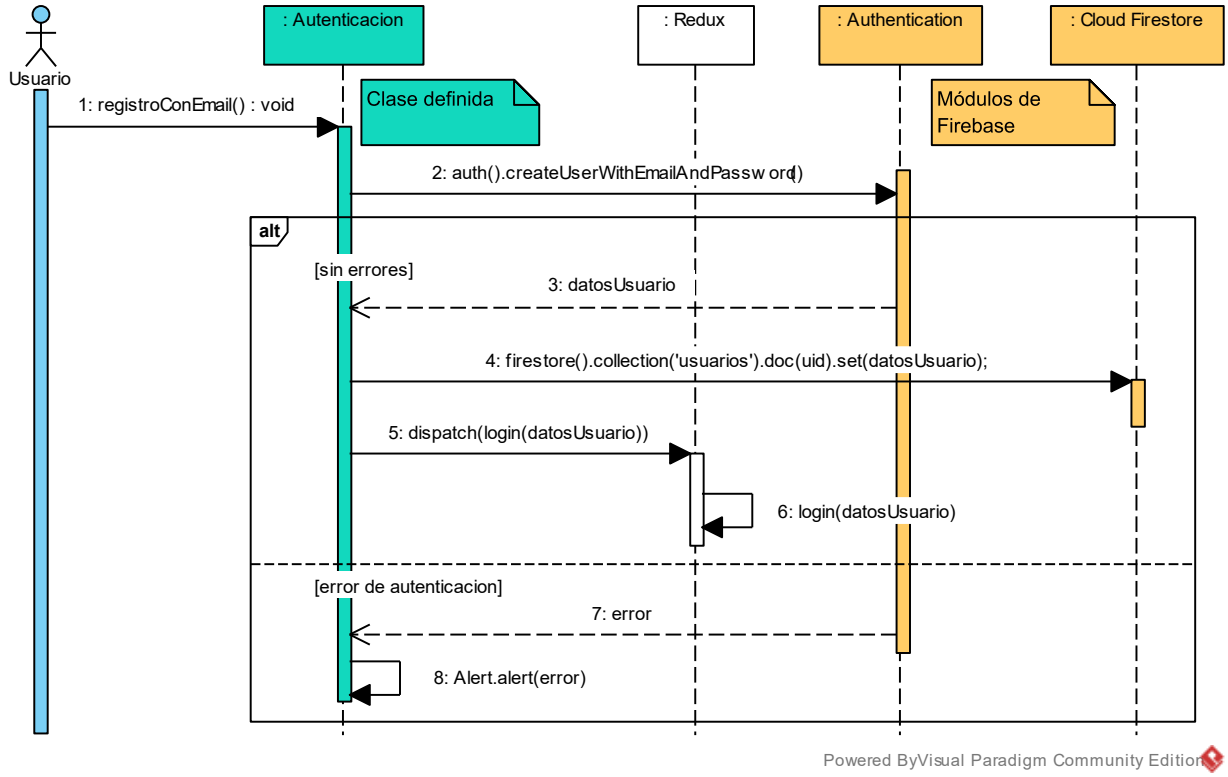

Ilustración 57 Diagrama de secuencia para registrar un usuario en la aplicación

4.5.2 Obtener los datos de un libro

En la Ilustración 58 se presenta el diagrama de secuencia para explicar cómo se obtienen los datos de un libro a partir del ISBN que hemos escaneado. Los elementos que participan en este diagrama son:

- Usuario, Cloud Firestore y Redux (ya descritos en el apartado anterior).
- React Native Camera. Módulo que controla el acceso a la cámara del dispositivo y el reconocimiento de los códigos de barras.
- BookStand. Aplicación móvil desarrollada en React Native.
- OpenLibrary (en verde). Parte de BookStand que se encarga de manejar el acceso a Open Library.
- Biblioteca. Parte de BookStand que maneja las operaciones sobre la biblioteca personal de un usuario.
- Open Library (en azul). Servidor de Open Library.

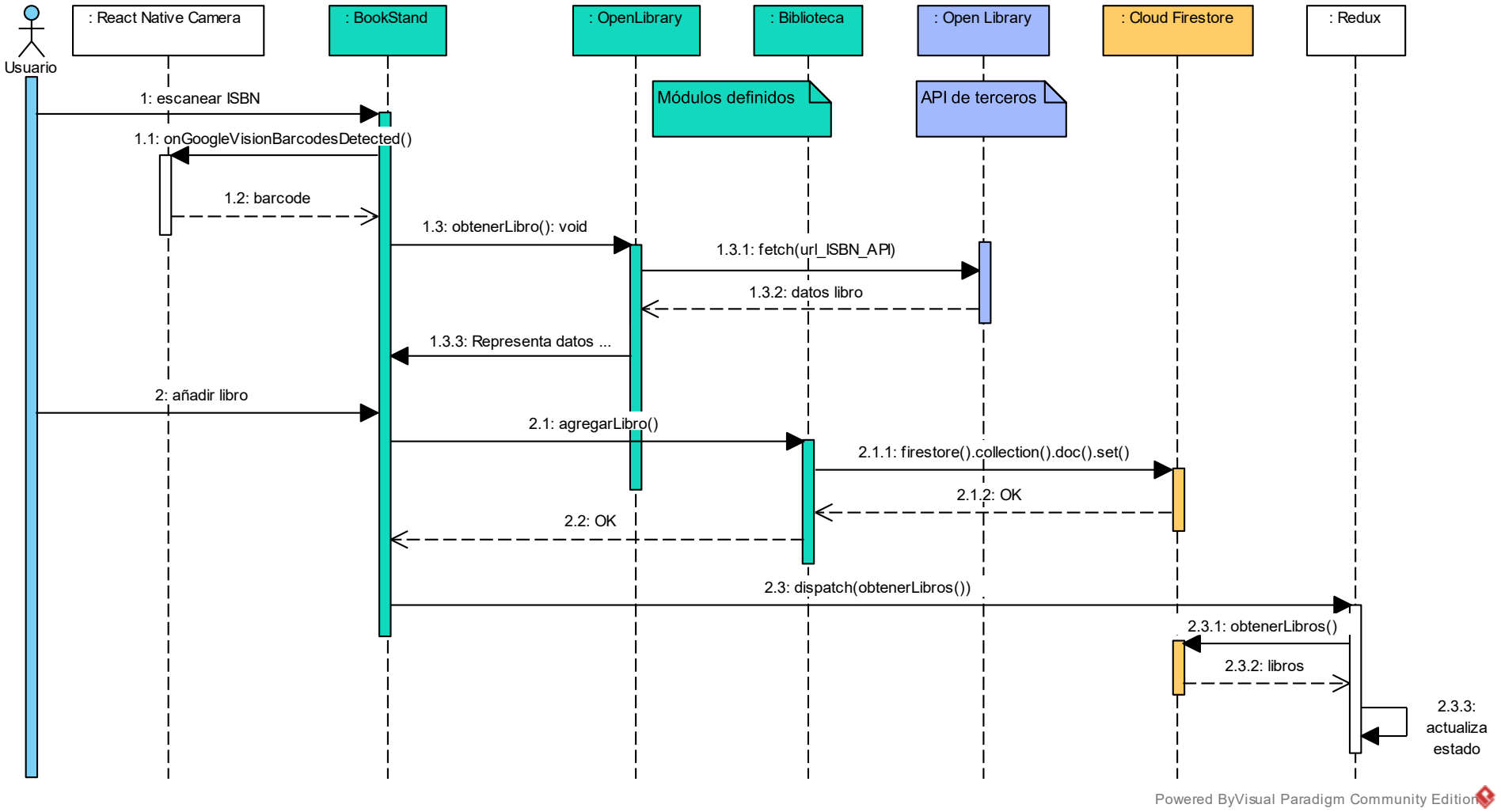

Ilustración 58 Diagrama de secuencia para obtener los datos de un libro

4.5.3 Prestar un libro a un usuario

En el diagrama de secuencia que aparece en la Ilustración 59 se describen las operaciones necesarias para prestar un libro entre dos usuario de BookStand. Los elementos que intervienen son:

- Usuario A. Es el usuario que inicia el préstamo. Debe ser la persona que vaya a prestar el libro.
- Usuario B. Es el usuario que acepta el préstamo.
- BookStand. Aplicaciones cliente para cada usuario.
- Cloud Firestore. (Ya descrito en apartados anteriores)
- Cloud Functions. Parte de Firebase donde se ejecutan las funciones de Cloud.
- Cloud Messaging. Parte de Firebase encargada de enviar mensajes a los dispositivos.

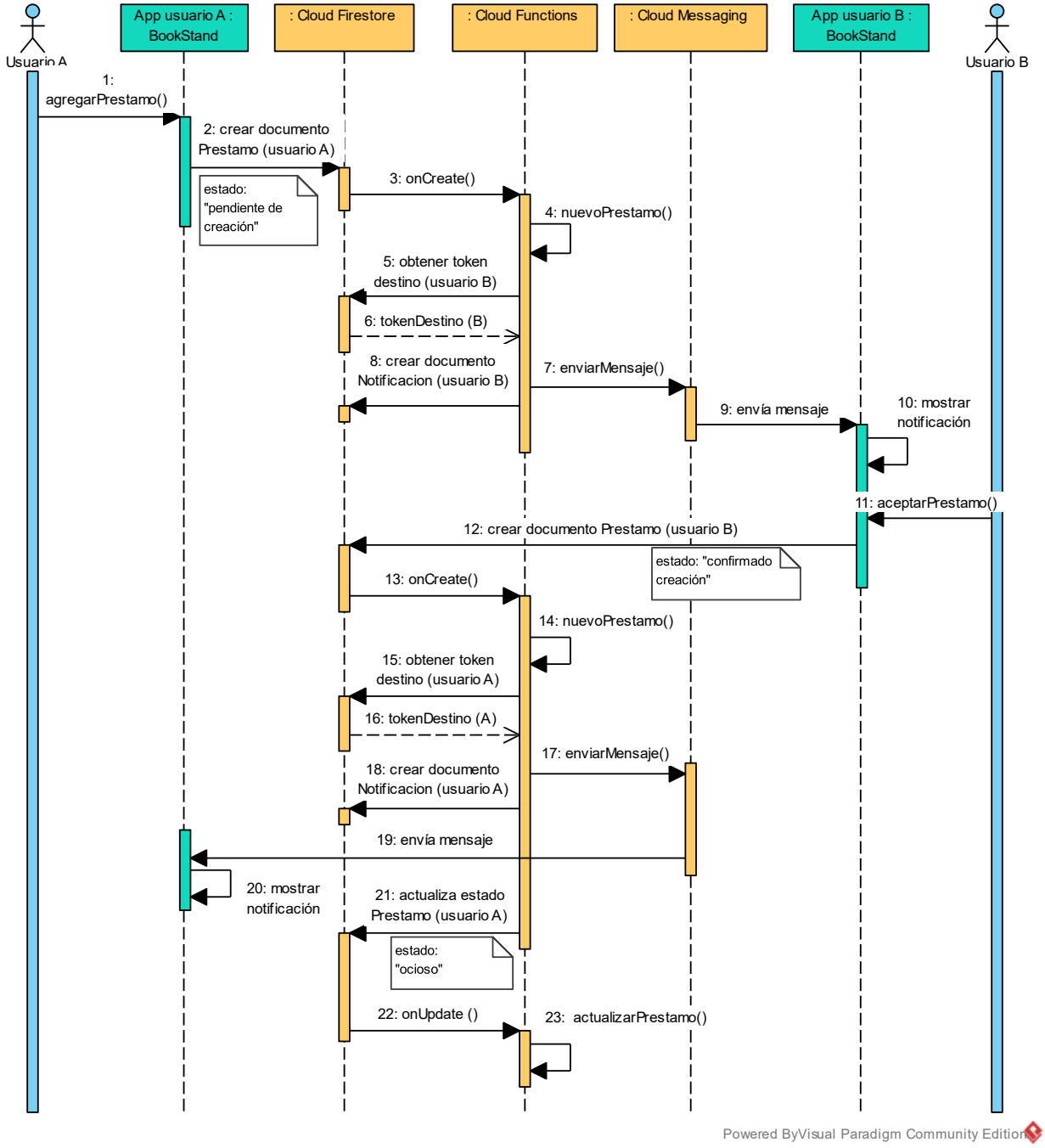

Ilustración 59 Diagrama de secuencia para prestar un libro a un usuario

Resulta interesante comentar brevemente cómo actúa el módulo de Cloud Functions en este escenario.

En este diagrama de secuencia intervienen las funciones `nuevoPrestamo` y `actualizarPrestamo`. La ejecución de estas se desencadena a partir de eventos de creación o actualización de algunos documentos de la base de datos, como se refleja en la secuencia de los mensajes 2-3-4 o 21-22-23 de la Ilustración 59.

Además, dependiendo del estado del préstamo las funciones de Cloud deberían realizar acciones diferentes: las notificaciones que se envían serán distintas, se realizan operaciones adicionales, etc. Para conseguir esto, se hace uso del atributo "estado" de los objetos de tipo préstamo, tal y como se mencionó en el apartado 4.3.

Este funcionamiento se puede extrapolar al resto de operaciones relacionadas con las funciones de Cloud, por lo que se podría deducir el comportamiento del sistema si en lugar de aceptar el préstamo, el usuario B lo denegase o si en lugar de crear un préstamo se modificasen sus datos.

Para conocer más acerca de este tema, se puede consultar el Anexo A: Código de las Funciones de Cloud de Firebase.

5 INTERFAZ DE USUARIO Y FUNCIONALIDAD

La interfaz de usuario es como un chiste: si tienes que explicarla entonces no es tan buena.

Martin LeBlanc

En este capítulo se muestra la interfaz de usuario de la aplicación a través de las diferentes pantallas y de la navegación.

5.1 Registro e inicio de sesión

En la Ilustración 60 se muestra la primera pantalla que se presenta al usuario al instalar la aplicación. Esta pantalla permite iniciar sesión o registrar a un nuevo usuario en la aplicación de formas diferentes: mediante correo y contraseña o mediante una cuenta de Google existente.

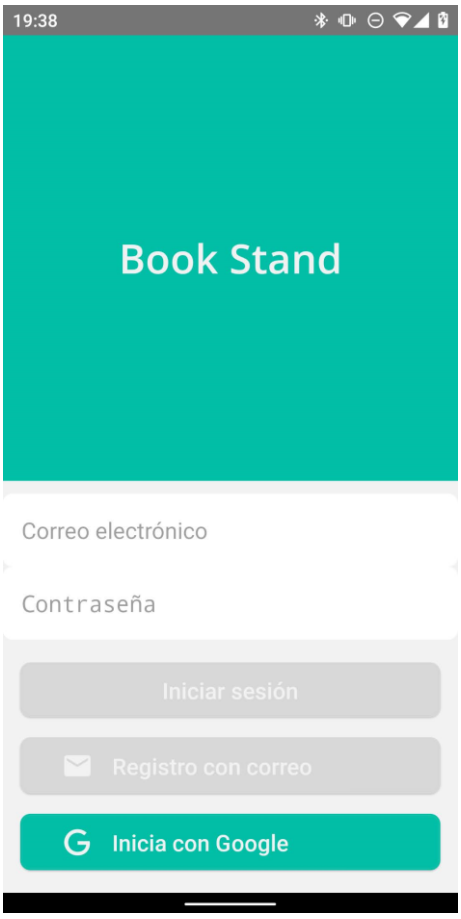

Ilustración 60 Pantalla de registro e inicio de sesión

5.1.1 Inicio de sesión o registro mediante correo

Si el usuario desea acceder a BookStand mediante un correo electrónico, deberá introducir dicho correo más la contraseña asociada en los campos que aparecen en la pantalla. Una vez estén rellenos estos campos, se desbloquearán los botones de inicio de sesión y registro con correo, permitiendo al usuario realizar dichas acciones, tal y como se muestra en la Ilustración 61.

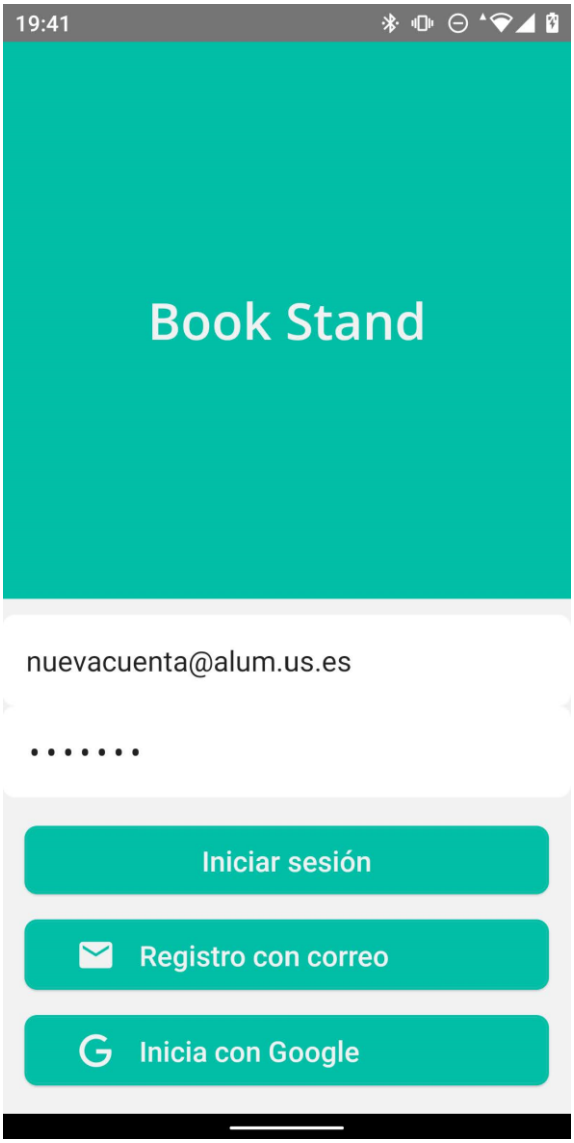

Ilustración 61 Al introducir los datos de usuario en la pantalla de registro, se habilitan las opciones de inicio de sesión y registro

Si ocurre algún error durante este proceso como, por ejemplo, que un usuario intente registrarse con un correo que ya esté en uso, que se registre con una contraseña demasiado débil o que introduzca la contraseña equivocada, se mostrará un error en pantalla y se pedirá al usuario que lo rectifique, tal y como se observa en la Ilustración 62:

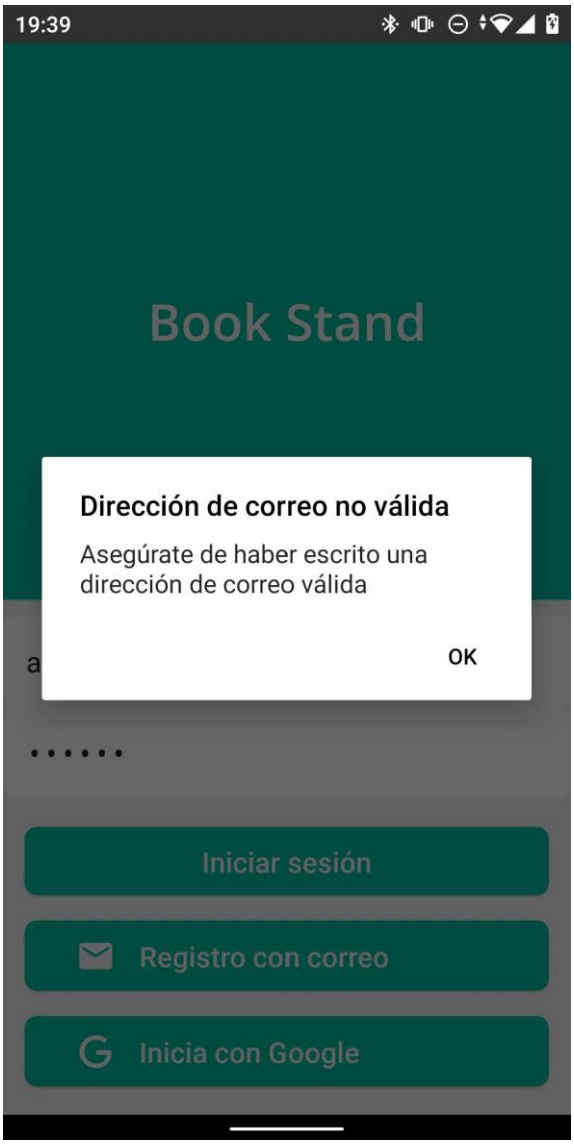

Ilustración 62 Mensaje de error al introducir una dirección de correo no válida en el registro de un usuario

5.1.2 Acceso mediante una cuenta de Google

Si el usuario decide acceder a BookStand mediante una cuenta de Google existente, sólo debe presionar el botón “Inicia con Google”, ya sea para registrarse o para iniciar sesión. Una vez haya hecho esto, aparecerá una tarjeta con todas las cuentas de Google que tenga registradas en el dispositivo, permitiendo al usuario seleccionar la deseada. Esto lo podemos ver en la Ilustración 63:

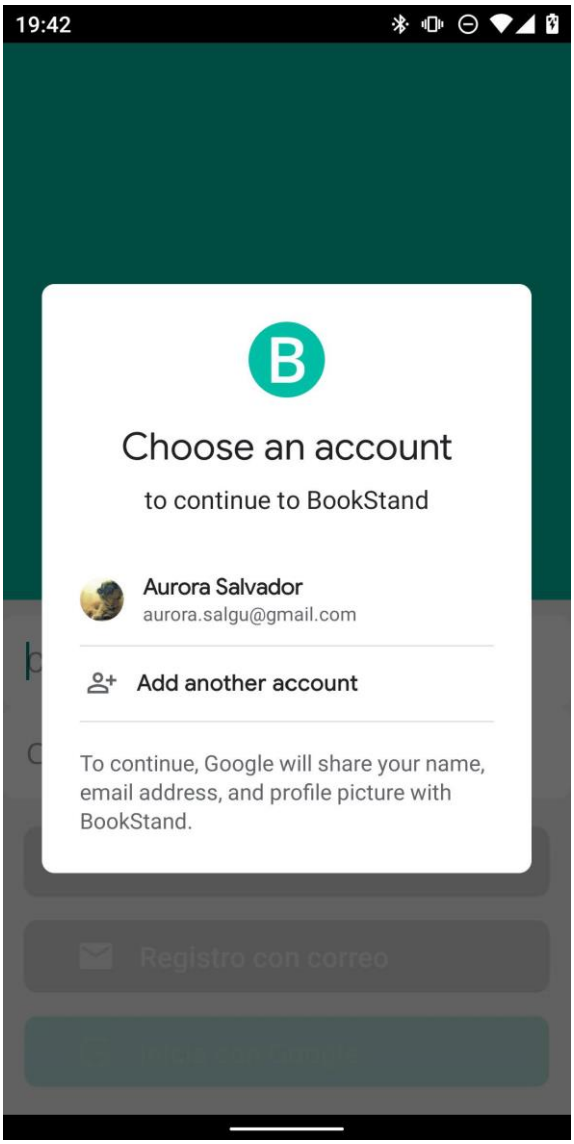

Ilustración 63 Acceso a BookStand mediante una cuenta de Google

5.2 Navegación a través de las diferentes pantallas

Una vez el usuario se haya registrado correctamente o haya iniciado sesión en el sistema se le mostrará el resto de las pantallas de la aplicación. Antes de comenzar a describir cada una de estas pantallas, es importante comentar cuáles son y cómo puede navegar el usuario a través de ellas.

Las pantallas más importantes de la aplicación son:

- Pantalla Principal. Esta pantalla permite al usuario ver las últimas notificaciones que ha recibido, además de interactuar con ellas. También permite que un usuario añada libros a su biblioteca desde aquí.
- Pantalla Biblioteca. En esta pantalla el usuario puede consultar los libros que tiene agregados a su biblioteca personal, realizar búsquedas y ver sus detalles.
- Pantalla Mis Contactos. En esta pantalla se ofrece la opción a los usuarios de buscar y agregar otros contactos a los que poder prestar libros.
- Pantalla Perfil. En esta pantalla se concentran las operaciones relacionadas con la gestión de la cuenta del usuario.

- Pantalla Acerca de BookStand. Esta pantalla proporciona al usuario información adicional acerca de la aplicación o permitir descargar el contenido de la biblioteca en formato CSV[8].

La navegación entre estas pantallas se realiza a través de un menú de cajón, tal y como se puede ver en la Ilustración 65. A este menú se puede acceder presionando el botón menú del encabezado, que se muestra en la Ilustración 64, o mediante un gesto de deslizamiento desde el extremo izquierdo de la pantalla.

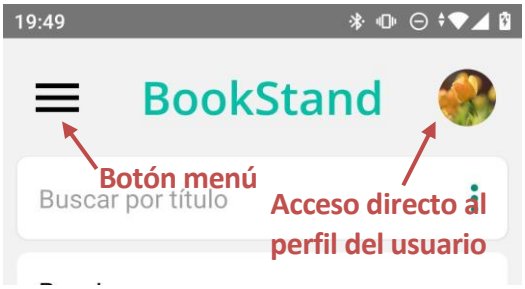

Ilustración 64 Encabezado de la aplicación

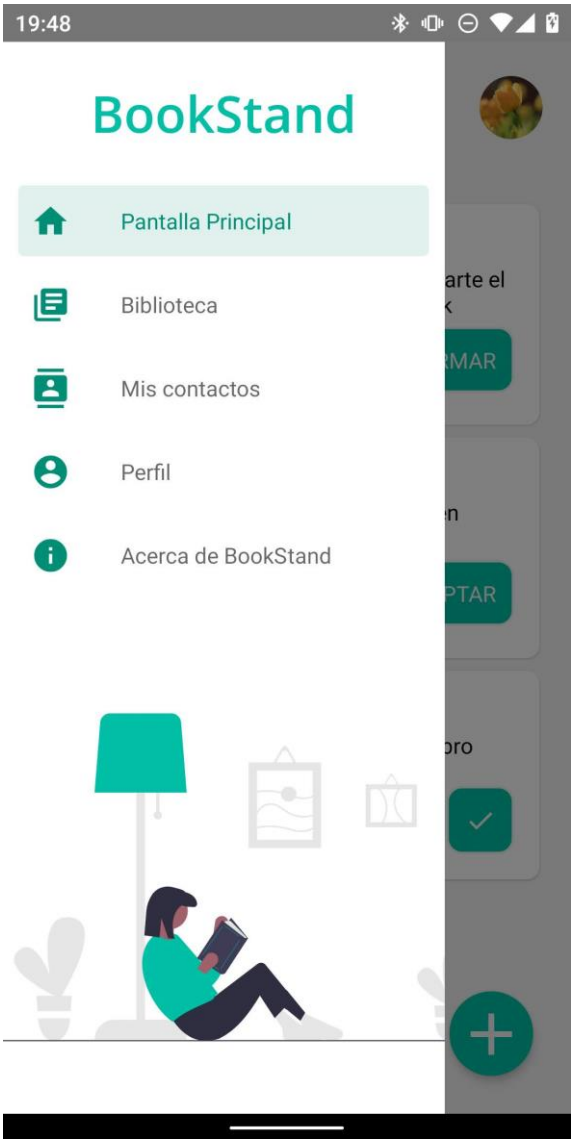

Ilustración 65 Menú de cajón de BookStand

[8] Un documento CSV (Comma Separated Values) es un archivo de texto plano que contiene una lista de datos en formato tabular.

5.3 Pantalla Principal

En la Ilustración 66 podemos ver una captura de la pantalla principal de la aplicación. Como se comentó en el apartado anterior, esta pantalla nos permite ver e interactuar con el *feed* de notificaciones y agregar nuevos libros a nuestra biblioteca. En las siguientes subsecciones se describen estos procesos de forma más detallada.

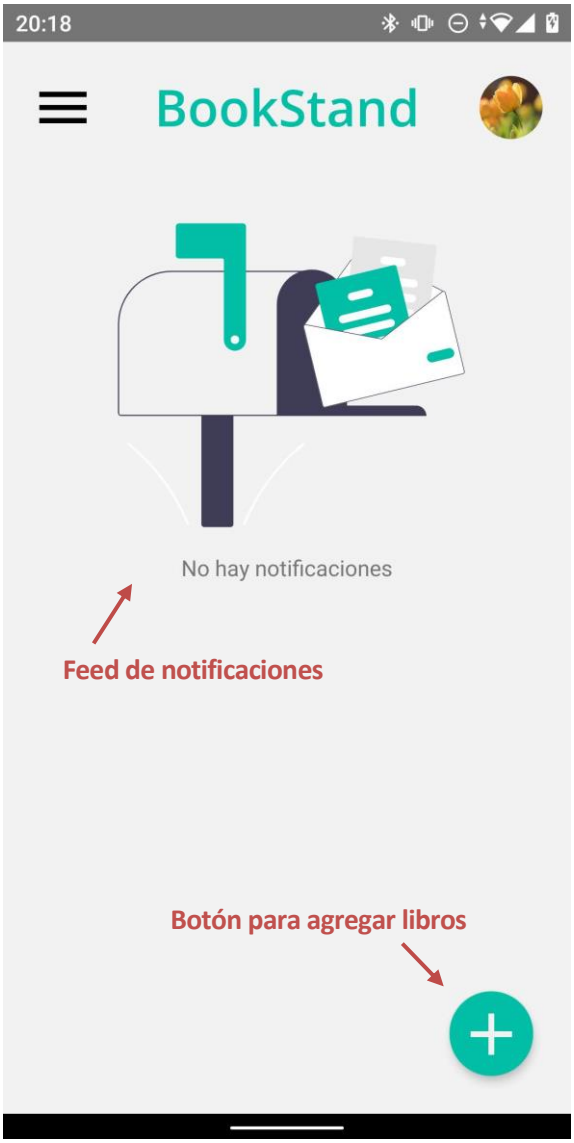

Ilustración 66 Pantalla principal sin notificaciones

5.3.1 Feed de Notificaciones

El *feed* de notificaciones no es más que una lista con los últimos mensajes recibidos en el dispositivo a través de Cloud Messaging. Es el lugar en el que el usuario puede interactuar con sus contactos y aceptar o rechazar préstamos, modificaciones de los préstamos y solicitudes de amistad.

En la Ilustración 67 podemos ver un conjunto de diferentes notificaciones que ha recibido un usuario, así como las diversas acciones que se pueden realizar sobre cada una de ellas.

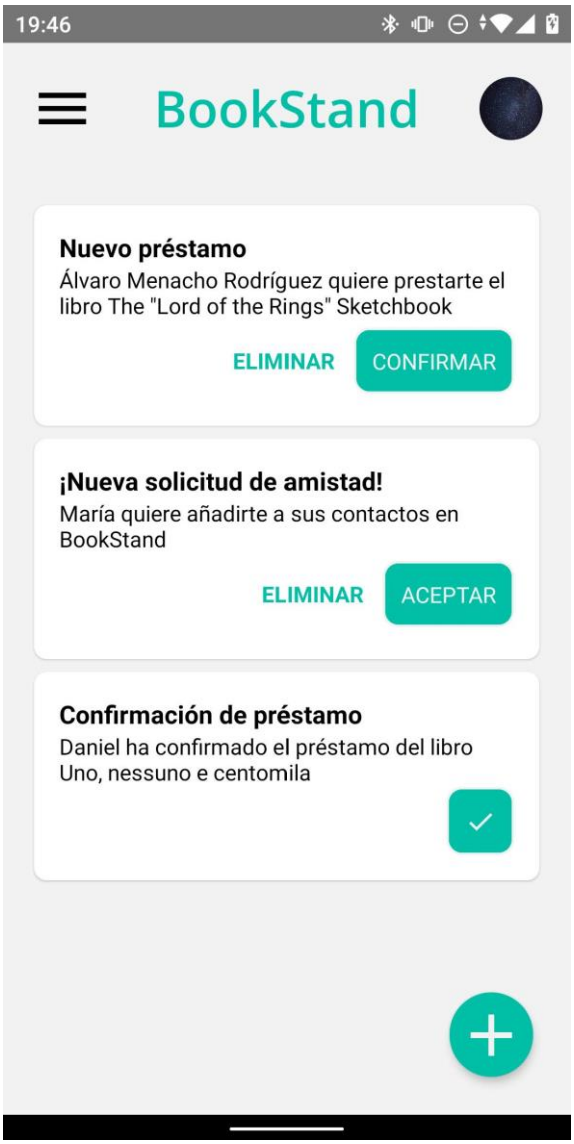

Ilustración 67 Notificaciones en la pantalla principal

5.3.2 Añadir libros

Al presionar el botón para añadir libros de la pantalla principal aparece un menú con un conjunto de botones, como se muestra en la Ilustración 68, que permiten al usuario añadir libros de diferentes formas. En los siguientes apartados se describen cada una de estas formas.

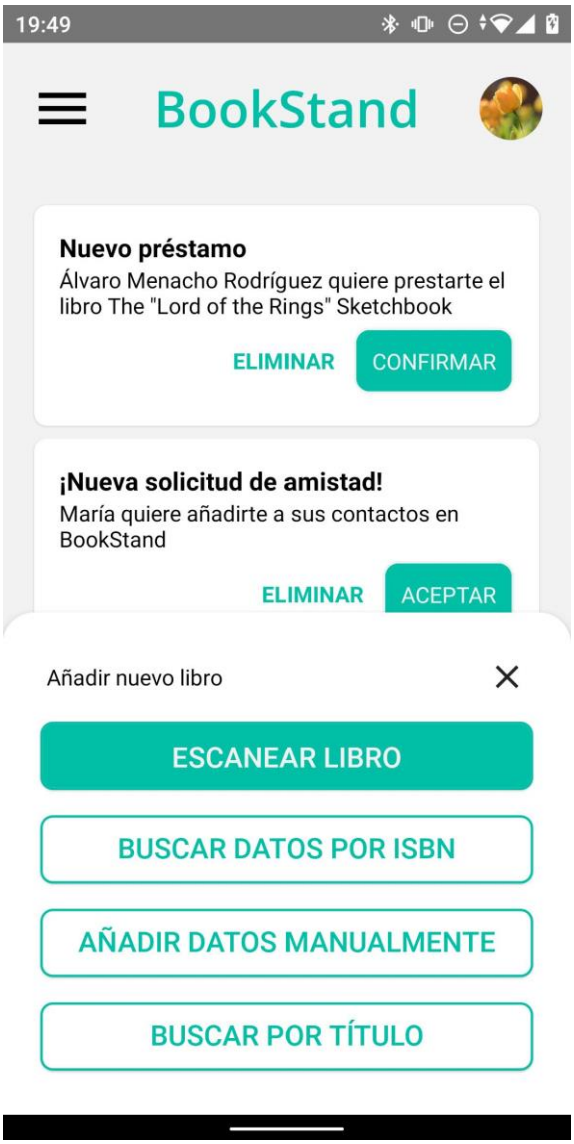

Ilustración 68 Menú con las diferentes opciones para añadir un libro

5.3.2.1 Escanear libro

Mediante esta opción se permite al usuario escanear el ISBN de un libro con la cámara de su dispositivo móvil. Antes de acceder a la cámara, la aplicación solicita al usuario acerca de los permisos necesarios, tal y como se muestra en la Ilustración 69.

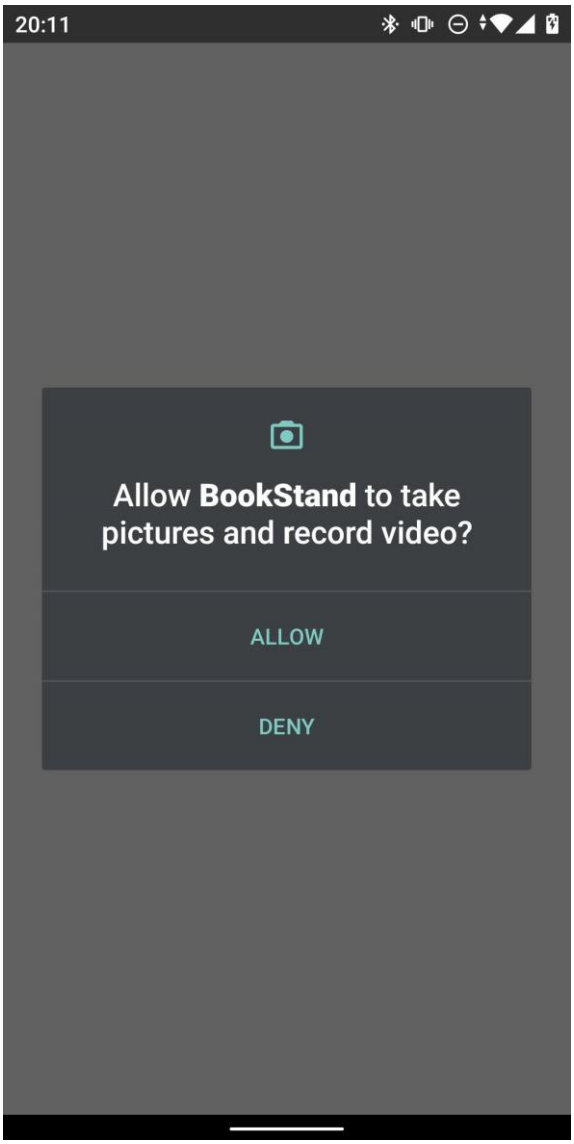

Ilustración 69 Solicitud de los permisos para acceder a la cámara del dispositivo

Una vez se le hayan otorgado estos permisos a la aplicación, accederemos a la cámara del dispositivo. En pantalla aparecerá un recuadro que sirve de guía al usuario para que alinee el código de barras con el ISBN del libro y este se pueda leer de forma óptima.

Cuando la aplicación reconozca el ISBN, aparecerá el número en pantalla, tal y como se muestra en la Ilustración 70. En este momento, si el usuario presiona el botón con el icono de la cámara, se buscan los datos del libro en el registro de Open Library y se presenta el resultado por pantalla.

Ilustración 70 Se escanea el código de barras con el ISBN del libro

Ilustración 71 Presentación de los datos obtenidos tras realizar la búsqueda del libro en Open Library

En este punto, el usuario puede añadir el libro a su biblioteca personal mediante el botón "Añadir" o editar los datos del libro antes de hacerlo mediante el botón "Editar". Si lo desease, también podría cancelar la operación con el botón "Cancelar". En el apartado 5.4.1 se detallará el proceso para editar los datos de un libro de forma más detallada.

Este método para escanear libros resulta idóneo para hacer un inventariado rápido de los libros físicos que tiene un usuario. Sin embargo, es posible que busquemos un libro que no tenga un registro en Open Library del que podamos sacar información. En este caso el sistema informa al usuario del error y propone al usuario añadir la información de forma manual, como se muestra en la Ilustración 72. En el apartado Añadir datos manualmente 5.3.2.3 se describe el proceso para añadir los datos de un libro de forma manual.

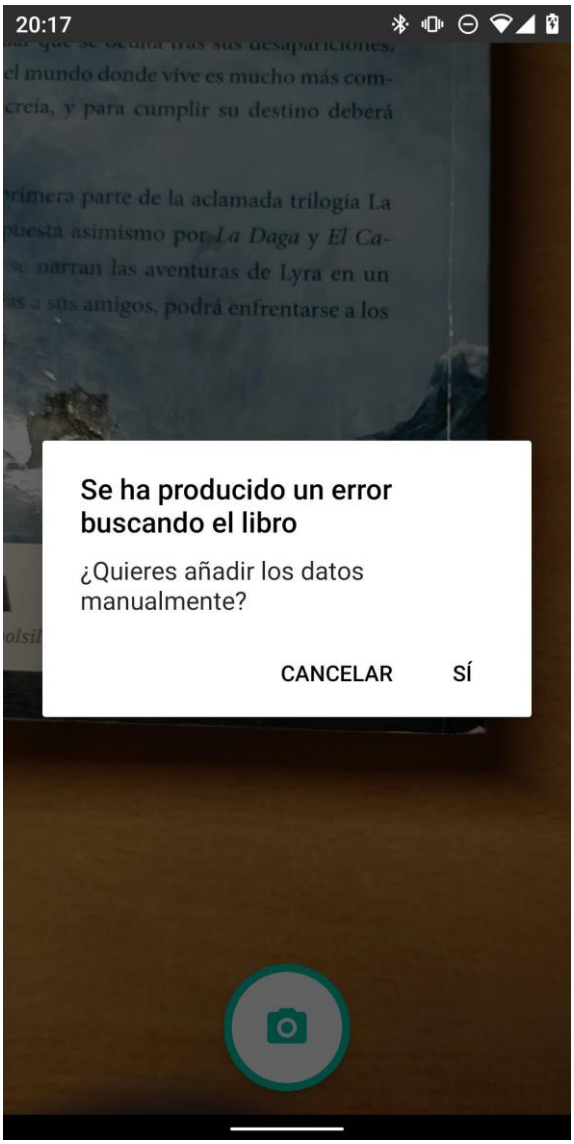

Ilustración 72 Al no encontrar los datos para el libro, el sistema propone añadir los datos de forma manual

Para terminar, es necesario comentar que, para evitar que un usuario añada un libro varias veces a su biblioteca personal, se informa al usuario cuando se quiera escanear un libro que ya tiene registrado (Ilustración 73).

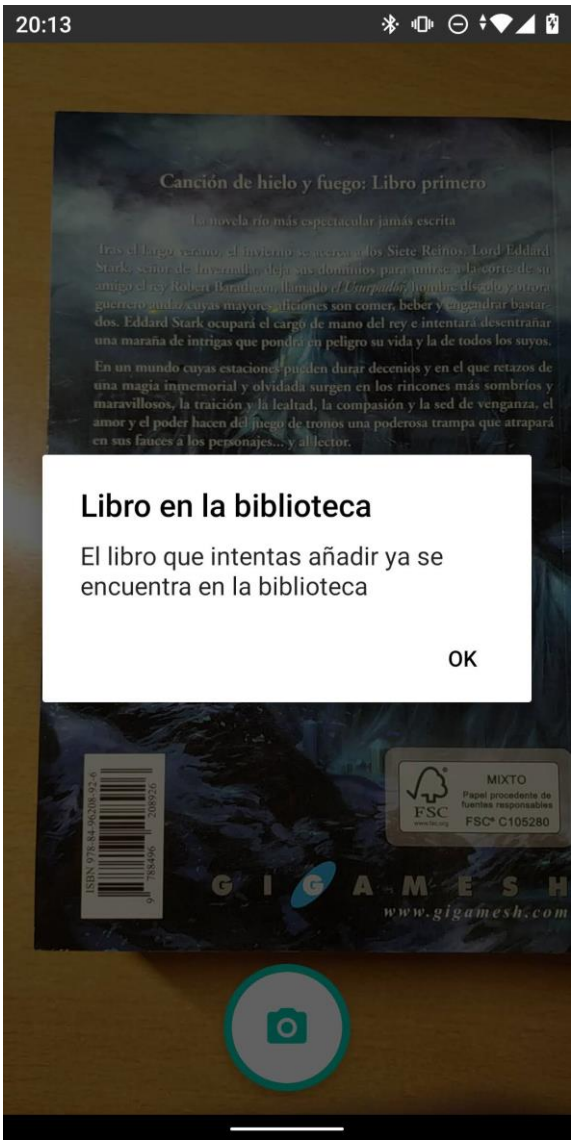

Ilustración 73 Mensaje cuando intentamos añadir un libro que ya teníamos registrado

5.3.2.2 Buscar datos por ISBN

Esta opción permite a un usuario buscar los datos de un libro introduciendo de forma manual el ISBN mediante el teclado, como se demuestra en la Ilustración 74, lo que podría ser útil si un usuario no puede utilizar la cámara de su dispositivo para escanear el código de barras del libro.

Tal y como sucedía en el caso anterior, al encontrar los datos del libro se mostrarían en una tarjeta como la que se mostró en la Ilustración 71 o, si no se llegara a encontrar los datos, se propondrían al usuario añadirlos manualmente como aparece en la Ilustración 72.

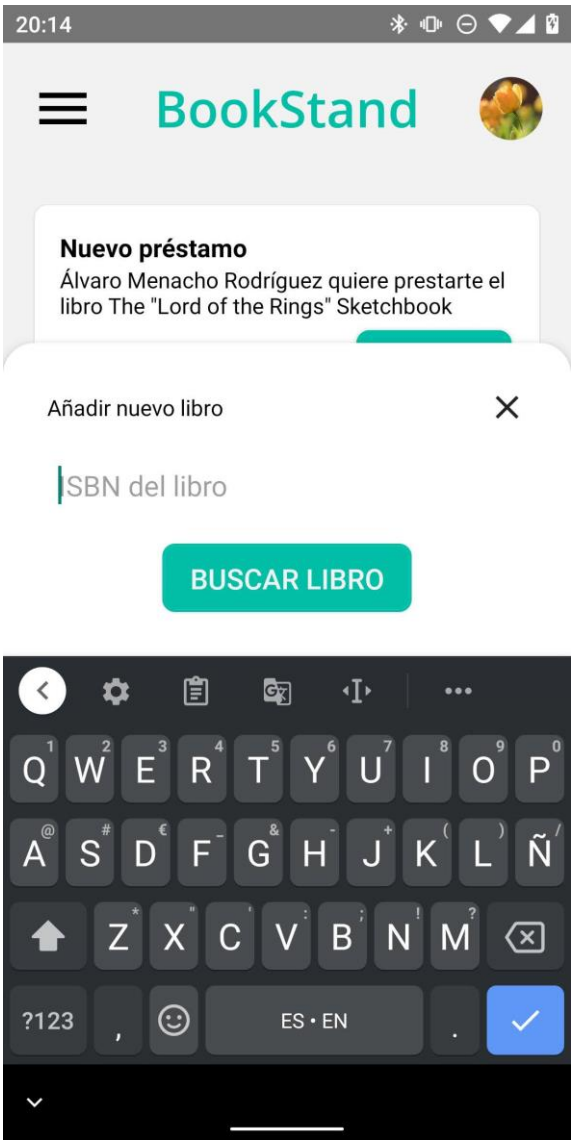

Ilustración 74 Buscar los datos de un libro a través de un ISBN añadido de forma manual

5.3.2.3 Añadir datos manualmente

Esta opción permite al usuario añadir los datos de un libro de forma manual, útil para aquellos ejemplares que no dispongan de ISBN pero que el usuario quiera incluir en su colección, Además, como ya se comentó, este método proporciona una forma alternativa para añadir los datos de aquellos libros que no tengan un registro en la base de datos de Open Library.

En la Ilustración 75 se observa que esta pantalla no es más que una tarjeta que representa los datos de un libro en blanco, que se aprovecha de la opción para editar los datos para que un usuario introduzca la información que crea necesaria.

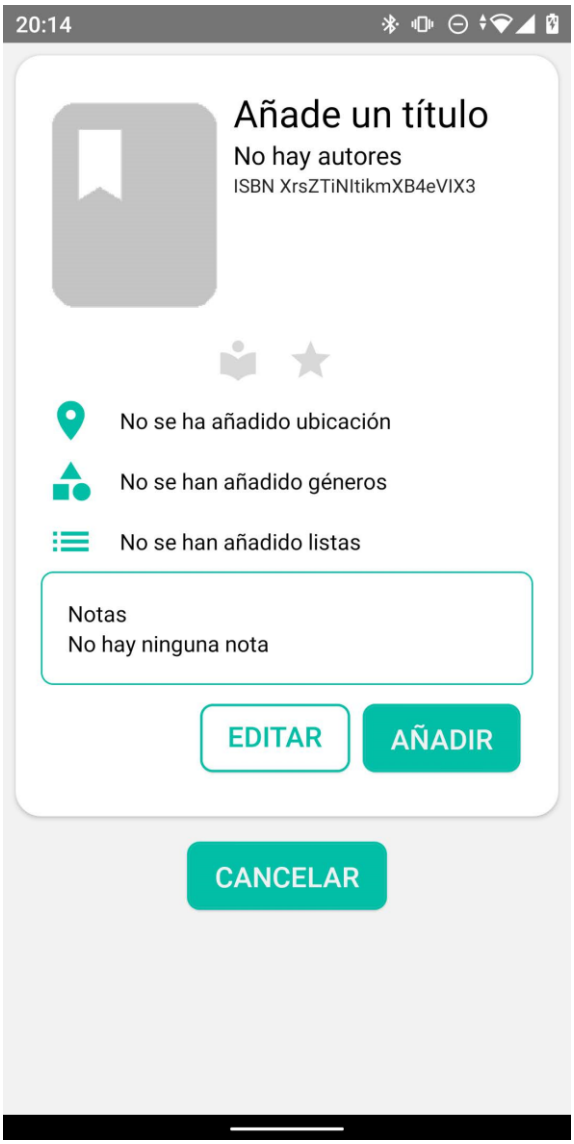

Ilustración 75 Inserción de datos de un libro de forma manual

En el caso en el que se añadan los datos de un libro de forma manual por no tener ISBN, el sistema asociará automáticamente a dicho libro un identificador aleatorio alfanumérico para poder indexarlo correctamente en el sistema.

5.3.2.4 Buscar por título

Por último, este método para añadir los datos de un libro se basa en introducir el título del libro que queremos añadir a nuestra biblioteca en un campo de texto, como se muestra en la Ilustración 76.

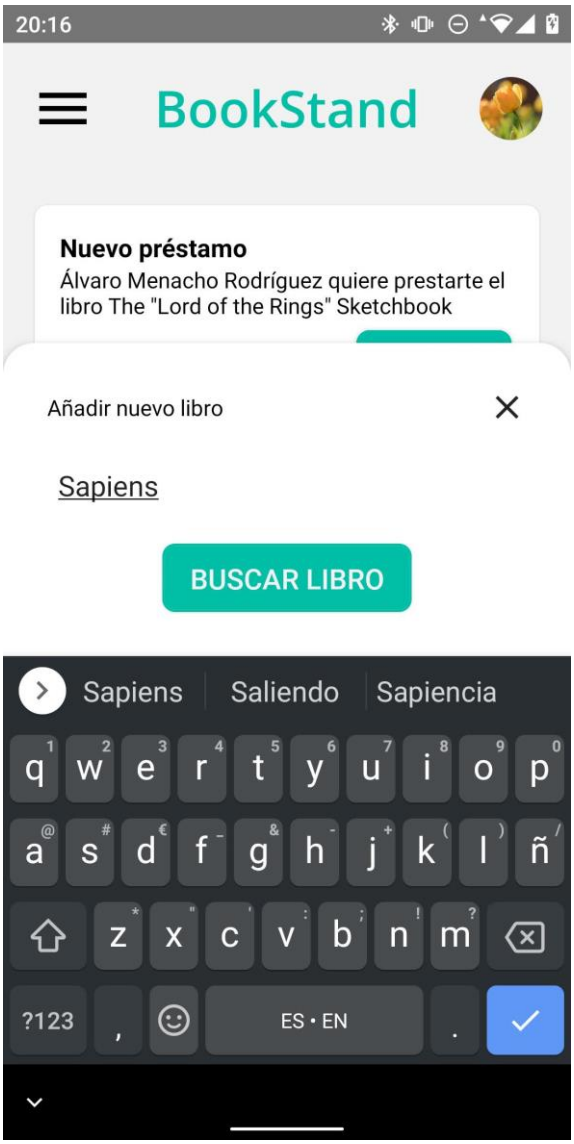

Ilustración 76 El usuario deberá introducir el título del libro que desee buscar

A partir del título se presentará al usuario una lista con los resultados más relevantes de la búsqueda, tal y como queda reflejado en la Ilustración 77. En este punto, el usuario podrá previsualizar cada uno de estos libros, como se muestra en la Ilustración 78, y añadir el que prefiera a su biblioteca.

Es necesario comentar que, en la Ilustración 77, aparecen varios registros con el mismo nombre y autores. Esto no es un error, sino que cada uno de los registros de ese libro corresponden a diferentes ediciones y, por tanto, tienen un ISBN diferente.

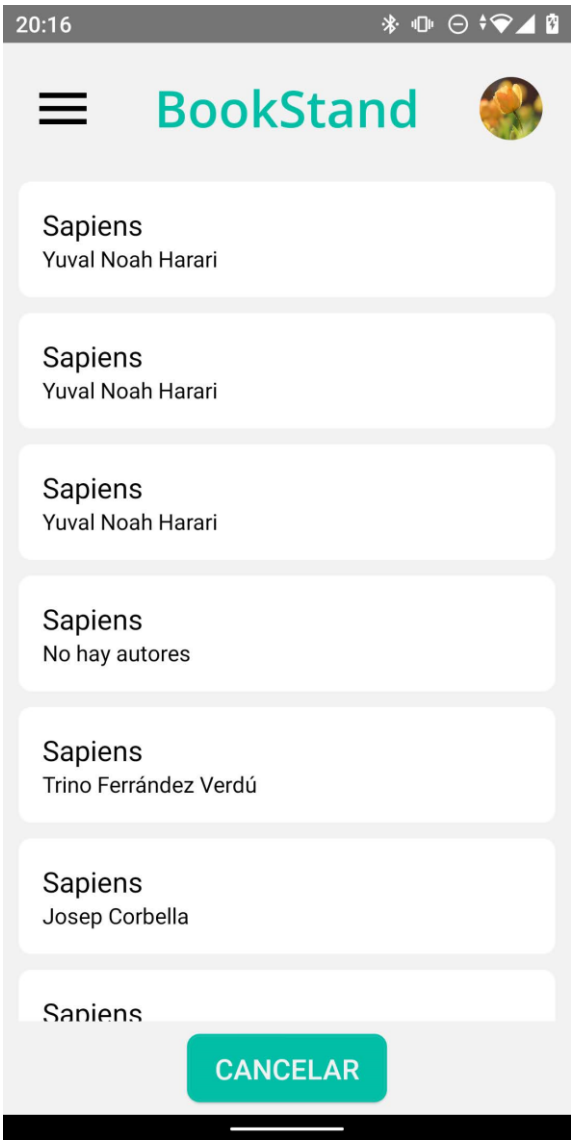

Ilustración 77 Lista de resultados al buscar un libro por título

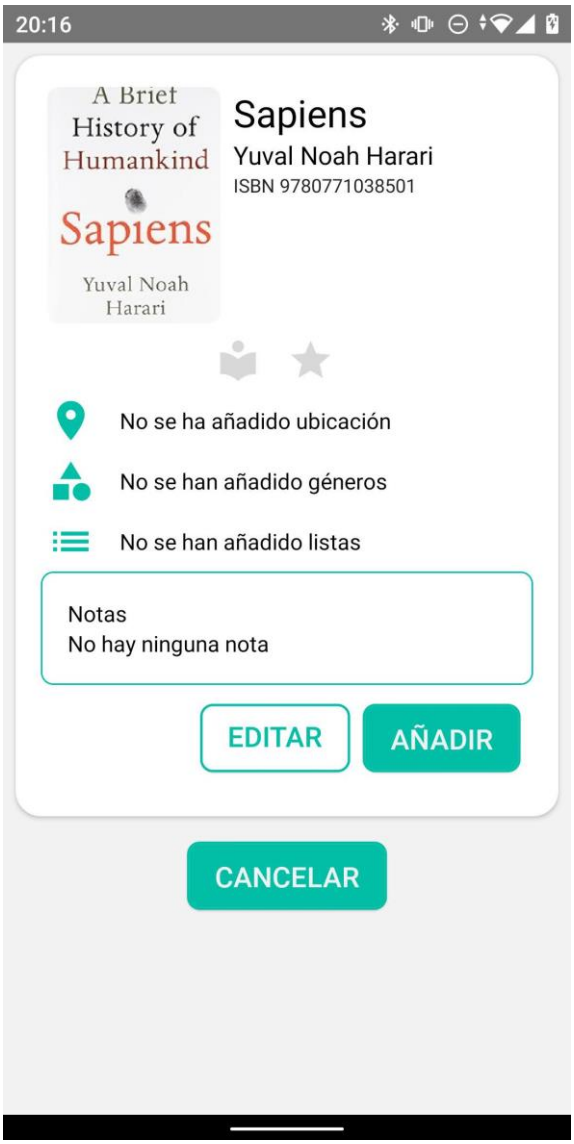

Ilustración 78 Datos de uno de los resultados de la búsqueda del libro por título

5.4 Pantalla Biblioteca

En este apartado se describen todas las operaciones que se pueden realizar en la pantalla Biblioteca, el corazón de BookStand. Esta sección de la aplicación nos permite ver los libros que se tienen agregados dentro de la aplicación, organizarlos, realizar búsquedas concretas y operar sobre libros en concreto.

En la Ilustración 79 podemos ver los principales elementos de la Biblioteca:

- Una barra de búsqueda que nos permite encontrar rápidamente libros
- Una barra de navegación que nos permite cambiar la vista de la biblioteca para ver la información más relevante de forma rápida. En concreto, nos permite ver todos los libros, las listas de los libros, nuestros libros favoritos y los libros que tenemos prestados.
- Un botón para añadir más libros a la biblioteca. Este botón nos permite realizar las mismas funciones que las descritas en el apartado 5.3.2.

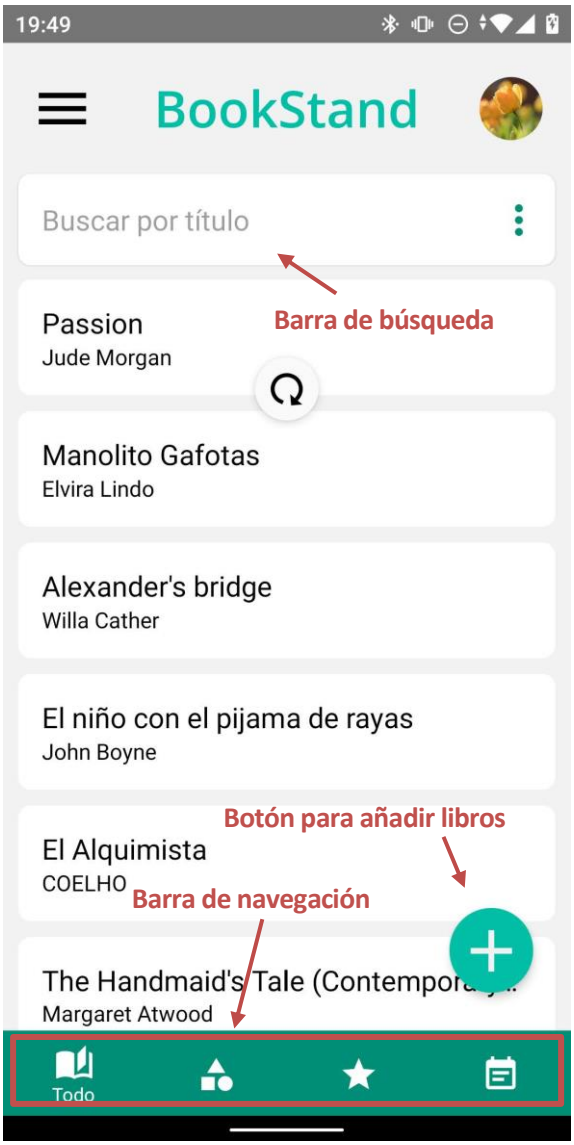

Ilustración 79 Pantalla de la Biblioteca

5.4.1 Ver detalles de un libro

Cuando presionamos cualquier libro de una de las listas de la aplicación navegamos hacia una pantalla con los detalles del libro, como la que se muestra en la Ilustración 80. Estos detalles, como se pueden ver en la Ilustración 80, son:

- Imagen de portada
- Título
- Autores
- ISBN (o, para libros personalizados, un identificador único)
- Botones que marcan si un libro ha sido leído o si lo hemos marcado como favorito
- La ubicación en la que el libro está guardado
- Los géneros a los que pertenece
- Las listas de usuario a las que pertenece
- Las notas que un usuario añada al libro

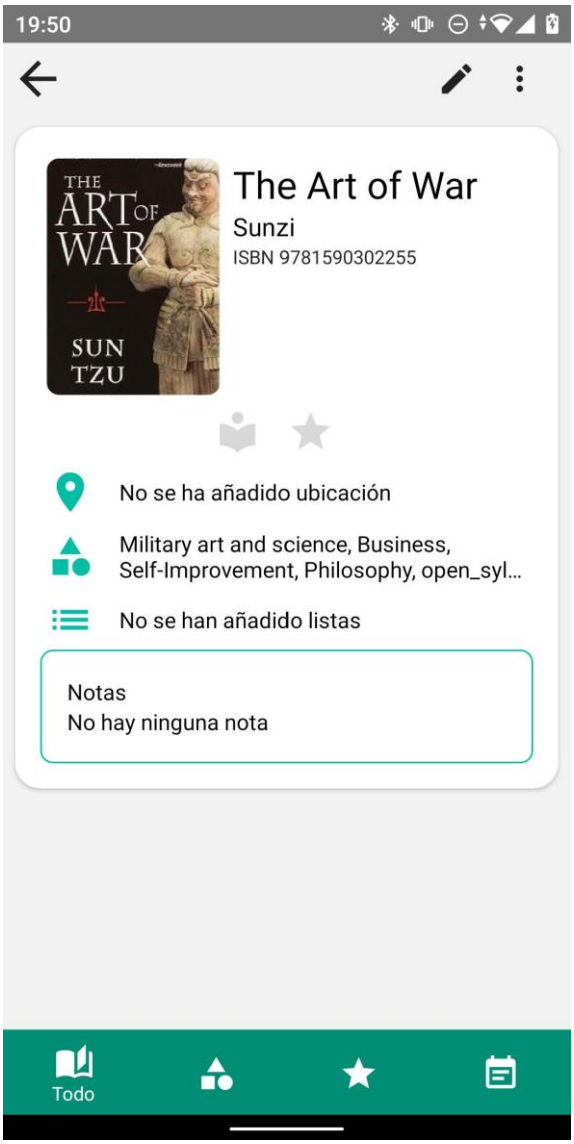

Ilustración 80 Vista de detalles de un libro

Excepto el ISBN y la foto de portada, todos estos datos puede modificarlos un usuario al presionar el botón con la forma de lápiz que aparece en la esquina superior derecha de la pantalla. Una vez hecho esto, entramos en el modo edición (Ilustración 81), y podemos cambiar los datos al tocarlos, como se muestra en la Ilustración 82 y en la Ilustración 83.

Ilustración 81 Modo edición de un libro

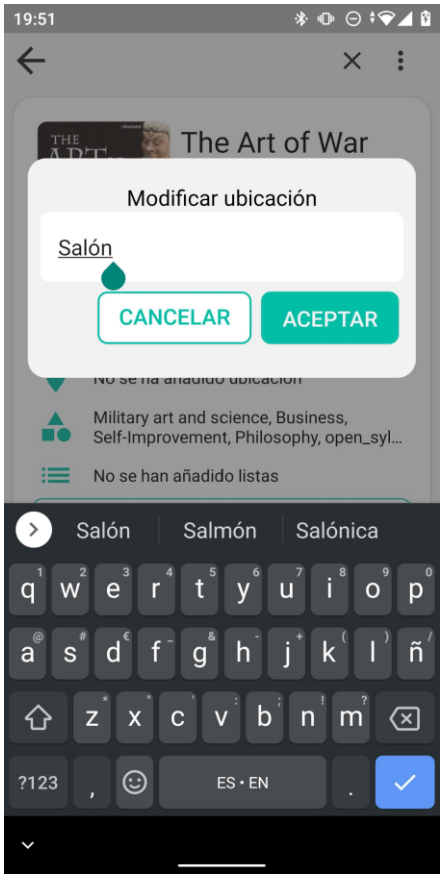

Ilustración 82 Edición del campo “ubicación”

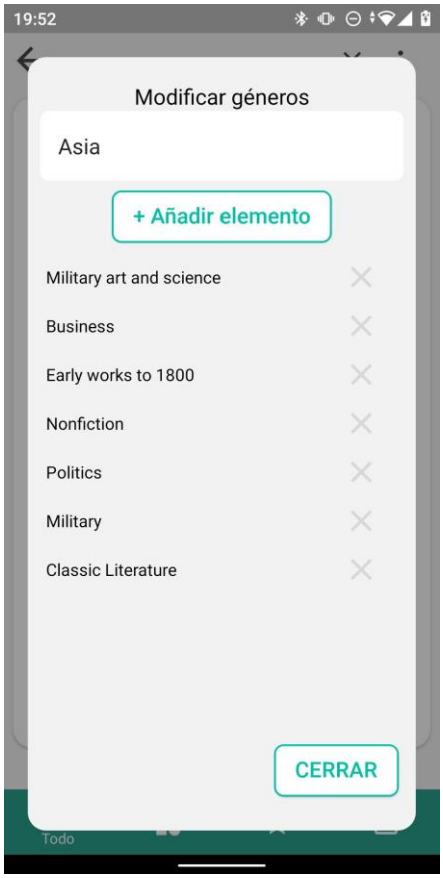

Ilustración 83 Edición del campo "géneros"

En la Ilustración 84, podemos observar el menú de opciones que aparece al pulsar el botón con tres puntos verticales en la esquina superior derecha de la pantallla. En este menú aparecen otras operaciones que podemos realizar sobre el libro: añadir un préstamo, ver el historial de préstamos y eliminar el libro de nuestra biblioteca. En la sección 5.5 se detallarán las acciones relacionadas con los préstamos.

Cabe mencionar que solo seremos capaces de eliminar un libro de nuestra biblioteca siempre y cuando dicho libro no esté siendo prestado en ese momento.

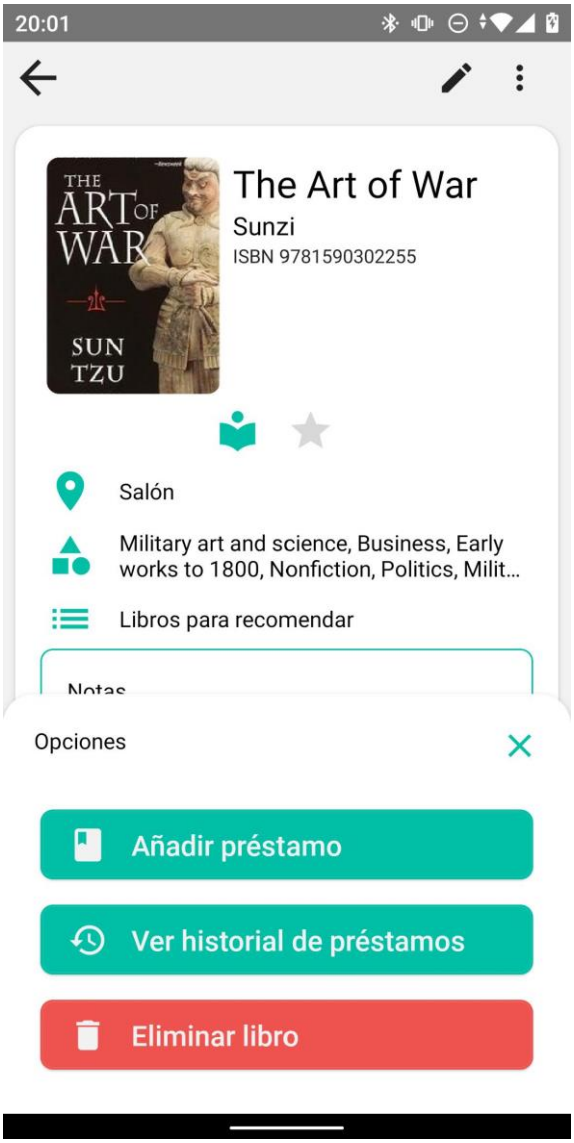

Ilustración 84 Menú de opciones de un libro

5.4.2 Buscar libros concretos

Como comentábamos hace unas líneas, la barra de búsqueda nos permite buscar libros concretos a partir de diferentes criterios. En las imágenes siguientes se muestran las diferentes configuraciones de búsqueda:

Ilustración 85 Ajustes de la búsqueda en la biblioteca

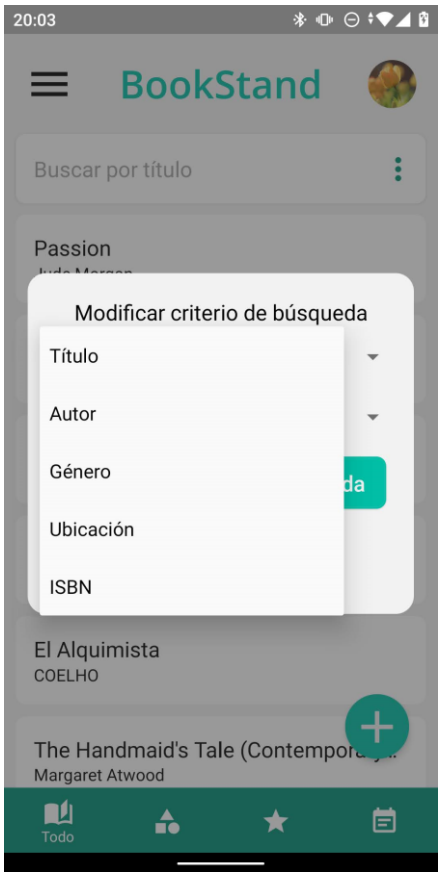

Ilustración 86 Diferentes criterios de búsqueda

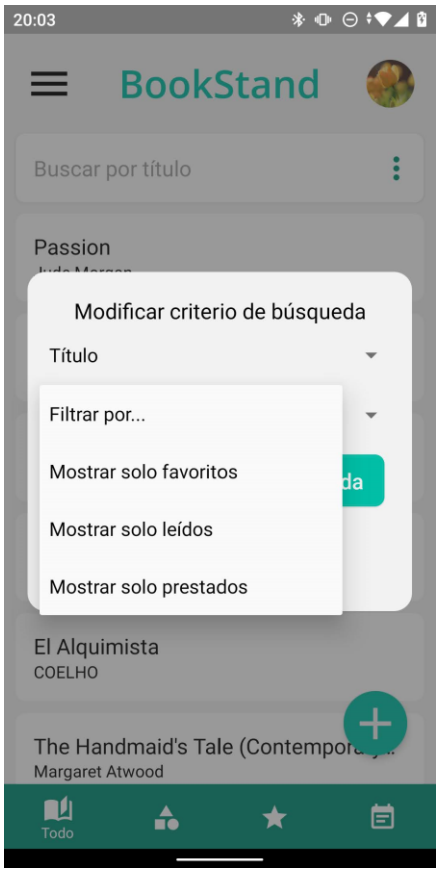

Ilustración 87 Diferentes criterios de filtrado

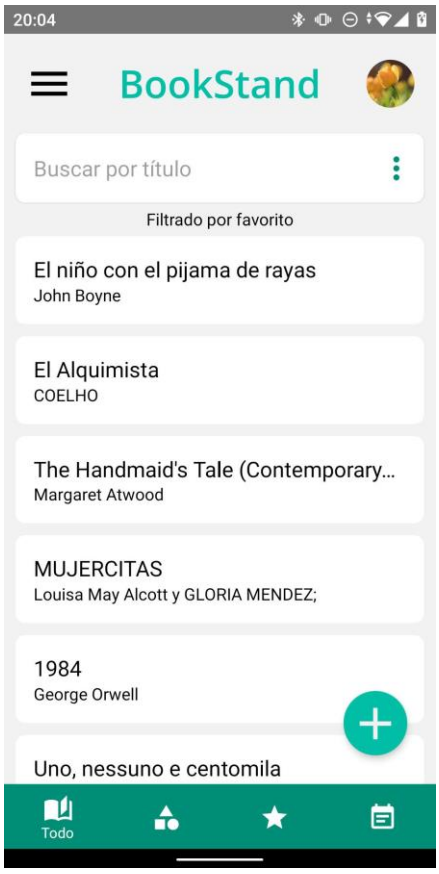

Ilustración 88 Biblioteca filtrada por favoritos

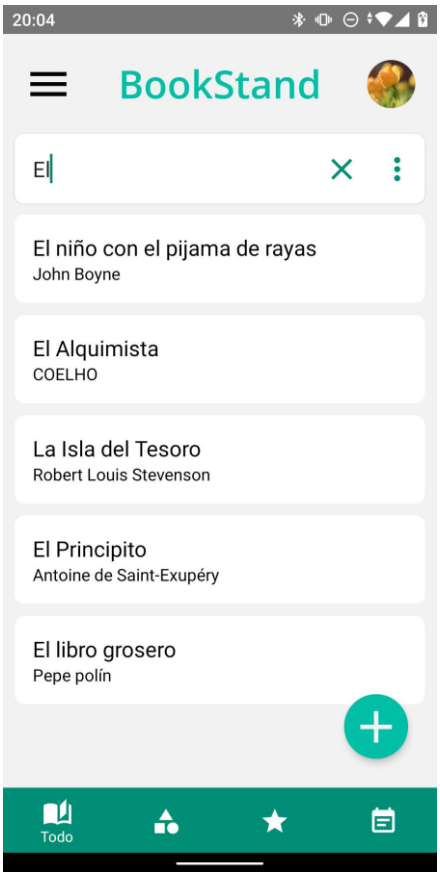

Ilustración 89 Búsqueda de libros por título

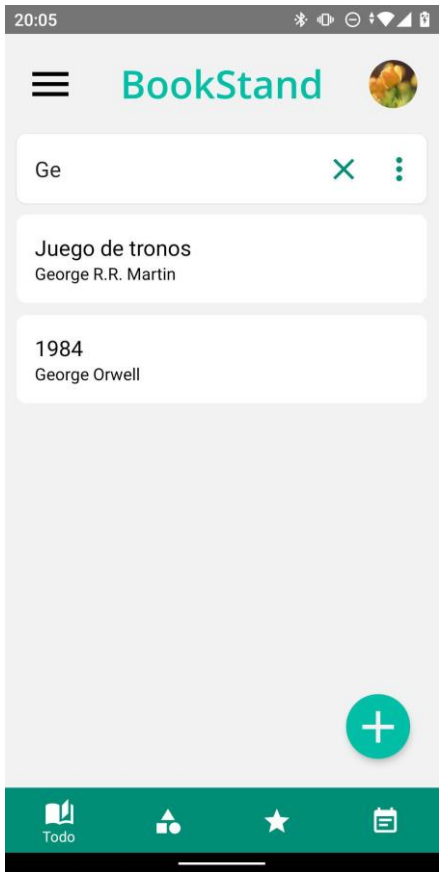

Ilustración 90 Búsqueda de libros por autor

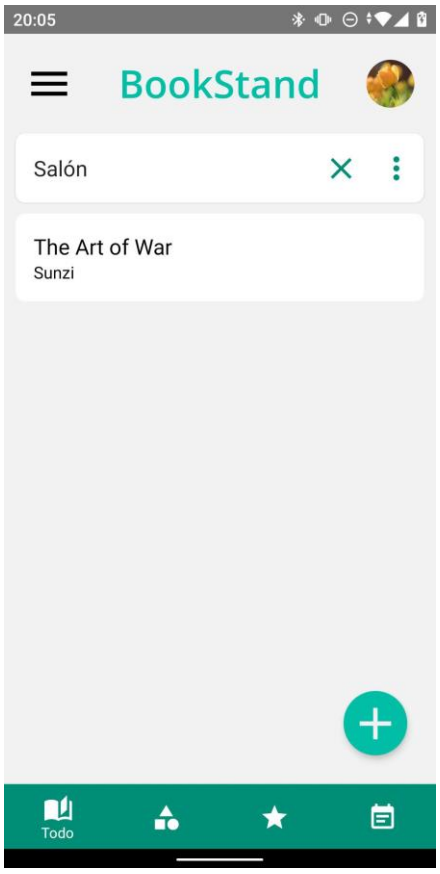

Ilustración 91 Búsqueda de libros por ubicación

5.4.3 Ver listas de libros

Cuando editamos los datos de un libro y añadimos una lista, se crea un acceso directo en la sección "Listas" de la biblioteca, a la que podemos acceder mediante la barra de navegación, tal y como se muestra en la Ilustración 92.

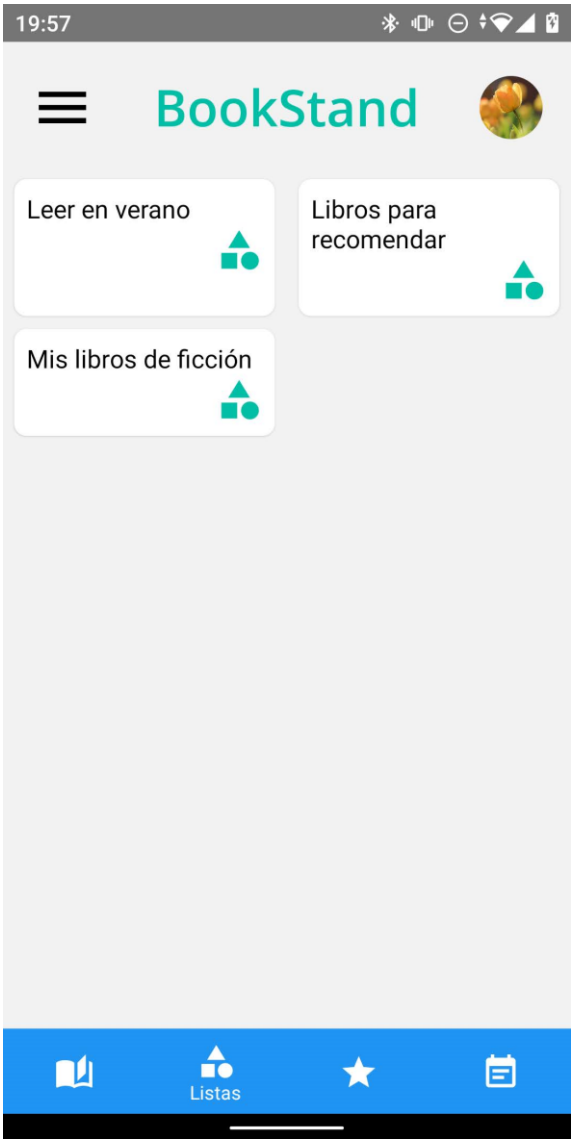

Ilustración 92 Listas de libros en la Biblioteca

Al pulsar sobre cualquiera de las listas creadas, se muestra una lista con todos los libros que pertenecen a la misma, como la que aparece en la Ilustración 93. Gracias a esta funcionalidad, se permite al usuario organizar sus libros en colecciones personalizadas y poder visualizarlas de una forma rápida.

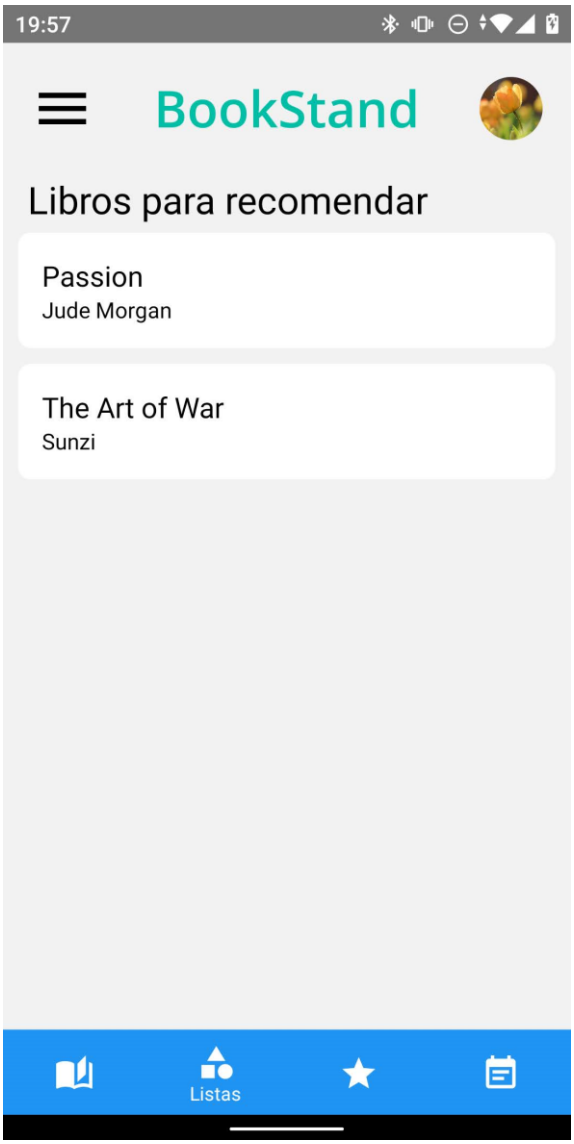

Ilustración 93 Lista "Libros para recomendar"

5.4.4 Ver los libros favoritos

Otra de las vistas rápidas de los libros de la Biblioteca es la vista de favoritos, que, como su nombre indica, muestra una lista con todos los libros que hemos marcado como favoritos, como se muestra en la Ilustración 94. Podemos acceder a esta vista rápida a través de la tercera pestaña de la barra de navegación.

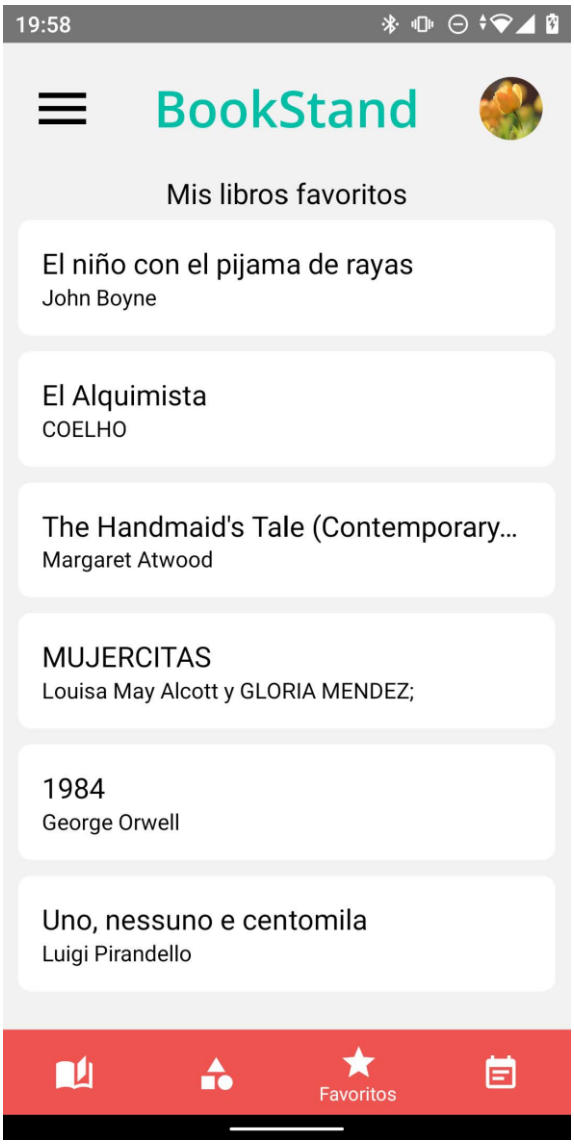

Ilustración 94 Vista de libros favoritos

5.4.5 Ver los libros prestados

Finalmente, la última de las vistas de la Biblioteca presenta al usuario los libros que tienen un préstamo en curso con su fecha de devolución, de tenerla establecida. En la Ilustración 95 podemos ver esta pantalla.

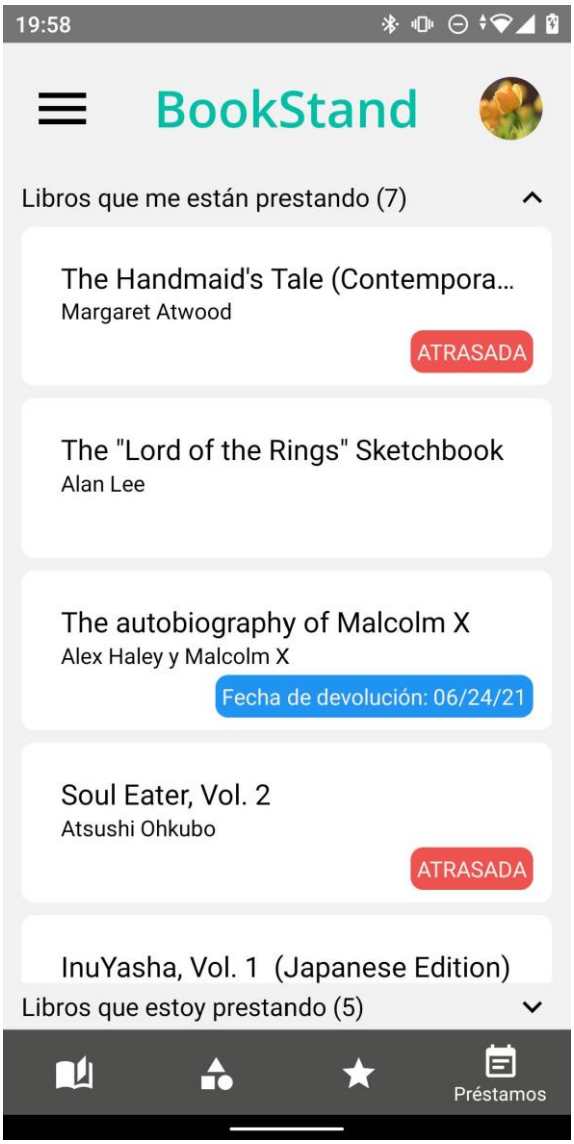

Ilustración 95 Vista de los libros con préstamos

5.5 Gestión de los préstamos

Tal como introdujimos en el apartado 5.4.1, en las siguientes líneas comentaremos una de las funcionalidades más importantes de la aplicación: la gestión de préstamos. En esta sección describiremos cómo crear un préstamo de un libro, modificarlo o devolverlo y ver el historial de préstamos de un libro.

5.5.1 Realizar préstamos a otros usuarios

Comenzaremos por crear un préstamo. Es importante comentar que la persona que inicia el préstamo debe ser el dueño del libro y que dicho libro no esté siendo actualmente prestado a algún usuario. En el caso de que no se cumpla alguno de estos requisitos, aparecerá un mensaje en pantalla informando al usuario.

Una vez comentado esto, para crear un préstamo debemos presionar sobre el botón “Añadir préstamo” que aparece en el menú de opciones que describimos en la Ilustración 84. Tras esto, aparecerá en pantalla una tarjeta en la que se nos pedirá introducir los datos relativos al préstamo, como el contacto a quien se lo vamos a prestar, la fecha en la que realizamos el préstamo y, de forma opcional, una fecha provisional para la devolución del libro. En las siguientes ilustraciones se detallan estos pasos.

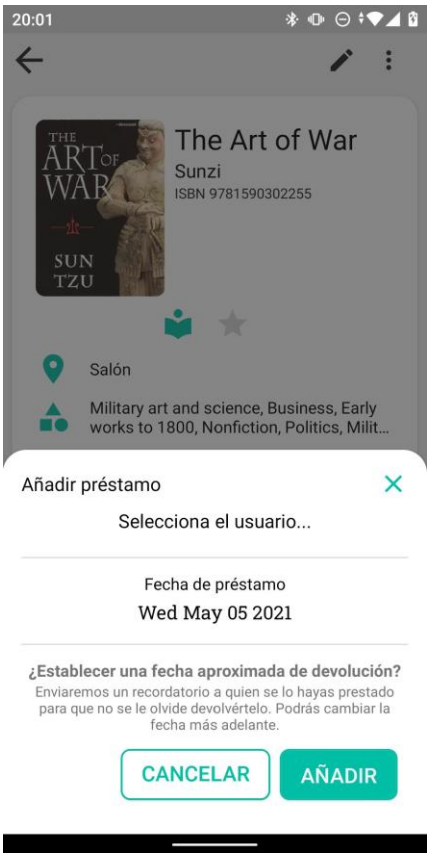

Ilustración 96 Tarjeta para añadir los datos del préstamo

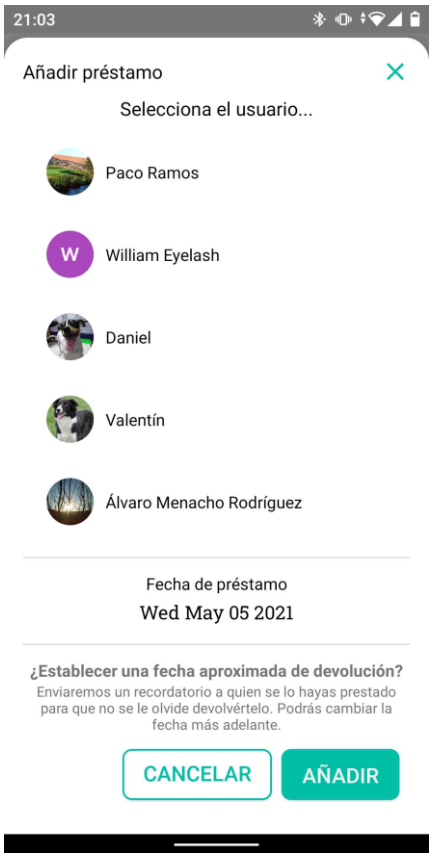

Ilustración 97 Selección del contacto a quien se presta el libro

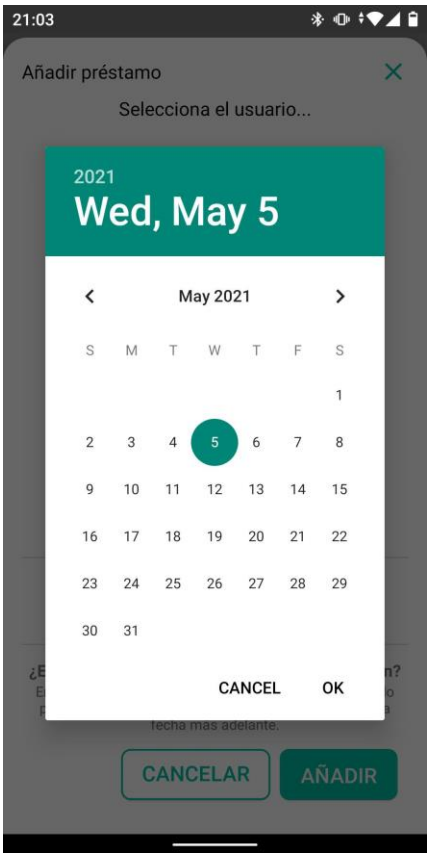

Ilustración 98 Introducción de la fecha de préstamo

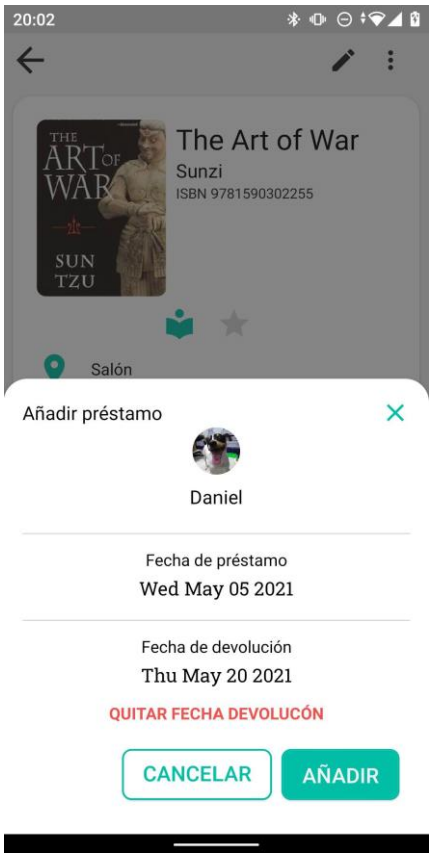

Ilustración 99 Vista preliminar del préstamo

Una vez hayamos introducido los datos necesarios, al pulsar en el botón "Añadir" que aparece en la Ilustración 99, se enviará una notificación al contacto seleccionado para que confirme la creación del préstamo.

Mientras tanto, al pulsar sobre el botón "Historial de préstamos" del menú de opciones del libro podremos ver que ha quedado registrado el préstamo con estado pendiente de confirmación, como se muestra en la Ilustración 100.

Además, si observamos la Ilustración 101, comprobamos que aparece una marca en la pantalla de los detalles del libro que nos informa de que actualmente el libro forma parte de un préstamo.

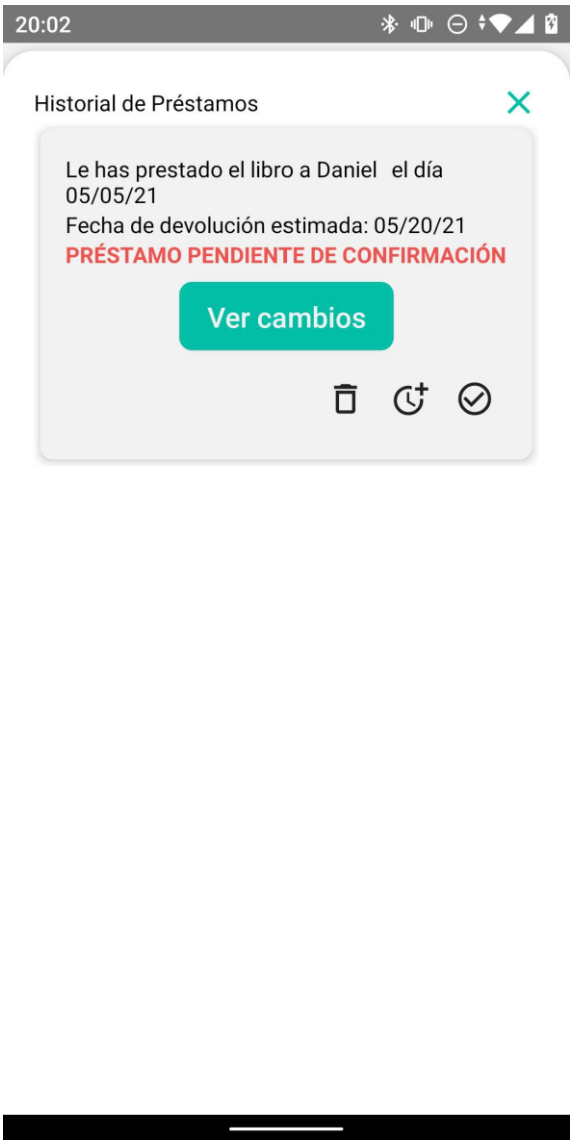

Ilustración 100 Historial de préstamos de un libro

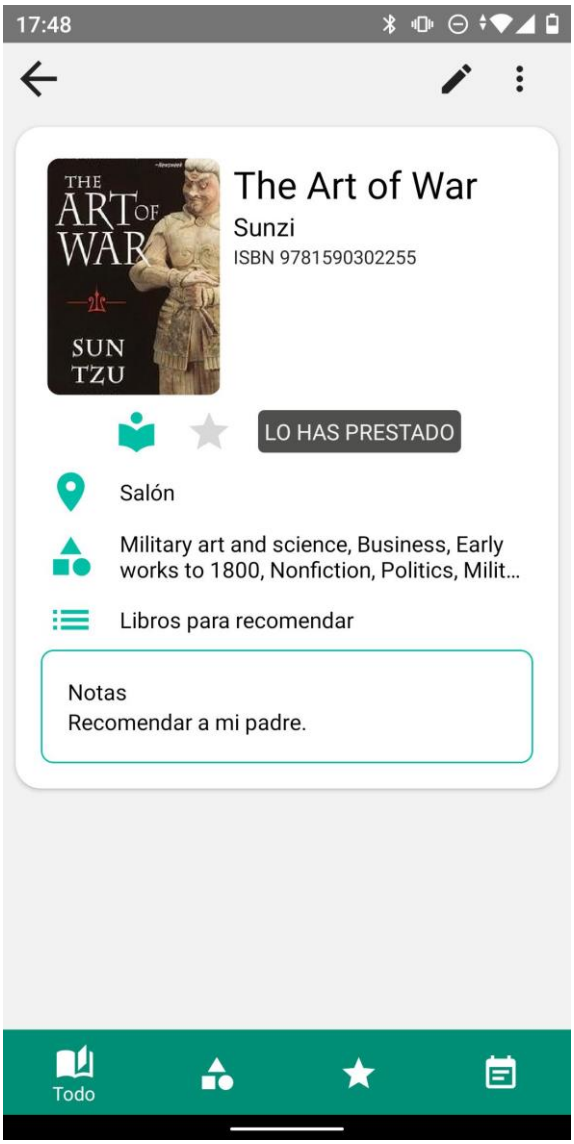

Ilustración 101 Marca que informa de que un libro está siendo prestado

5.5.2 Modificar o devolver préstamos

En este apartado describiremos cómo hacer modificaciones o devoluciones de un préstamo. Para ello, necesitamos navegar hasta el historial de préstamos de un libro que esté siendo prestado. Como comentábamos unas líneas más arriba, a este historial se accedía a través del menú de opciones de los detalles de un libro.

En dicho historial aparecen tarjetas en las que se aparecen los datos más relevantes de un préstamo además de diferentes botones de acción, tal y como se muestra en la Ilustración 102:

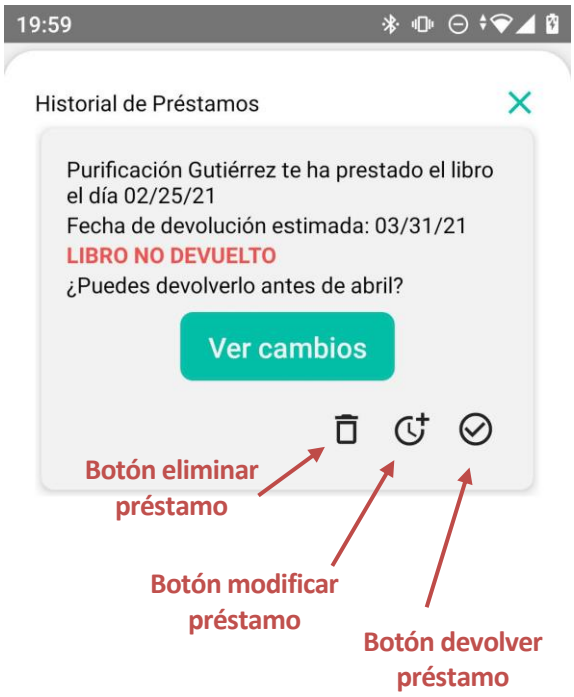

Ilustración 102 Historial de préstamos (préstamo aceptado)

Como aparece reflejado en la figura anterior, las opciones que se pueden realizar con un préstamo son: eliminar el registro del préstamo, modificarlo o marcar el préstamo como libro devuelto.

El sistema solo dejará eliminar el registro de un préstamo si el libro se encuentra devuelto, de forma que los prestamos entre los usuarios sean consistentes y no se produzcan errores o confusiones entre los mismos.

Si se desea modificar un préstamo, aparecerá una tarjeta en la que el usuario podrá introducir una fecha de devolución estimada o, si se introdujo en el momento de la creación del préstamo, cambiarla. Adicionalmente, se permite que el usuario introduzca un pequeño mensaje junto a la modificación, tal y como se muestra en la Ilustración 103.

A la hora de marcar un libro como devuelto aparecerá una tarjeta similar a la descrita anteriormente en la que se puede indicar tanto la fecha de devolución como añadir un pequeño mensaje, como se refleja en la Ilustración 104.

Ilustración 103 Modificación de un préstamo

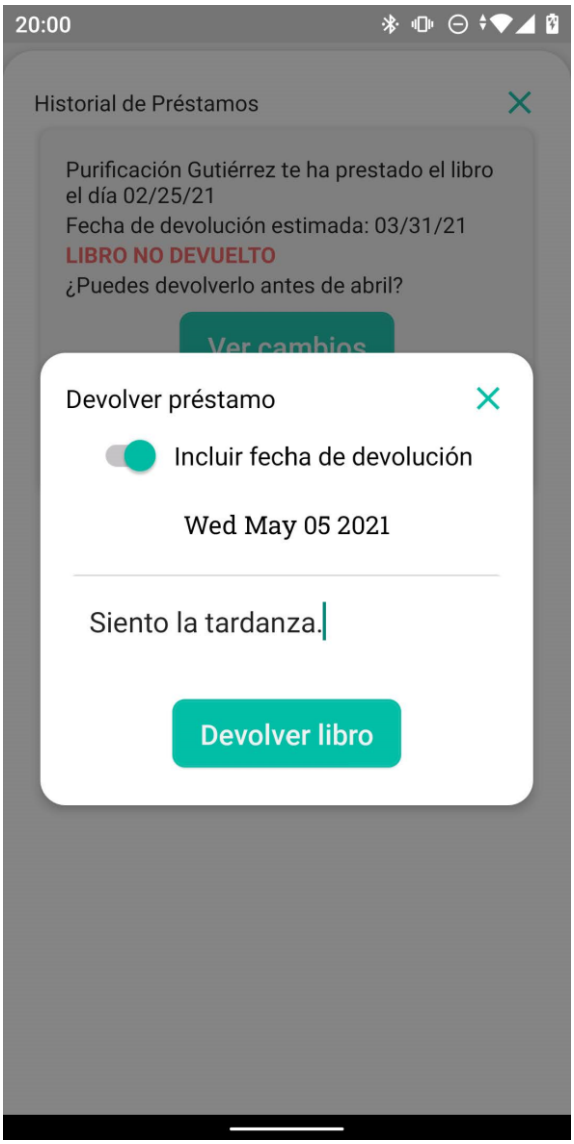

Ilustración 104 Devolución de un préstamo

Además, si presionamos en el botón “Ver cambios” aparecerá una lista con todos los cambios que se han realizado sobre el préstamo desde el momento de su creación hasta su devolución, de la forma en la que se muestra en la Ilustración 105.

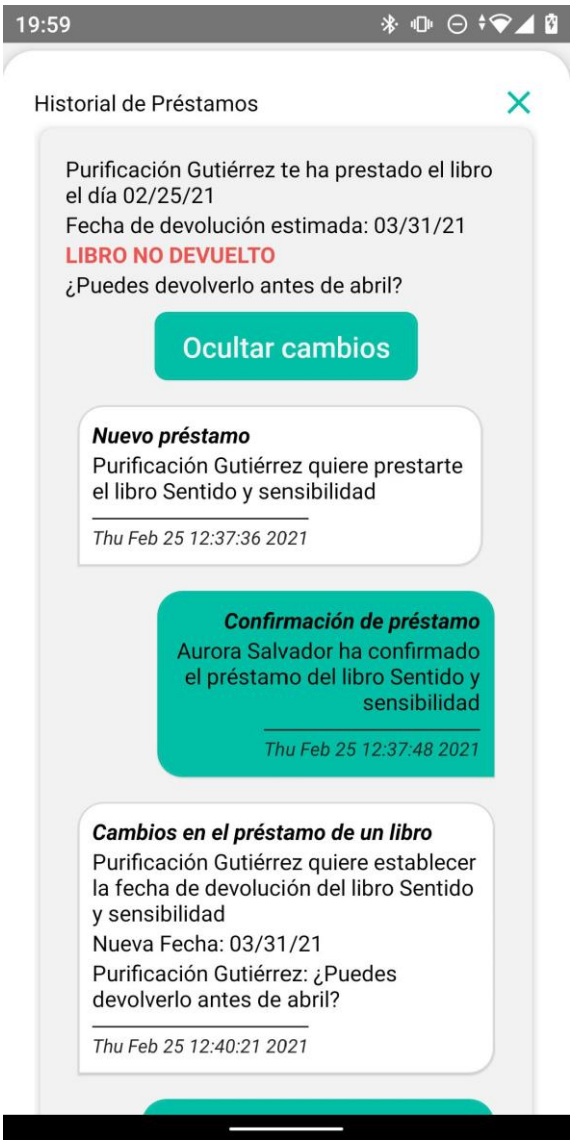

Ilustración 105 Lista de cambios de un préstamo

5.5.3 Recordatorios y notificaciones

Si en los apartados anteriores hablábamos de cómo un usuario puede crear un préstamo, proponer cambios en el mismo o devolver libros, en este apartado describiremos cómo hacer estos cambios efectivos o, por el contrario, denegarlos.

Como ya se introdujo en la sección 5.3.1, esto se hace aceptando o denegando las notificaciones que le llegan a un usuario en su *feed* de la pantalla principal, como la notificación "Nuevo préstamo" que se muestra en la Ilustración 106.

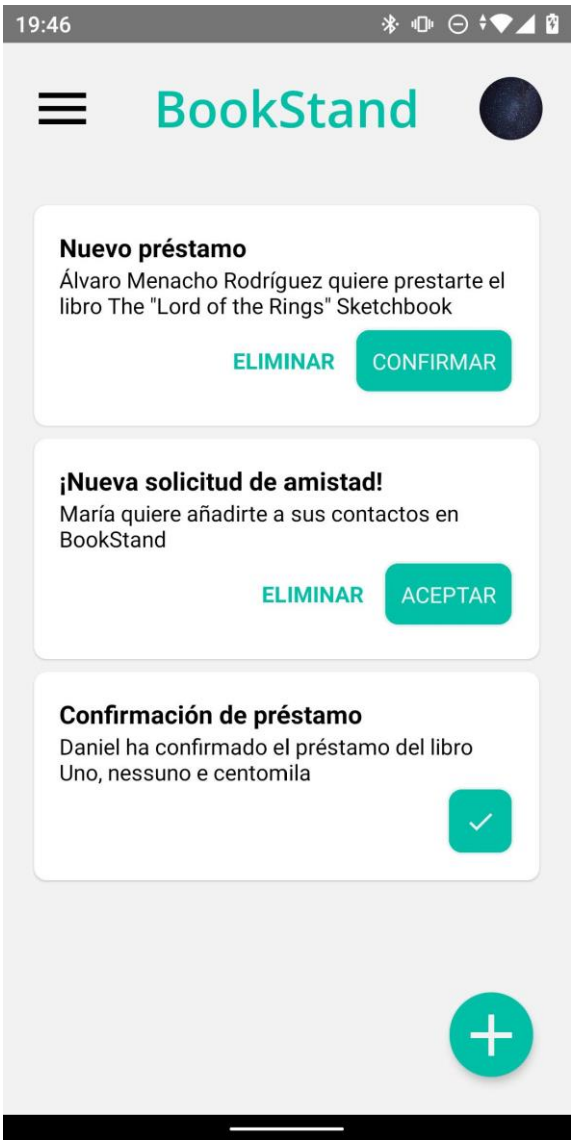

Ilustración 106 Notificaciones de los préstamos en el *feed* de notificaciones

Estas notificaciones no solo se muestran dentro de la aplicación, sino que también llegan como notificaciones *push* en segundo plano como la que aparecen en la Ilustración 107. En la Ilustración 108 aparece la misma notificación dentro de la aplicación.

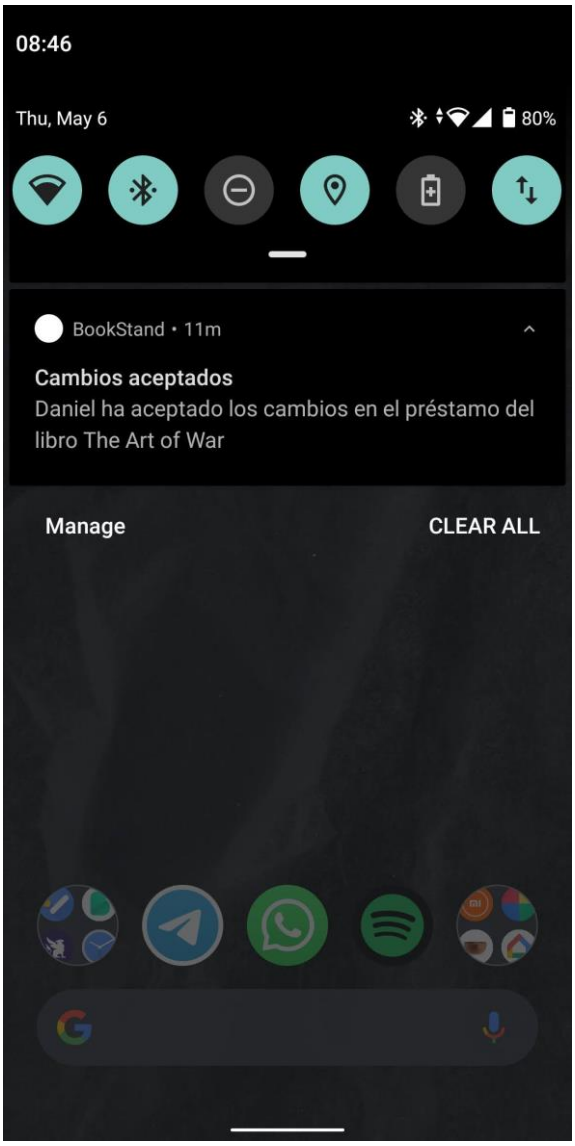

Ilustración 107 Notificación push de “cambios confirmados”

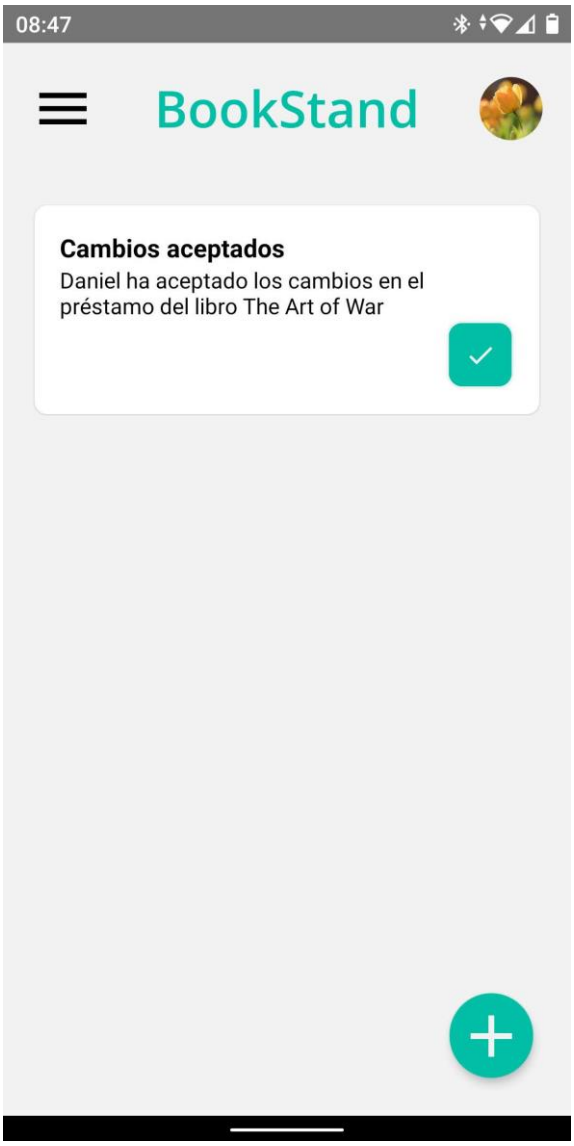

Ilustración 108 Notificación de cambios confirmados dentro de la app

Para finalizar con las notificaciones, BookStand cuenta con una tarea programada que se ejecuta cada vez que se inicia la aplicación y que comprueba el estado de los préstamos. Tras esto, el sistema lanza notificaciones push de recordatorio para informar de los préstamos atrasados o aquellos cuya fecha de devolución es próxima. En la Ilustración 109 se pueden ver este tipo de notificaciones:

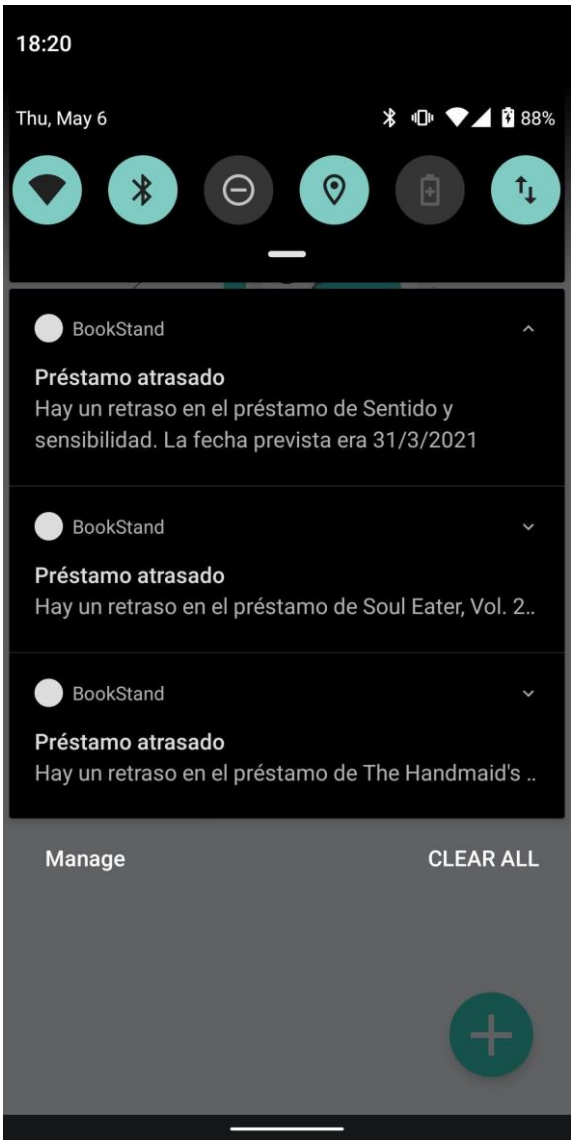

Ilustración 109 Notificaciones push de recordatorio

5.6 Pantalla Mis Contactos

En esta sección se muestran las pantallas relacionadas con los contactos en BookStand. Como se comentó en el apartado 5.5.1, a la hora de realizar un préstamo es necesario seleccionar el contacto a quien queremos prestar el libro.

Los contactos son aquellos usuarios de BookStand con los que podemos realizar o recibir préstamos de libros. Aquellos usuarios que sean nuestros contactos aparecerán listados seguidos de un botón para eliminarlos, si lo quisiéramos, tal y como se muestra en la Ilustración 110:

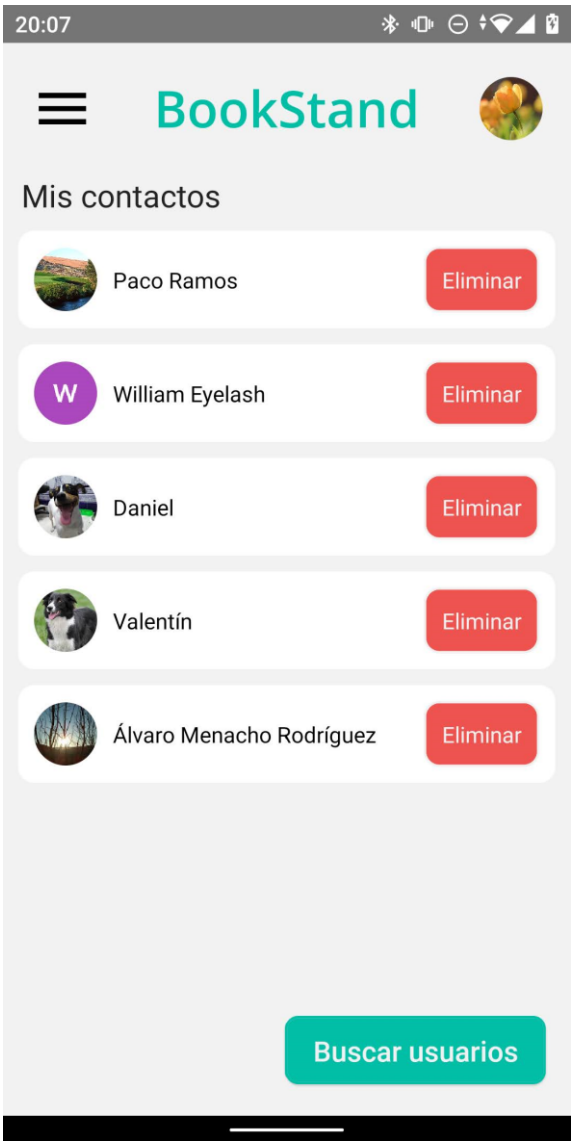

Ilustración 110 Pantalla "Mis Contactos" - Lista de contactos

Para agregar un usuario a nuestros contactos, simplemente pulsamos el botón “Buscar usuarios” que aparece en la esquina inferior derecha de la pantalla. Tras esto, escribimos el nombre del contacto que queramos buscar.

Se mostrará una lista con los usuarios más relevantes a partir del nombre introducido. Los usuarios que aún no sean nuestros contactos aparecerán con un botón “Añadir”, como se muestra en la Ilustración 111. Al pulsar este botón, se le enviará una solicitud de amistad a dicho contacto y, mientras este no la acepte, aparecerá un texto indicando que estamos pendiente de su confirmación como el que aparece en la Ilustración 112.

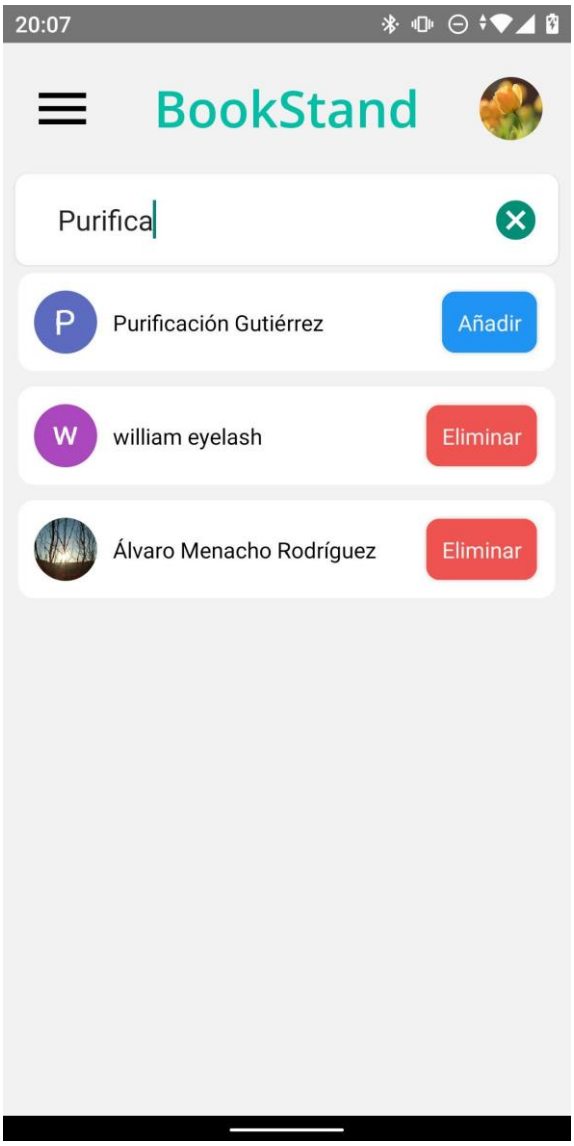

Ilustración 111 Buscamos un usuario en BookStand

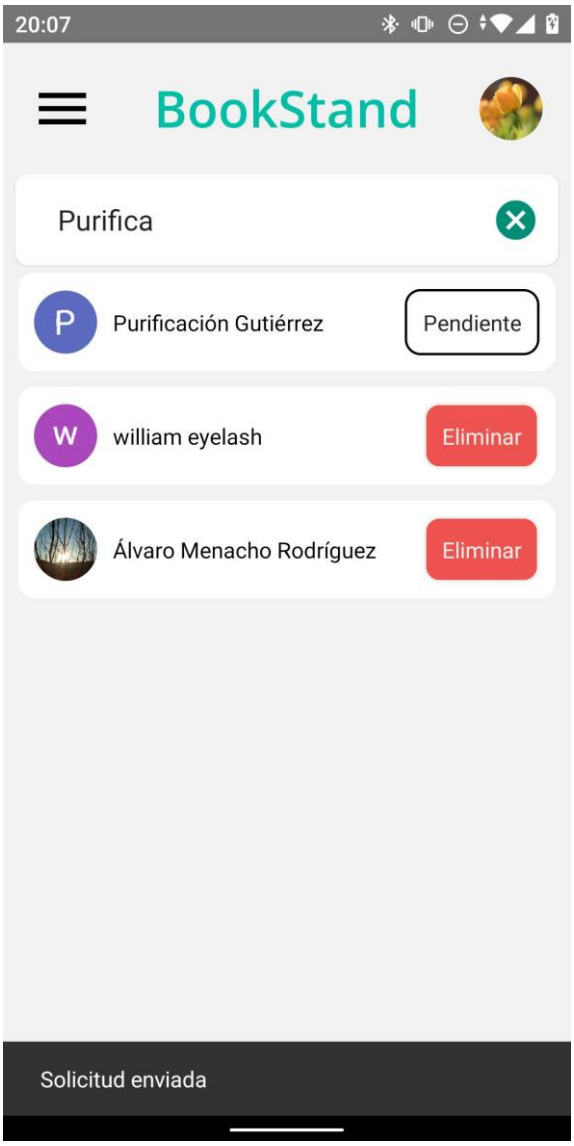

Ilustración 112 Envío de petición de amistad a un usuario

5.7 Pantalla Perfil

En esta pantalla se muestra la información básica que BookStand almacena sobre el usuario y le ofrece diversas operaciones como modificar sus datos de perfil, cerrar sesión o eliminar todos los datos de su cuenta. En la Ilustración 113 podemos ver la interfaz de esta pantalla.

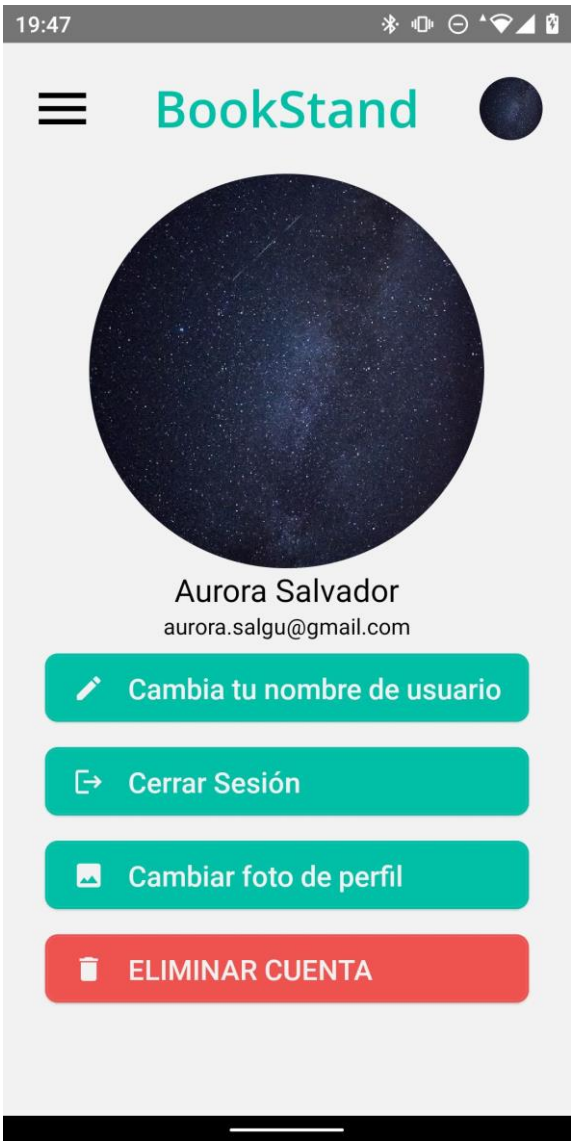

Ilustración 113 Pantalla Perfil de BookStand

BookStand ofrece al usuario modificar tanto su nombre dentro de la aplicación como su foto de perfil, ya que son los datos públicos que se mostrarán a todos los usuarios de la aplicación.

En la Ilustración 114 podemos ver cómo cambiar el nombre de usuario:

Ilustración 114 Modificar el nombre del usuario

Para cambiar la foto de perfil, presionamos sobre el botón “Cambiar foto de perfil”, lo que lanzará la aplicación de galería del dispositivo para que el usuario pueda seleccionar la imagen que desee, como se aprecia en la Ilustración 115. Una vez hecho esto, los cambios se aplicarán y quedará el perfil actualizado de forma similar a como se muestra en la Ilustración 116.

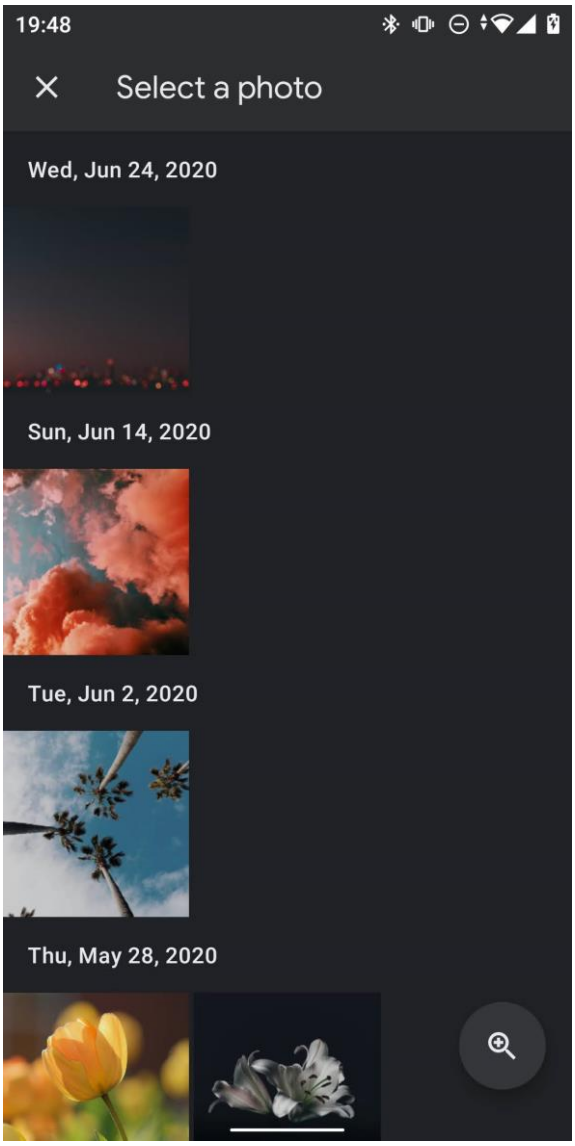

Ilustración 115 Selección de la foto de perfil desde la galería de fotos del dispositivo

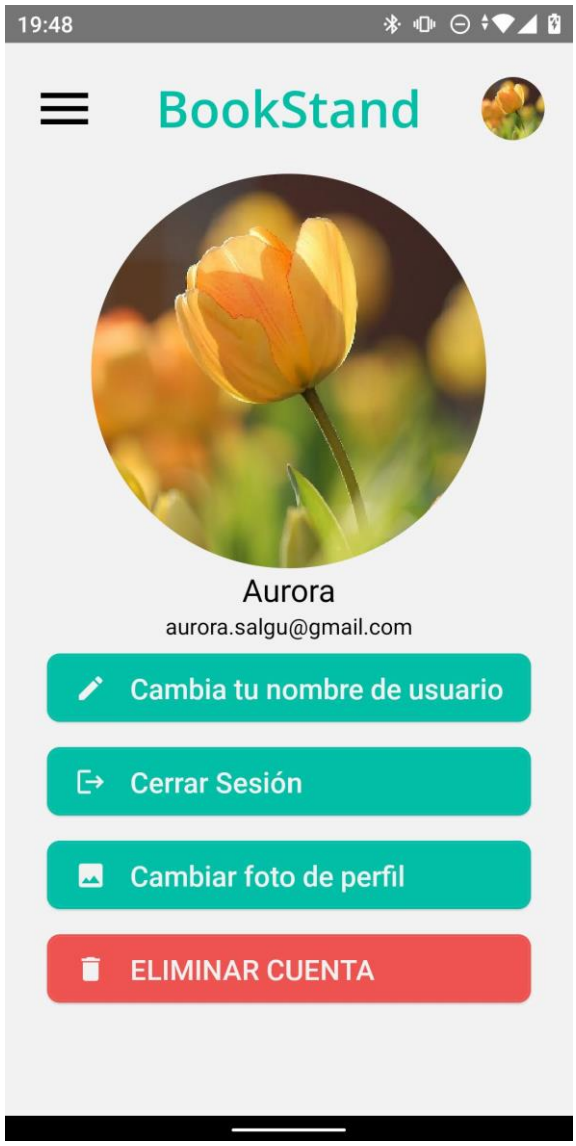

Ilustración 116 Perfil actualizado tras cambiar el nombre y la foto

Finalmente, la opción eliminar cuenta borra todos los datos del usuario del sistema, incluyendo los datos del servidor de autenticación y todos los libros que haya incluido en su biblioteca personal. Por ser una acción irreversible, se informa al usuario antes de proceder, tal y como podemos ver en la Ilustración 117.

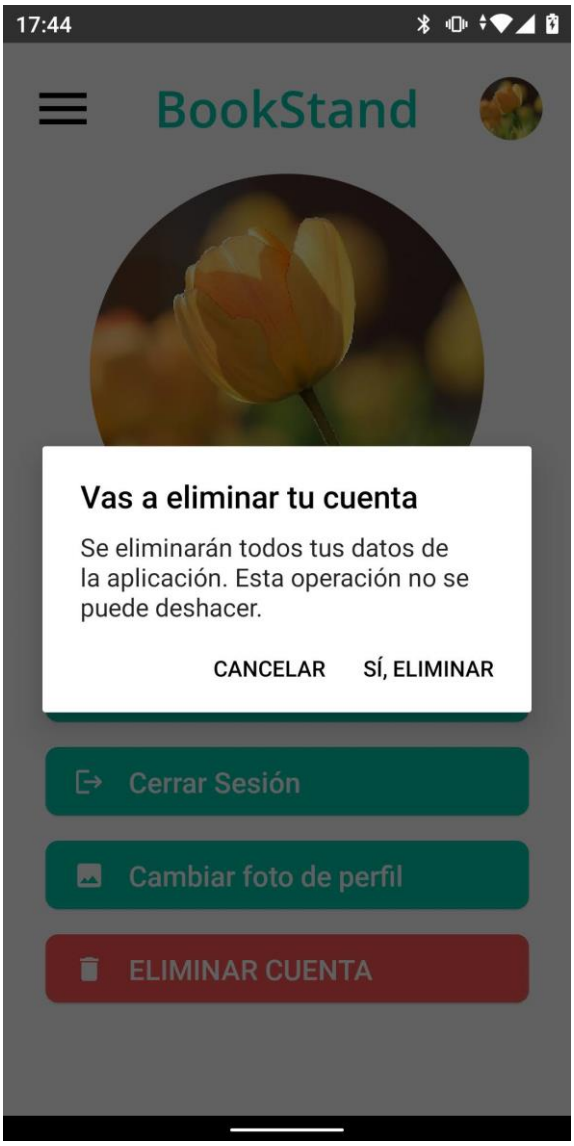

Ilustración 117 Eliminar la cuenta de BookStand

5.8 Pantalla Acerca de BookStand

En esta pantalla se muestra al usuario información relativa a la aplicación, como la versión de la app, o enlaces a la web de Open Library, al código fuente usado o a un formulario de Google donde dejar comentarios acerca de la aplicación. La interfaz se muestra en la Ilustración 118:

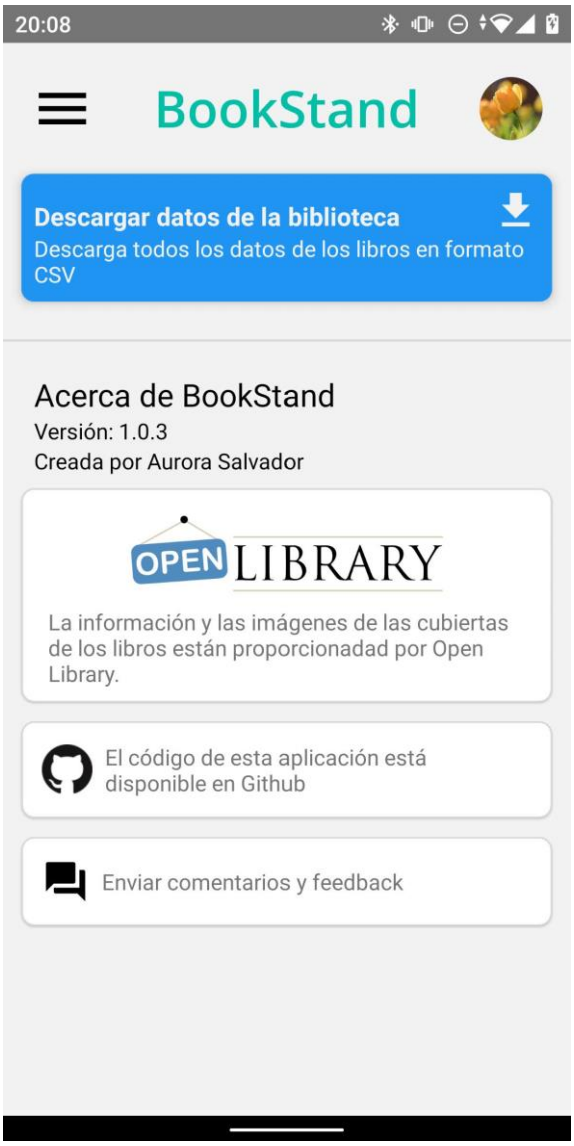

Ilustración 118 Pantalla "Acerca de BookStand"

Además, se ofrece al usuario la posibilidad de exportar los datos de los libros en formato CSV. Para ello, el usuario deberá otorgar primero los permisos de almacenamiento, como aparece en la Ilustración 119. Una vez se hayan aceptado los permisos, se informará al usuario de que los datos se han descargado correctamente y se encuentran en la carpeta “Descargas” del dispositivo, como vemos en la Ilustración 120 e Ilustración 121.

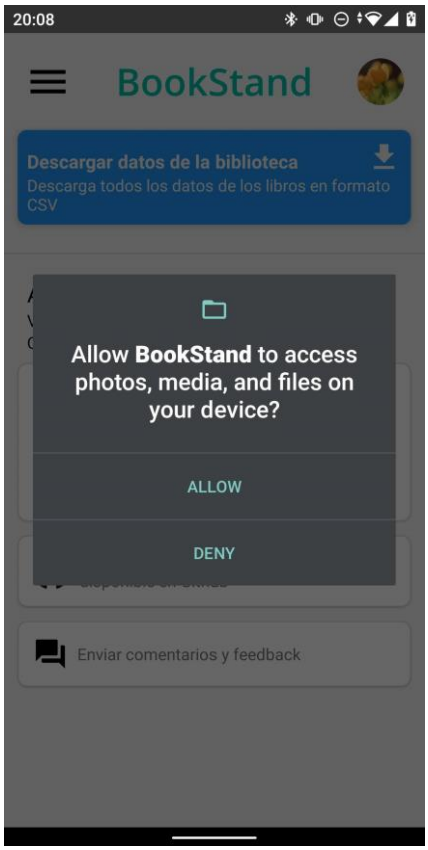

Ilustración 119 Permisos de almacenamiento necesarios para descargar un archivo

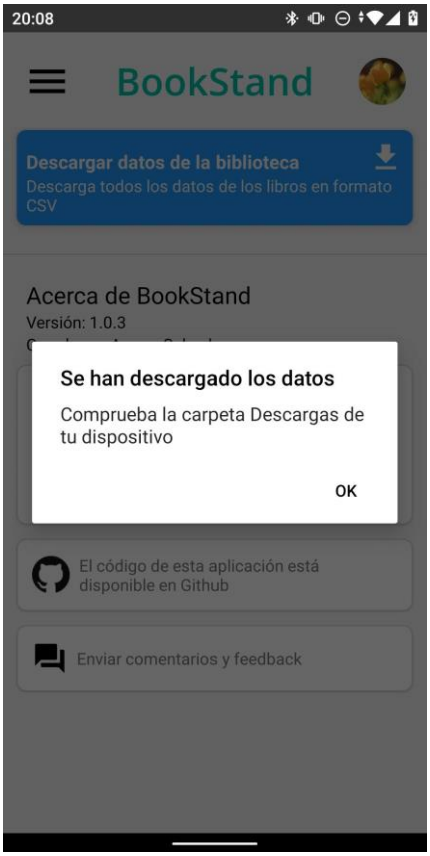

Ilustración 120 Mensaje que informa de la descarga de los datos

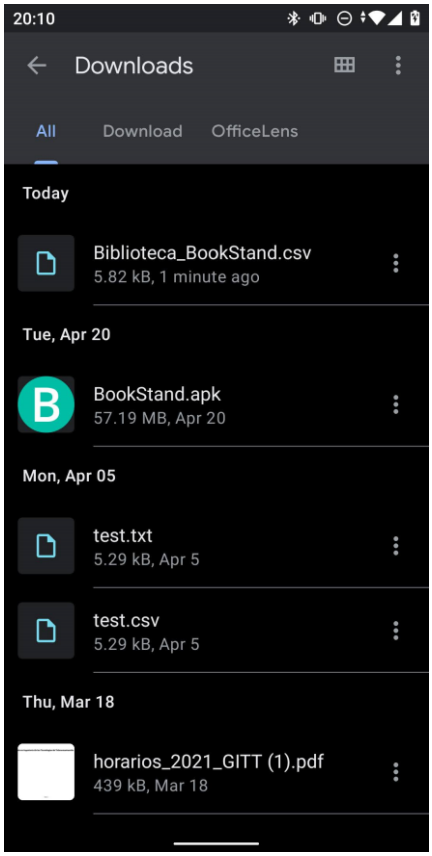

Ilustración 121 En la carpeta descargas los datos se guardan en el fichero “Biblioteca_BookStand.csv”

6 PRUEBAS E INSTALACIÓN

Como regla general, los sistemas de software no funcionan bien hasta que han sido utilizados, y han fallado en varias ocasiones, en aplicaciones reales.

Dave Parnas

La realización de pruebas del software en entornos y dispositivos reales es uno de los pasos fundamentales del desarrollo de aplicaciones móviles, ya que permite a los programadores descubrir errores o inconsistencias en sus aplicaciones. La información aportada por los usuarios que prueban las versiones preliminares resulta esencial para mejorar la calidad de la aplicación y la experiencia del usuario final.

Este capítulo tiene como objetivo describir de forma breve las pruebas realizadas de la aplicación para los sistemas operativos objetivo: Android y iOS.

6.1 Instalación y pruebas en Android

Debido a que el sistema operativo Android permite instalar aplicaciones no registradas en la Play Store, la tienda de aplicaciones para la plataforma ha sido posible ofrecer la aplicación a varios voluntarios para que la prueben.

Para ello necesitamos generar el archivo necesario para instalar la aplicación en los dispositivos Android: el archivo APK (Android Package Kit).

Primero debemos generar una clave pública RSA para poder firmar nuestro paquete. En la Ilustración 122 se muestra el comando y los datos necesarios para crear y configurar la clave. Una vez se ha creado la clave, la almacenamos en el directorio `android/app` de nuestro proyecto.

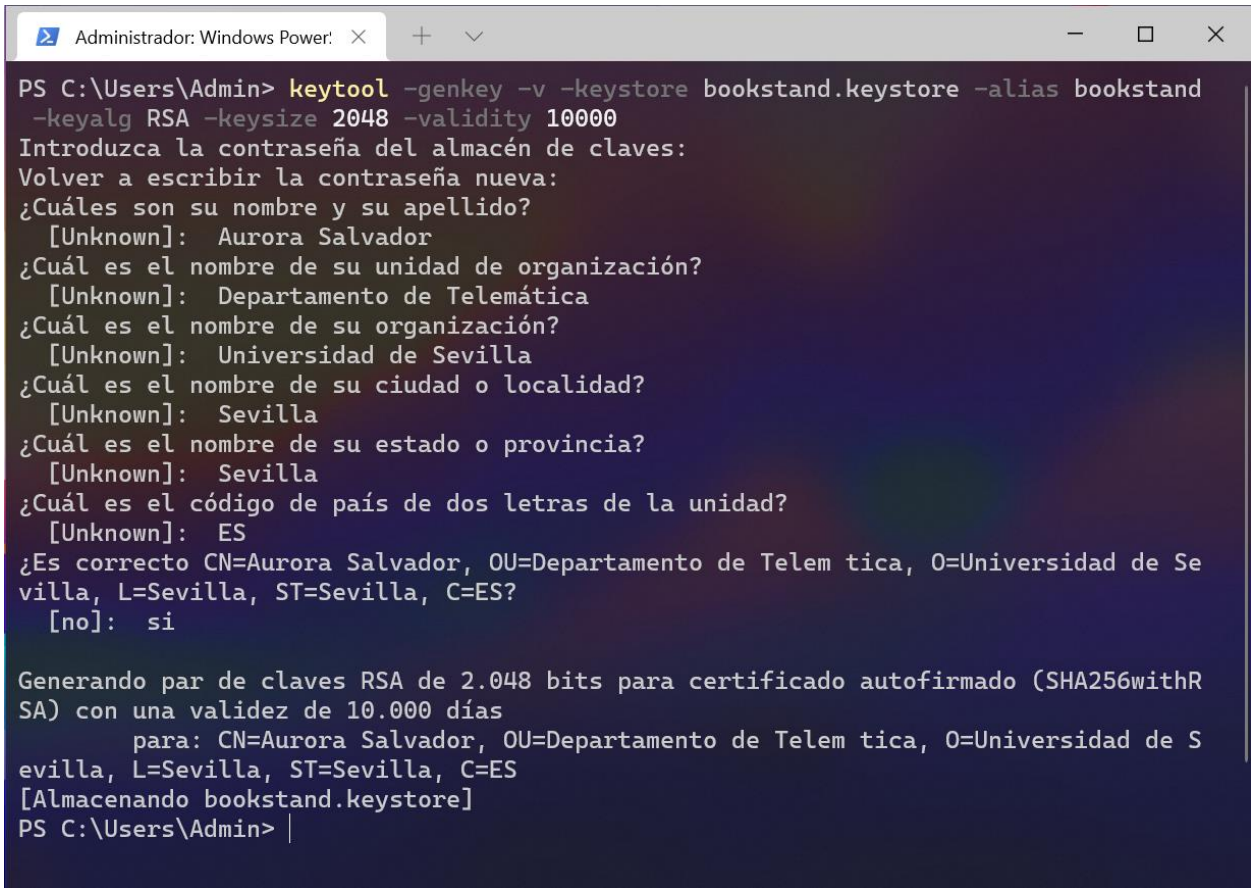

Ilustración 122 Creación de una clave RSA mediante el terminar de Windows

Posteriormente, debemos configurar el proyecto para que use esta clave cuando genere el archivo APK. En la Ilustración 123 se muestra la modificación que hay que realizar sobre el archivo `app/build.gradle`.

```
android {
    ...
    signingConfigs {
        release {
            storeFile file('bookstand.keystore')
                    storePassword 'tfgpass'
                    keyAlias 'bookstand'
                    keyPassword 'tfgpass'
        }
        ...
    }
    buildTypes {
                ...
        release {
            signingConfig signingConfigs.release
            ...
        }
    }

    ...
}
```

Ilustración 123 Modificaciones necesarias para configurar la clave de la versión de distribución

Esta configuración será la que se use cuando se vaya a construir el APK de la aplicación. Como se puede observar, en este archivo se guardan las contraseñas del certificado en claro, lo que no es una práctica muy segura, pero que por motivos de simpleza lo hemos dejado así. Se deja, como una posible mejora, realizar un script que en el momento de la creación del paquete de la aplicación se pidan las contraseñas por la línea de comandos.

Tras la modificación del fichero `app/gradle.build` generamos el archivo APK mediante el comando que se muestra en la :

```
npx gradlew assembleRelease
```

Ilustración 124 Comando para generar el fichero APK de la aplicación

Una vez finalice la ejecución del anterior comando, el paquete de la aplicación se genera en la ruta `android/app/build/outputs/apk/` de nuestro proyecto.

Como último paso antes de poder instalar la aplicación en el dispositivo, debemos añadir la huella digital SHA de la clave que generamos unas líneas más arriba en la consola de Firebase, como se muestra en la Ilustración 125:

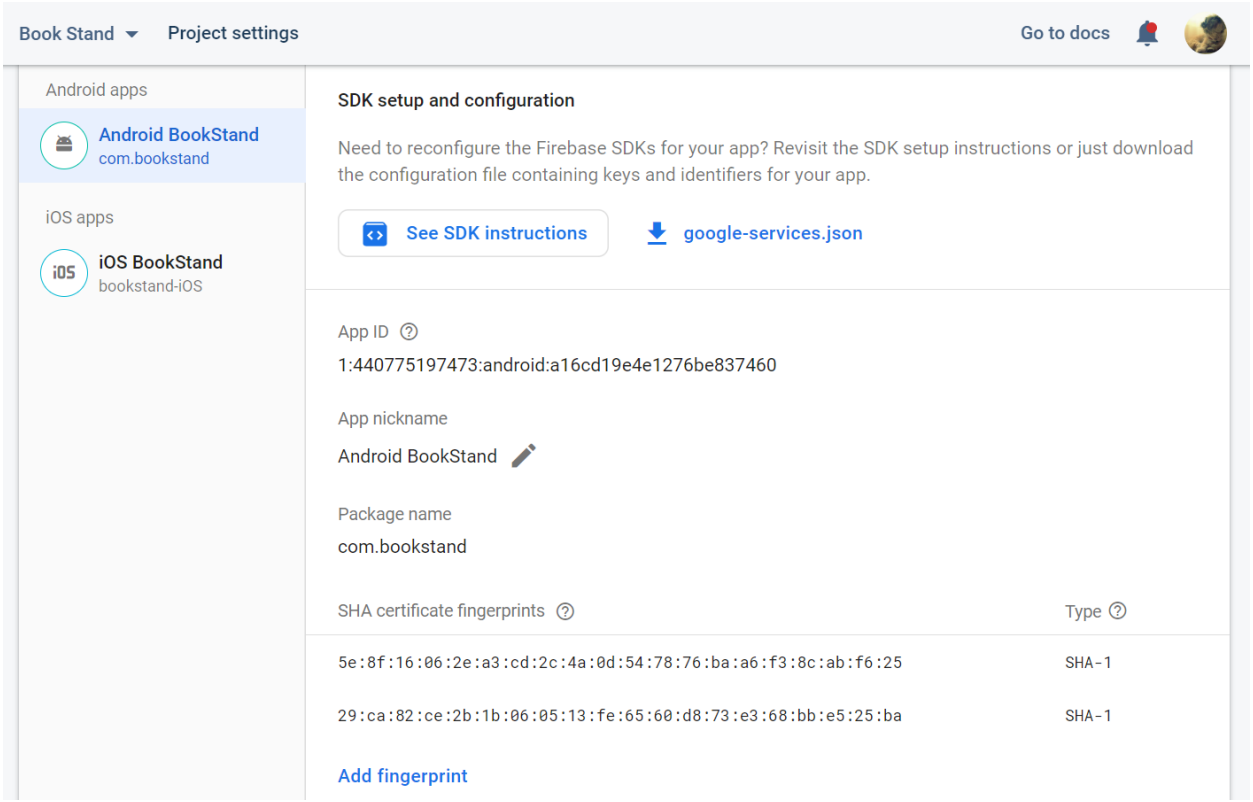

Ilustración 125 Añadimos las claves de nuestra aplicación al proyecto de Firebase

Para conocer las claves que debemos introducir en nuestra app, podemos ejecutar el siguiente comando que aparece en la Ilustración 126:

```
keytool -list -v -alias bookstand -keystore
.\BookStand\android\app\bookstand.keystore -storepass tfgpass -keypass
tfgpass
```

Ilustración 126 Comando para obtener las huellas SHA con las que configurar la aplicación

BookStand está lista para instalarse en un dispositivo real y, si abrimos el archivo APK desde un dispositivo Android, se preguntará al usuario si quiere instalarla, como se muestra en la Ilustración 127.

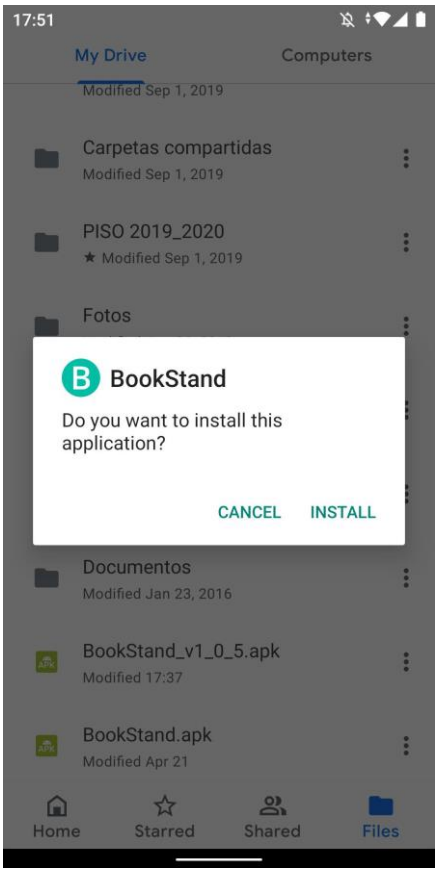

Ilustración 127 Cuadro de diálogo para instalar la aplicación

Sin embargo, debido a que hemos creado la aplicación con un certificado autofirmado, es posible que aparezca un mensaje de precaución durante la instalación de la aplicación (como sucede en la Ilustración 128) y la primera vez que abramos la aplicación (Ilustración 129).

Esto sucede debido a Google Play Protect [46], una herramienta de seguridad proporcionada por Google que escanea los dispositivos Android en busca de aplicaciones potencialmente peligrosas. Para poder realizar la instalación de forma correcta, los usuarios deben seleccionar las opciones “Instalar de todas formas” y “No enviar”.

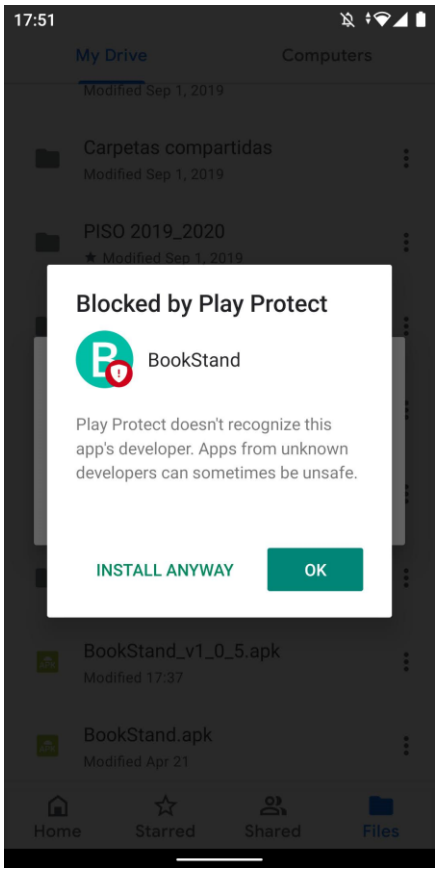

Ilustración 128 Cuadro de seguridad de Play Protect

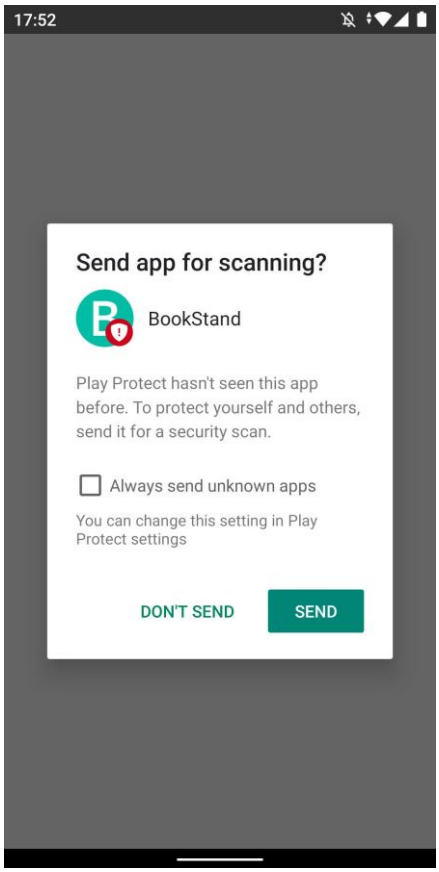

Ilustración 129 Mensaje de seguridad para escanear la app

Tras esto, la aplicación está completamente lista para que sea usada y probada por el usuario en su dispositivo físico.

El proceso de pruebas consistió en un conjunto de voluntarios que descargaron e instalaron el archivo APK en sus teléfonos y, durante el periodo de una semana, estuvieron probando la aplicación.

Al final del periodo de pruebas, algunos usuarios reportaron errores y sugerencias. Tras analizar y sopesar la retroalimentación proporcionada por los usuarios, se corrigieron los errores y se implementaron algunas de las sugerencias en la aplicación, consiguiendo así el objetivo principal de la fase de pruebas: la mejora de la calidad del software.

6.2 Pruebas en iOS

En esta sección describiremos las pruebas que se han realizado de la aplicación en iOS ya que, como comentábamos al principio de este documento, React Native permite desarrollar aplicaciones multiplataforma.

Antes de continuar, hay que mencionar que, debido a las restricciones que establece Apple para el desarrollo de aplicaciones para sus dispositivos, no se han podido realizar pruebas exhaustivas en este sistema operativo. En concreto, los principales obstáculos son la imposibilidad de desarrollar y probar las aplicaciones en un simulador iOS desde un sistema operativo que no sea macOS y el uso obligatorio de herramientas específicas de la plataforma, como Xcode.

Al no disponer de una máquina Apple con la que realizar las pruebas, se ha hecho uso de una máquina virtual que, aunque proporciona un entorno que está lejos de ser óptimo, al menos nos permite comprobar el funcionamiento de la aplicación.

Comenzaremos por describir el procedimiento necesario para configurar la aplicación para iOS:

Una vez configurado el entorno de desarrollo de la máquina virtual tal y como se describió en el apartado 3.1.6, utilizando Visual Studio Code y Github, clonamos el repositorio. Tras realizar este paso, debemos instalar los paquetes npm que usa este proyecto. En este caso, además de utilizar el comando `npm install`, es necesario utilizar el gestor de dependencias de iOS, CocoaPods, para añadir correctamente las librerías. Los comandos necesarios se muestran en la Ilustración 130:

```
cd ios
pod install
```

Ilustración 130 Comandos para añadir las librerías con CocoaPods

En ocasiones se deberán resolver conflictos generados por alguna de las librerías utilizadas o a la necesidad de ajustar el código fuente a las diferentes plataformas.

Una vez hecho esto, hay que configurar Firebase para su uso con aplicaciones iOS. Para ello, debemos añadir una nueva aplicación desde la consola de Firebase, como se muestra en Ilustración 131 Consola de Firebase: ajustes para una aplicación iOS, y descargar el archivo con la configuración, `GoogleService-Info.plist`, que incorporaremos a nuestro proyecto mediante Xcode.

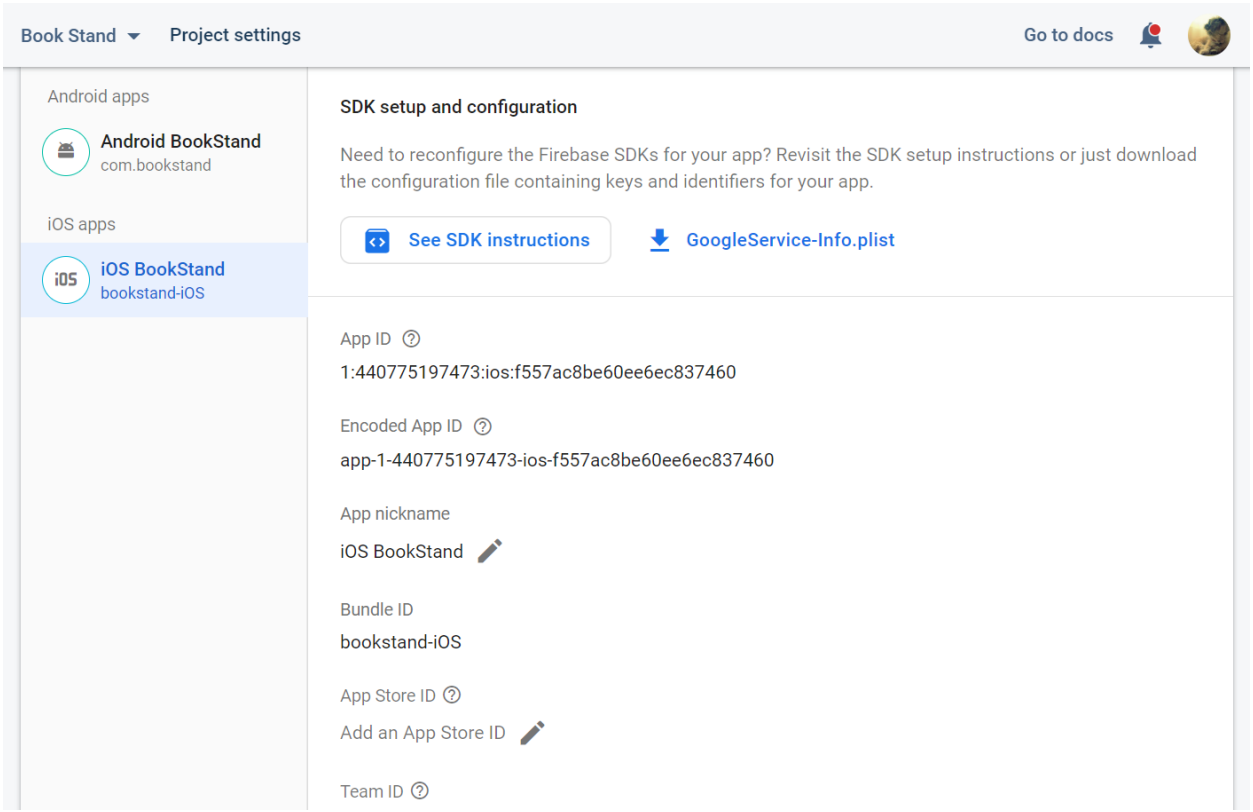

Ilustración 131 Consola de Firebase: ajustes para una aplicación iOS

Una vez hecho esto, ejecutamos el código en el simulador de iOS y comprobamos el funcionamiento de la app.

Es necesario comentar que, dadas las limitaciones que tienen los simuladores, hay partes de la aplicación que no se han podido probar, como el escaneo de los códigos ISBN a través de la cámara o las notificaciones push.

Las siguientes imágenes muestran la ejecución de BookStand en iOS:

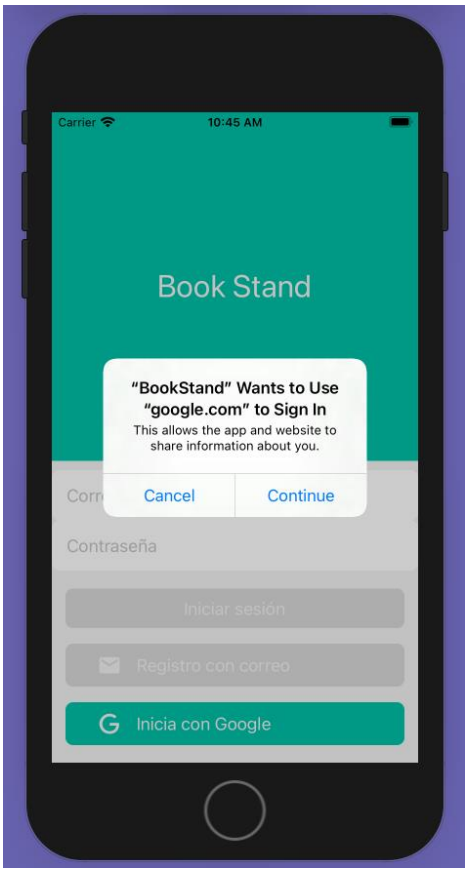

Ilustración 132 Permisos para iniciar con Google

Ilustración 133 Inicio de sesión con Google en iOS

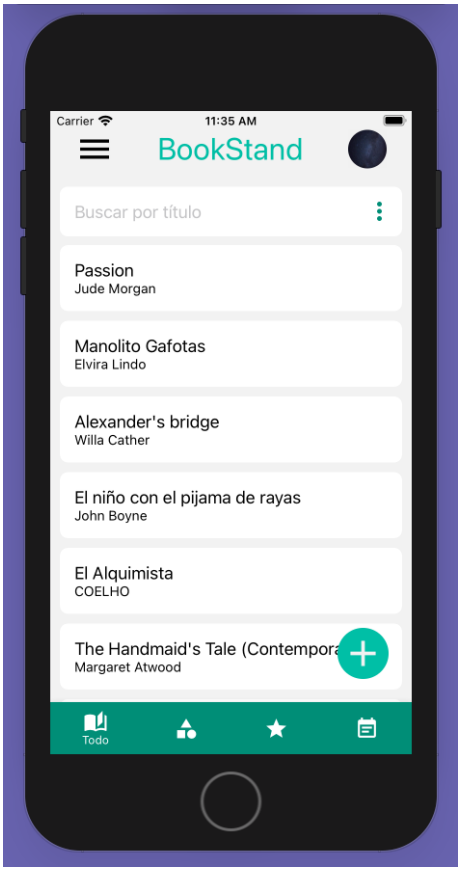

Ilustración 134 Pantalla Biblioteca en iOS

Ilustración 135 Filtros de búsqueda de la Biblioteca

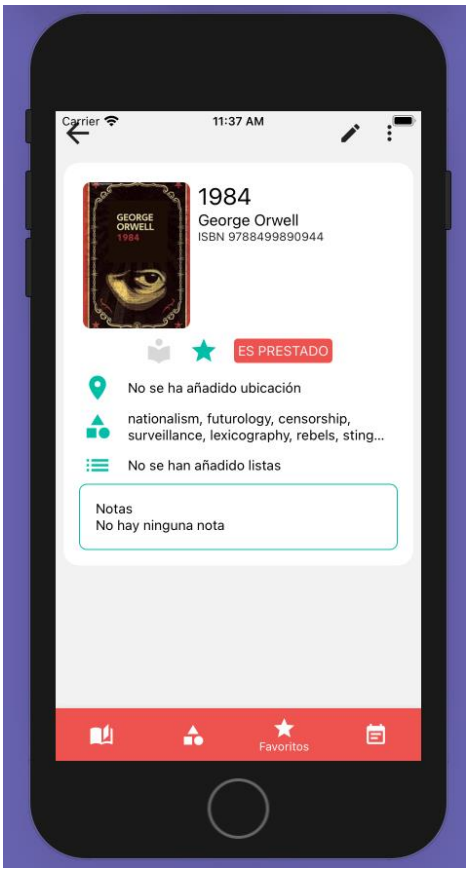

Ilustración 136 Detalles de un libro

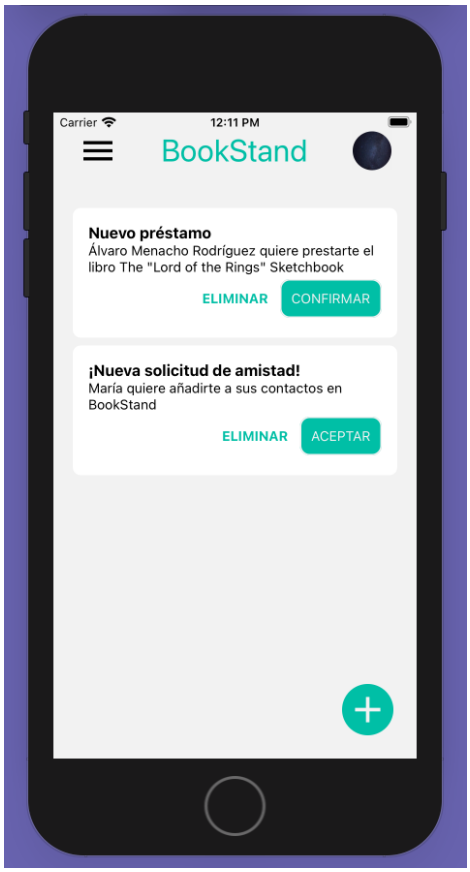

Ilustración 137 Pantalla principal de la aplicación

Ilustración 138 Tarjeta para añadir un préstamo

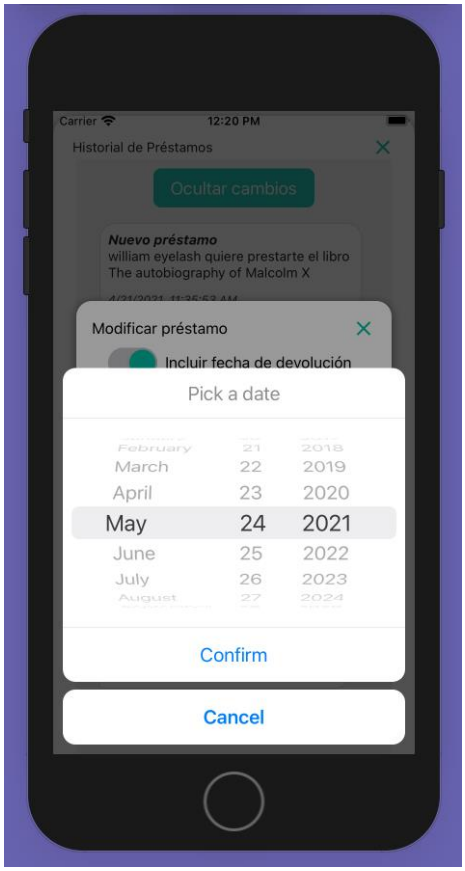

Ilustración 139 Selector de fecha para iOS

Ilustración 140 Tarjeta modificar un préstamo

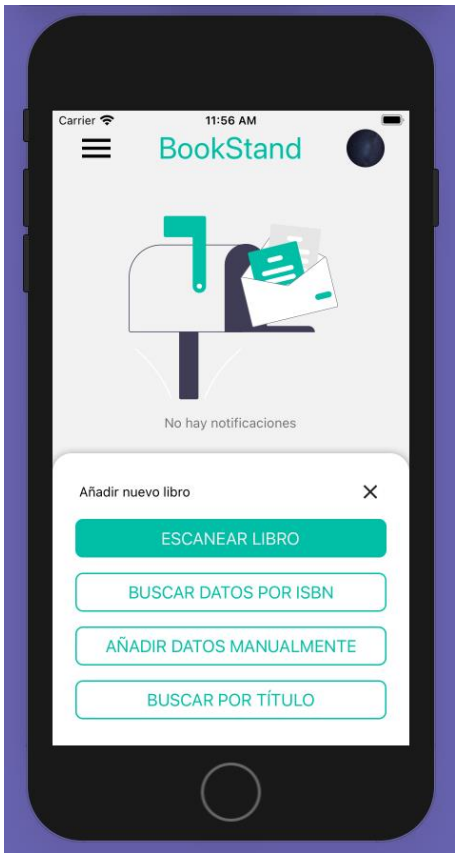

Ilustración 141 Opciones para añadir un libro

Ilustración 142 Lista de resultados al buscar un libro por título

Ilustración 143 Añadimos un libro a BookStand

7 LÍNEAS DE MEJORA Y CONCLUSIONES

La finalidad de este último capítulo es recoger las lecciones aprendidas y poder analizar las ventajas e inconvenientes del uso de React Native para el desarrollo de aplicaciones multiplataforma. Para ello retomaremos los objetivos que se especificaron en el apartado 1.2 de este documento.

Comenzaremos con las conclusiones derivadas del uso de React Native en el proyecto.

En lo que respecta a la facilidad de aprendizaje, podemos considerar que React Native se sitúa en una dificultad media. Si bien la sintaxis es sencilla y los lenguajes de programación que usa React Native son populares y conocidos, lo cierto es que existen conceptos más difíciles de comprender, en especial aquellos relacionados con el ciclo de vida de los componentes, como los hooks o los patrones de Redux.

Con respecto a la capacidad para realizar tareas avanzadas como enviar mensajes entre dispositivos o usar la cámara como escáner, podemos constatar que, en general, con React Native se pueden implementar estas funcionalidades con relativa facilidad siempre y cuando se usen módulos de la comunidad o librerías que nos lo permita. De no ser así, la dificultad y el tiempo necesario para desarrollar una aplicación aumentarían de forma considerable, ya que tendríamos que crear nuestros propios componentes o módulos nativos.

Si nos centramos en la eficiencia de la multiplataforma, desde el punto de vista de las modificaciones que se precisan realizar para asegurar el correcto funcionamiento de la aplicación para cada una de las plataformas, podemos concluir que los cambios necesarios han sido mínimos. Además, estos ajustes se han debido, principalmente a las diferentes formas de configuración de algunas de las librerías utilizadas para Android y para iOS.

Para finalizar, me gustaría comentar que, aunque una de las principales bondades de React Native es la gran cantidad de módulos de la comunidad que nos permiten implementar todo tipo de características con facilidad y rapidez, esto es también uno de sus mayores inconvenientes. Esto supone que, en ocasiones la viabilidad de implementar alguna funcionalidad en la aplicación depende de la existencia de una librería de terceros de calidad.

Un ejemplo de esto lo encontramos a la hora de programar los recordatorios de los préstamos que se describieron en el apartado Recordatorios y notificaciones. Inicialmente, para implementar dicha funcionalidad, se necesitaría programar una tarea asíncrona en el dispositivo que fuese capaz de iniciarse incluso si la aplicación estuviese en segundo plano o cerrada. Sin embargo, actualmente, las opciones que nos permiten realizar esta tarea requieren introducir código nativo para cada plataforma, es decir, crear la parte correspondiente a dicha funcionalidad en Java/Kotlin para Android o en Objective-C/Swift para iOS.

Por otro lado, dado que se ha usado Firebase servidor, conviene hacer referencia a la experiencia que resulta de utilizar este servicio. En relación con la facilidad de la integración con React Native, gracias a la amplia documentación de Firebase y a la librería React Native Firebase, podemos afirmar que el uso del entorno de servidor de Google con React Native es factible y totalmente compatible.

Si recordamos lo mencionado en el apartado de Firebase, era necesaria una cuenta de pago por uso para poder utilizar las funciones de Cloud, por lo que resulta interesante comentar el coste económico que ha supuesto el servidor. Así, durante los cuatro meses que ha estado activo el plan de pago del proyecto, el coste ha alcanzado un total de 0,07€. Estos gastos han sido enteramente debidos al almacenamiento de imágenes en Cloud Storage, ya que el uso del resto de módulos utilizados ha permanecido por debajo de los límites gratuitos que ofrece el plan de pago.

Si bien en un proyecto real con un mayor número de usuarios los costes aumentarán, no cabe duda de que Firebase proporciona una solución barata y de calidad para el desarrollador.

Para finalizar, se detallan un conjunto de posibles líneas de mejora que podrían llevarse a cabo para mejorar la calidad de la aplicación.

Desde el punto de vista de la funcionalidad de la aplicación y la experiencia de usuario:

1. Una de las posibles mejoras sería realizar **cambios en la interfaz de usuario** para hacer más sencillas algunas tareas, como los métodos para crear listas de libros.
2. **Perfeccionar el funcionamiento de los recordatorios de los préstamos**. En lugar de lanzarse cada vez que iniciásemos la aplicación, se debería programar esta tarea, lo que requiere utilizar código nativo.
3. Otra de las posibles mejoras consistiría en **explotar la parte social de la aplicación**, creando perfiles públicos de los usuarios en los que se puedan publicar los libros que se están leyendo actualmente, o los libros que tienen disponibles para prestar. Si esto lo acompañásemos de una función de búsqueda y un chat entre usuarios, se podría transformar BookStand en una red social de intercambios de libros.

Finalmente, a partir de todos los conocimientos adquiridos tras realizar este proyecto, podríamos hacer mejoras en la calidad y en la estructura del código, haciendo uso de patrones de diseños.

ANEXO A: CÓDIGO DE LAS FUNCIONES DE CLOUD DE FIREBASE

index.js

```
const admin = require("firebase-admin");
admin.initializeApp();

const db = admin.firestore();
// Para ignorar propiedades no definidas
db.settings({ignoreUndefinedProperties: true});

// Exportamos las funciones
exports.nuevoContacto = require("./contactos/nuevoContacto");
exports.eliminarContacto = require("./contactos/eliminarContactos");
exports.nuevoPrestamo = require("./prestamos/nuevoPrestamo");
exports.actualizarPrestamo = require("./prestamos/actualizarPrestamo");
```

nuevoContacto.js

```
const functions = require("firebase-functions");
const admin = require("firebase-admin");

const utilidades = require("../utilidades");
// Constantes para la base de datos
const usuarios = "usuarios";
const contactos = "contactos";
const solicitudAmistad = "SOLICITUD_AMISTAD";
const info = "INFO";

// Referencia a la base de datos Firestore
const db = admin.firestore();

exports.nuevoContacto = functions.firestore
    .document(`${usuarios}/{user_id}/${contactos}/{contacto_id}`)
    .onCreate((snapshot, context) => {
      // El destino y el origen de la notificación son
      // Destino: el contacto que hemos añadido a la colección "contactos"
      const destino = snapshot.data();
      // Origen: el contacto que ha realizar la acción
      const origen = context.params.user_id;

      if (destino.esContacto === "PENDIENTE") {
        // En este caso, el origen mandará una solicitud de amistad al destino
```

```
    /* Obtenemos el token de registro del dispositivo destino para poder
    enviar una notificación */
    db.collection(usuarios)
        .doc(destino.uid)
        .get()
        .then((respuestaDestino) => {
          // El token del destino
          const tokenDestino = respuestaDestino.data().token;

          /* Obtenemos los datos del origen para añadir los datos a la
          notificación */
          db.collection(usuarios)
              .doc(origen)
              .get()
              .then((respuestaOrigen) => {
                // Los datos del usuario origen son:
                const datosOrigen = respuestaOrigen.data();

                // Construimos el mensaje de la notificación
                const mensaje = {
                  notification: {
                    title: "¡Nueva solicitud de amistad!",
                    body: datosOrigen.nombre +
                      " quiere añadirte a sus contactos en BookStand",
                  },
                };

                // Enviamos el mensaje de la notificación
                utilidades.enviarMensaje(tokenDestino, mensaje);


                /* Y añadimos la notificación a la base de datos de
                notificaciones del destinatario */
                utilidades.agregarNotificacion(datosOrigen, destino.uid,
                    mensaje, solicitudAmistad);
              })
              .catch((error) => {
                console.log(
                    "Error al obtener los datos del origen",
                    error);
              });
        })
        .catch((error) => {
          console.log(
              "Error al obtener el token del destino",
              error);
        });
    } else if (destino.esContacto === "SI") {
      /* En este caso, el origen acaba de aceptar la solicitud de amistad del
```

```
    destino y le mandará una notificación */

    /* Obtenemos el token de registro del destino para poder enviar una
    notificación */
    db.collection(usuarios)
        .doc(destino.uid)
        .get()
        .then((respuestaDestino) => {
          // El token del destino
          const tokenDestino = respuestaDestino.data().token;

          /* Obtenemos los datos del origen para añadir los datos a la
          notificación */
          db.collection(usuarios)
              .doc(context.params.user_id)
              .get()
              .then((respuestaOrigen) => {
                // Los datos del usuario origen son:
                const datosOrigen = respuestaOrigen.data();

                // Construimos el mensaje de la notificación
                const mensaje = {
                  notification: {
                    title: "¡Solicitud de amistad aceptada!",
                    body: datosOrigen.nombre +
                      " ha aceptado tu petición de amistad",
                    image: datosOrigen.foto,
                  },
                };

                // Enviamos el mensaje
                utilidades.enviarMensaje(tokenDestino, mensaje);

                // Y añadimos la notificación a la base de datos de
                // notificaciones del destinatario
                utilidades.agregarNotificacion(datosOrigen, destino.uid,
                    mensaje, info);

                /* Actualizamos el contacto de PENDIENTE a SI en la base de
                datos del destino */
                db.collection(usuarios)
                    .doc(destino.uid)
                    .collection(contactos)
                    .doc(origen)
                    .update({esContacto: "SI"});
              })
              .catch((error) => {
                console.log(
                    "Error al obtener los datos del origen",
```

```
                        error);
                });
            })
            .catch((error) => {
              console.log(
                  "Error al obtener el token del destino",
                  error);
            });
      }

      return Promise.resolve();
    });
```

eliminarContacto.js

```
const functions = require("firebase-functions");
const admin = require("firebase-admin");

// Constantes para la base de datos
const usuarios = "usuarios";
const contactos = "contactos";

// Referencia a la base de datos Firestore
const db = admin.firestore();

exports.eliminarContacto = functions.firestore
    .document(`${usuarios}/{user_id}/${contactos}/{contacto_id}`)
    .onDelete((snapshot, context) => {
      /* Cuando se borra un contacto, se debe actualizar la base de datos del
      contacto borrado sin enviar ninguna notificación */
      db.collection(usuarios)
          .doc(context.params.contacto_id)
          .collection(contactos)
          .doc(context.params.user_id)
          .delete().then(() =>
            console.log("Se ha borrado el contacto correctamente"))
          .catch((error) =>
            console.log("Error al borrar el contacto", error));

      return Promise.resolve();
    });
```

nuevoPrestamo.js

```
const functions = require("firebase-functions");
const admin = require("firebase-admin");

// Constantes para la base de datos
const usuarios = "usuarios";
const libros = "libros";
const prestamos = "prestamos";

const TipoNotificacion = {
  actualizacionPrestamo: "ACTUALIZACION_PRESTAMO",
  info: "INFO",
  confirmacionPrestamo: "CONFIRMACION_PRESTAMO",
};
const EstadoConfirmacion = {
  pendienteCreacion: "PEND_CREACION",
  confirmadoCreacion: "OK_CREACION",
  pendienteModificacion: "PEND_MODIFICACION",
  confirmadoModificacion: "OK_MODIFICACION",
  pendienteDevolucion: "PEND_DEVOLUCION",
  confirmadoDevolucion: "OK_DEVOLUCION",
  denegadoCreacion: "DEN_CREACION",
  degenadoModidicacion: "DEN_MODIFICACION",
  denegadoDevolucion: "DEN_DEVOLUCION",
  ocioso: "OCIOSO",
};

// The database reference to perform writes and updates
const db = admin.firestore();

const utilidades = require("../utilidades");

exports.nuevoPrestamo = functions.firestore
    .document(
        `${usuarios}/{user_id}/${libros}/{lido_id}/${prestamos}/{prestamo_id}`)
    .onCreate((snapshot, context) => {
      /* Cuando agregamos un préstamo por primera vez, debemos enviar una
      notificación al deudor para confirmar el préstamo o, si acabamos de
      aceptar un préstamo, actualizar la base de datos del prestador*/

      // Los datos del préstamo son:
      const prestamo = snapshot.data();

      if (prestamo.estado === EstadoConfirmacion.pendienteCreacion) {
        /* En este caso, el prestador acaba de prestar un libro al deudor.
        Ddebemos enviar una notificación al deudor para confirmar el préstamo *
/
        // Destino: el usuario a quien queremos prestar el libro
```

```
        // Origen: el usuario que ha prestado el libro
        const destino = prestamo.uidDeudor;
        const datosOrigen = {
          uid: prestamo.uidPrestador,
          nombre: prestamo.nombrePrestador,
        };

        /* Obtenemos el token de registro del dispositivo destino para poder
        enviar una notificación */
        db.collection(usuarios)
            .doc(destino)
            .get()
            .then((respuestaDestino) => {
              // El token del destino
              const tokenDestino = respuestaDestino.data().token;

              // Construimos el mensaje de la notificación
              const mensaje = {
                notification: {
                  title: "Nuevo préstamo",
                  body: datosOrigen.nombre +
                          " quiere prestarte el libro " + prestamo.titulo,
                },
              };

              // Enviamos el mensaje de la notificación
              utilidades.enviarMensaje(tokenDestino, mensaje);

              /* Y añadimos la notificación a la base de datos de notificacione
s
              del destinatario */
              utilidades.agregarNotificacion(datosOrigen, destino, mensaje,
                  TipoNotificacion.confirmacionPrestamo, prestamo);
            })
            .catch((error) => {
              console.log(
                  "Error al obtener el token del destino",
                  error);
            });
      } else if (prestamo.estado === EstadoConfirmacion.denegadoCreacion) {
        // Se ha denegado el préstamo del libro
        // Obtenemos origen y destino de los mensajes FCM
        const destino = prestamo.uidPrestador;
        const datosOrigen = {
          uid: prestamo.uidDeudor,
          nombre: prestamo.nombreDeudor,
        };

        /* Obtenemos el token de registro del destino para poder enviar una
```

```
        notificación */
        db.collection(usuarios)
            .doc(destino)
            .get()
            .then((respuestaDestino) => {
              // El token del destino
              const tokenDestino = respuestaDestino.data().token;

              // Construimos el mensaje de la notificación
              const mensaje = {
                notification: {
                  title: "Denegación de préstamo",
                  body: datosOrigen.nombre +
                          " ha denegado el préstamo del libro " +
                          prestamo.titulo,
                },
              };

              // Enviamos el mensaje
              utilidades.enviarMensaje(tokenDestino, mensaje);

              /* Y añadimos la notificación a la base de datos de
              notificaciones del destinatario */
              utilidades.agregarNotificacion(datosOrigen, destino, mensaje,
                  TipoNotificacion.info, prestamo);

              /* Eliminamos los registros del préstamo de ambas partes*/
              utilidades.eliminarRegistroPrestamo(destino, prestamo);
              utilidades.eliminarRegistroPrestamo(datosOrigen.uid, prestamo);

              // Adicionalmente, para el destino, cambiamos el estado del libro
              utilidades.cambiarEstadoLibro(destino, prestamo.isbn, "NINGUNO");
            })
            .catch((error) => {
              console.log(
                  "Error al obtener el token del destino",
                  error);
            });
      } else if (prestamo.estado === EstadoConfirmacion.confirmadoCreacion) {
        /* En este caso, el deudor acaba de confirmar que le han prestado un
        libro. Ddebemos enviar una notificación al prestador para informarle
        de que se ha confirmado el préstamo */
        // Destino: el usuario que presta el libro
        // Origen: el usuario deudor
        const destino = prestamo.uidPrestador;
        const datosOrigen = {
          uid: prestamo.uidDeudor,
          nombre: prestamo.nombreDeudor,
        };
```

```
      /* Obtenemos el token de registro del destino para poder enviar una
      notificación */
      db.collection(usuarios)
          .doc(destino)
          .get()
          .then((respuestaDestino) => {
            // El token del destino
            const tokenDestino = respuestaDestino.data().token;

            // Construimos el mensaje de la notificación
            const mensaje = {
              notification: {
                title: "Confirmación de préstamo",
                body: datosOrigen.nombre +
                        " ha confirmado el préstamo del libro " +
                        prestamo.titulo,
              },
            };

            // Enviamos el mensaje
            utilidades.enviarMensaje(tokenDestino, mensaje);

            /* Y añadimos la notificación a la base de datos de
            notificaciones del destinatario */
            utilidades.agregarNotificacion(datosOrigen, destino, mensaje,
                TipoNotificacion.info, prestamo);

            /* Actualizamos el préstamo que ya no está pendiente de
            confirmación por el deudor en la base de datos del PRESTADOR */
            utilidades.cambiarEstadoPrestamo(destino, prestamo,
                EstadoConfirmacion.ocioso);

            // Finalmente, añadimos los datos del libro a la base de
            // datos del DEUDOR
            utilidades.agregarLibro(prestamo);
          })
          .catch((error) => {
            console.log(
                "Error al obtener el token del destino",
                error);
          });
    }

    return Promise.resolve();
  });
```

actualizarPrestamo.js

```
const functions = require("firebase-functions");
const admin = require("firebase-admin");

// Constantes para la base de datos
const usuarios = "usuarios";
const libros = "libros";
const prestamos = "prestamos";
const TipoNotificacion = {
  actualizacionPrestamo: "ACTUALIZACION_PRESTAMO",
  info: "INFO",
  confirmacionPrestamo: "CONFIRMACION_PRESTAMO",
};
const EstadoConfirmacion = {
  pendienteCreacion: "PEND_CREACION",
  confirmadoCreacion: "OK_CREACION",
  pendienteModificacion: "PEND_MODIFICACION",
  confirmadoModificacion: "OK_MODIFICACION",
  pendienteDevolucion: "PEND_DEVOLUCION",
  confirmadoDevolucion: "OK_DEVOLUCION",
  denegadoCreacion: "DEN_CREACION",
  degenadoModidicacion: "DEN_MODIFICACION",
  denegadoDevolucion: "DEN_DEVOLUCION",
  ocioso: "OCIOSO",
};

/* FIXME: Cuando actualizamos un préstamo el mensaje de confirmación
que se envía es "Se ha devuelto el libroundefined correctamente" */

// The database reference to perform writes and updates
const db = admin.firestore();

const utilidades = require("../utilidades");

exports.actualizarPrestamo = functions.firestore
    .document(
        `${usuarios}/{user_id}/${libros}/{lido_id}/${prestamos}/{prestamo_id}`)
    .onUpdate((change, context) => {
      /* Cuando se actualiza un préstamo, debemos enviar una notificación al
      otro extremo para confirmar los detalles */
      /* Habrá que notificar a la otra parte. El destino será el UID que no
      coincida con user_id (el origen) */

      // Los datos del préstamo son:
      const prestamo = change.after.data();

      // Obtenemos los datos del origen y el destino para la notificación
      // El origen es del usuario que genera el evento (user_id)
      const origen = context.params.user_id;
```

```
    // El destino será el UID que no sea el origen
    const datos = utilidades.obtenerOrigenDestino(prestamo, origen);
    const datosOrigen = datos.origen;
    const destino = datos.destino.uid;

    if (prestamo.estado === EstadoConfirmacion.pendienteModificacion ||
      prestamo.estado === EstadoConfirmacion.pendienteDevolucion) {
      // En este caso, se acaban de proponer los cambios del préstamo.
      /* Obtenemos el token de registro del dispositivo destino para poder
      enviar una notificación */
      db.collection(usuarios)
          .doc(destino)
          .get()
          .then((respuestaDestino) => {
            // El token del destino
            const tokenDestino = respuestaDestino.data().token;

            // Creamos un mensaje que será diferente según la situación
            let mensaje;
            if (prestamo.devuelto === true) {
              // Se quiere marcar el libro como devuelto
              mensaje = {
                notification: {
                  title: "Devolución de libro",
                  body: datosOrigen.nombre +
                    " quiere marcar como devuelto el libro " +
                    prestamo.titulo,
                },
              };
            } else {
              // Si no se ha marcado como devuelto, se ha modificado la fecha
              const fechaNueva = prestamo.fechaDevolucion;
              const fechaAntigua = change.before.data().fechaDevolucion;

              let cambio;
              if (fechaAntigua !== undefined) {
                if (fechaAntigua < fechaNueva) {
                  cambio = "retrasar";
                } else {
                  cambio = "adelantar";
                }
              } else {
                cambio = "establecer";
              }

              mensaje = {
                notification: {
                  title: "Cambios en el préstamo de un libro",
                  body: datosOrigen.nombre +
```

```
                    " quiere " + cambio +
                    " la fecha de devolución del libro " +
                    prestamo.titulo,
                },
              };
            }

            // Enviamos el mensaje de la notificación
            utilidades.enviarMensaje(tokenDestino, mensaje);

            /* Y añadimos la notificación a la base de datos de
                  notificaciones del destinatario */
            utilidades.agregarNotificacion(datosOrigen, destino, mensaje,
                TipoNotificacion.actualizacionPrestamo, prestamo);
          })
          .catch((error) => {
            console.log(
                "Error al obtener el token del destino",
                error);
          });
    } else if (prestamo.estado === EstadoConfirmacion.confirmadoDevolucion ||
      prestamo.estado === EstadoConfirmacion.confirmadoModificacion) {
      /* En este caso, se acaban de aceptar los cambios  o la devolución del
      préstamo */

      /* Obtenemos el token de registro del destino para poder enviar una
      notificación */
      db.collection(usuarios)
          .doc(destino)
          .get()
          .then((respuestaDestino) => {
            // El token del destino
            const tokenDestino = respuestaDestino.data().token;

            // Creamos un mensaje que será diferente según la situación
            let mensaje;
            if (prestamo.devuelto === true) {
              // Se quiere marcar el libro como devuleto
              mensaje = {
                notification: {
                  title: "Devolución de libro",
                  body: "Se ha devuelto el libro " + prestamo.titulo +
                    " correctamente",
                },
              };
            } else {
              /* Si no se ha marcado el libro como devuelto, se han aceptado
              los cambios */
              mensaje = {
```

```
            notification: {
              title: "Cambios aceptados",
              body: datosOrigen.nombre +
                " ha aceptado los cambios en el préstamo del libro " +
              prestamo.titulo,
            },
          };
        }

        // Enviamos el mensaje
        utilidades.enviarMensaje(tokenDestino, mensaje);

        /* Y añadimos la notificación a la base de datos de
        notificaciones del destinatario */
        utilidades.agregarNotificacion(datosOrigen, destino, mensaje,
            TipoNotificacion.info, prestamo);

        /* Actualizamos el préstamo que ya no está pendiente de
        confirmación por el deudor en la base de datos del destino */
        utilidades.cambiarEstadoPrestamo(destino, prestamo,
            EstadoConfirmacion.ocioso);

        /* Finalmente, actualizamos los datos del libro del DESTINO
        (prestado a NINGUNO) si se ha devuelto el libro */
        if (prestamo.devuelto) {
          utilidades.cambiarEstadoLibro(destino, prestamo.isbn,
              "NINGUNO");
        }
      })
      .catch((error) => {
        console.log(
            "Error al obtener el token del destino",
            error);
      });
  } else if (prestamo.estsado === EstadoConfirmacion.denegadoDevolucion ||
    prestamo.estado === EstadoConfirmacion.degenadoModidicacion) {
    // Se han denegado los cambios

    /* Obtenemos el token de registro del destino para poder enviar una
    notificación */
    db.collection(usuarios)
        .doc(destino)
        .get()
        .then((respuestaDestino) => {
          // El token del destino
          const tokenDestino = respuestaDestino.data().token;

          // Creamos un mensaje que será diferente según la situación
          let mensaje;
```

```
            if (prestamo.estado === EstadoConfirmacion.denegadoDevolucion) {
              mensaje = {
                notification: {
                  title: "Devolución de libro rechazada",
                  body: datosOrigen.nombre +
                    " ha rechazado la devolución del libro " +
                    prestamo.titulo,
                },
              };
            } else {
              /* Si se han rechazado los cambios */
              mensaje = {
                notification: {
                  title: "Cambios rechazados",
                  body: datosOrigen.nombre +
                    " ha rechazado los cambios en el préstamo del libro " +
                  prestamo.titulo,
                },
              };
            }

            // Enviamos el mensaje
            utilidades.enviarMensaje(tokenDestino, mensaje);

            /* Y añadimos la notificación a la base de datos de
            notificaciones del destinatario */
            utilidades.agregarNotificacion(datosOrigen, destino, mensaje,
                TipoNotificacion.info, prestamo);

            /* Actualizamos el préstamo en la base de datos del origen */
            utilidades.cambiarEstadoPrestamo(origen, prestamo,
                EstadoConfirmacion.ocioso);

            // Y el préstamo en el destino
            db.collection(usuarios)
                .doc(destino)
                .collection(libros)
                .doc(prestamo.isbn)
                .collection(prestamos)
                .doc(prestamo.id)
                .update({...prestamo, estado: EstadoConfirmacion.ocioso});
          })
          .catch((error) => {
            console.log(
                "Error al obtener el token del destino",
                error);
          });
    }
```

```
      return Promise.resolve();
    });
```

utilidades.js

```
const {firestore} = require("firebase-admin");
const admin = require("firebase-admin");
// Constantes para la base de datos
const usuarios = "usuarios";
const notificaciones = "notificaciones";
const libros = "libros";
const prestamos = "prestamos";

// The database reference to perform writes and updates
const db = admin.firestore();

// Referencia a las plataforma de Cloud Messaging
const messaging = admin.messaging();

/**
 * Clase que contiene un conjunto de funciones auxiliares para simplificar el
 * código de las funciones de Cloud
 */
class Utilidades {
  /**
   * Función para crear una notificación y añadirla a la base de datos del
   * destino
   * @param {object} datosOrigen
   * @param {string} uidDestino
   * @param {object} mensaje
   * @param {string} tipo
   * @param {object} prestamo
   */
  static agregarNotificacion(datosOrigen, uidDestino, mensaje, tipo, prestamo)
{
    let notificacion;

    if (prestamo !== undefined) {
      notificacion = {
        origen: {
          uid: datosOrigen.uid,
          nombre: datosOrigen.nombre,
          foto: datosOrigen.foto,
        },
        titulo: mensaje.notification.title,
        cuerpo: mensaje.notification.body,
        tipo: tipo,
        leida: false,
```

```
      prestamo: prestamo,
      fecha: admin.firestore.Timestamp.now(),
    };
  } else {
    notificacion = {
      origen: {
        uid: datosOrigen.uid,
        nombre: datosOrigen.nombre,
        foto: datosOrigen.foto,
      },
      titulo: mensaje.notification.title,
      cuerpo: mensaje.notification.body,
      tipo: tipo,
      leida: false,
      fecha: admin.firestore.Timestamp.now(),
    };
  }

  db.collection(usuarios)
      .doc(uidDestino)
      .collection(notificaciones)
      .add(notificacion);
}

/**
 * Función que agrega los datos de un libro prestado a la base de
 * datos del usuario deudor de un préstamo
 * @param {object} prestamo
 */
static agregarLibro(prestamo) {
  // Obtenemos la información del libro que se presta
  db.collection(usuarios)
      .doc(prestamo.uidPrestador)
      .collection(libros)
      .doc(prestamo.isbn)
      .get()
      .then((bookRef) => {
      // Obtenemos los datos del libro
        const datosLibro = bookRef.data();

        /* Solo pasaremos al usuario deudor los datos básicos (título, isbn,
        autores y géneros) */
        const libro = {
          titulo: datosLibro.titulo,
          autores: datosLibro.autores,
          isbn: datosLibro.isbn,
          generos: datosLibro.generos,
          prestado: "DEUDOR",
          esPropietario: false,
```

```
          };

          /* Añadimos el libro a la base de datos del deudor */
          db.collection(usuarios)
              .doc(prestamo.uidDeudor)
              .collection(libros)
              .doc(prestamo.isbn)
              .set(libro);
        })
        .catch((error) =>
          console.error("Error al incluir los datos del libro", error),
        );
  }

  /**
   * Función que envía un mensaje a un dispositivo a través de Cloud Messaging
   * @param {string} tokenDestino
   * @param {object} mensaje
   */
  static enviarMensaje(tokenDestino, mensaje) {
    // Enviamos el mensaje de la notificación solo si tenemos un Token
    if (tokenDestino) {
      messaging
          .sendToDevice(tokenDestino, mensaje)
          .then((IdMensaje) => {
            console.log("Se ha enviado un mensaje correctamente: " + IdMensaje)
;
          })
          .catch((error) => {
            console.log("Error al enviar el mensaje " + error);
          });
    }
  }

  /**
   * Función que obtiene los nombres y uids del origen y el destino de una
   * notificación relacionada con los préstamos
   * @param {object} prestamo
   * @param {string} origen
   * @return {object} datos
   */
  static obtenerOrigenDestino(prestamo, origen) {
    let datosDestino;
    let datosOrigen;
    if (origen === prestamo.uidDeudor) {
      datosDestino = {
        uid: prestamo.uidPrestador,
        nombre: prestamo.nombrePrestador,
      };
```

```
    datosOrigen = {
      uid: prestamo.uidDeudor,
      nombre: prestamo.nombreDeudor,
    };
  } else if (origen === prestamo.uidPrestador) {
    datosOrigen = {
      uid: prestamo.uidPrestador,
      nombre: prestamo.nombrePrestador,
    };
    datosDestino = {
      uid: prestamo.uidDeudor,
      nombre: prestamo.nombreDeudor,
    };
  }

  const datos = {
    origen: datosOrigen,
    destino: datosDestino,
  };
  return datos;
}

/**
 * Función para enviar el registro de un préstamo de la base de datos de un
 * usuario
 * @param {string} usuario
 * @param {object} prestamo
 */
static eliminarRegistroPrestamo(usuario, prestamo) {
  db.collection(usuarios)
      .doc(usuario)
      .collection(libros)
      .doc(prestamo.isbn)
      .collection(prestamos)
      .doc(prestamo.id)
      .delete()
      .catch((error) => console.log(error));
}

/**
 * Función para cambiar el estado de un préstamo en la base de datos de un
 * usuario
 * @param {string} usuario
 * @param {obtect} prestamo
 * @param {string} estado
 */
static cambiarEstadoPrestamo(usuario, prestamo, estado) {
  firestore()
      .collection(usuarios)
```

```
        .doc(usuario)
        .collection(libros)
        .doc(prestamo.isbn)
        .collection(prestamos)
        .doc(prestamo.id)
        .update({estado: estado})
        .catch((error) =>
          console.log("Error al cambiar el estado del préstamo", error),
        );
  }

  /**
   * Función para cambiar el estado de un libro en la base de datos del usuario
   * @param {string} usuario
   * @param {string} libro
   * @param {string} estado
   */
  static cambiarEstadoLibro(usuario, libro, estado) {
    firestore()
        .collection(usuarios)
        .doc(usuario)
        .collection(libros)
        .doc(libro)
        .update({prestado: estado})
        .catch((error) =>
          console.log("Error actualizando el estado del libro", error),
        );
  }
}

module.exports = Utilidades;
```

ANEXO B: COMPILACIÓN DE LA APLICACIÓN

En este anexo se detallan los pasos a seguir para crear y poner en marcha el proyecto BookStand desde cero. De forma más concreta, se detallan los pasos que se deben tomar desde un ordenador con Windows 10 para probar la aplicación con un emulador Android.

Partiremos de la premisa de que se encuentra instalado todo el software necesario para utilizar *React Native CLI QuickStart*. Se puede consultar el procedimiento de instalación de dicho software en la documentación oficial de React Native [47].

Una vez dicho esto, pasaremos a la creación del proyecto React Native. Como ya comentamos a lo largo del documento, utilizaremos TypeScript como lenguaje de programación, así que deberemos indicarlo mediante la opción `--template`. Así pues, abrimos una ventana de terminal PowerShell e introducimos el siguiente comando:

```
npx react-native init NombreProyecto --template react-native-template-
typescript
```

Este comando creará un directorio con un conjunto de archivos y subdirectorios necesarios para el proyecto. De todos ellos destacaremos los más importantes. Con respecto a los directorios:

- El directorio `android` contiene el proyecto de Android Studio con todos los archivos y subdirectorios necesarios para compilar y configurar la aplicación React Native para su uso en esta plataforma. Uno de los archivos más importantes que alberga es `AndroidManifest.xml`, en el que se describen el nombre, los permisos o la compatibilidad con las diferentes versiones de Android. En ocasiones, cuando se utilicen algunas librerías React Native de terceros, será necesario modificar este archivo para conceder a la aplicación los permisos necesarios.
- De forma análoga, el directorio `ios` contiene el proyecto de Xcode necesario para poder compilar la aplicación para su uso con los emuladores de iPhone. También será necesario editar alguno de sus archivos, como `Info.plist` para configurar la app correctamente.
- Finalmente, el directorio `node_modules` contiene todos los módulos y librerías, tanto esenciales como de terceros, que usa nuestra aplicación.

Con respecto a los archivos:

- `index.js`. Es el punto de entrada de nuestra aplicación y registra el componente raíz.
- `package.json`. Un fichero que contiene información básica sobre la app, como el nombre, la versión y una lista de todos los módulos que usa.
- `App.tsx`. Es el fichero que contiene el componente raíz de la aplicación. En aplicaciones sencillas basta con un único archivo, pero en aplicaciones de mayor complejidad como BookStand, se opta por crear un directorio de nombre `src`, en el que se contendrá todo el código.

En este punto, estaríamos listos para desarrollar nuestra aplicación. Para copiar la funcionalidad de BookStand, una vez que tenemos el esqueleto del proyecto, descargamos el código de nuestro repositorio de GitHub [35] y movemos el contenido de la carpeta `App` a la carpeta raíz. Además, deberemos eliminar el archivo `App.tsx` que se creó de forma automática al iniciar el proyecto, y conservar solo el archivo con el mismo nombre que se encuentra dentro de la carpeta `src`.

Con nuestro código ya insertado en el proyecto, es hora de instalar todos los módulos que usará la aplicación. Para ello, simplemente debemos ejecutar el comando `npm install` desde la raíz de nuestro proyecto. Este comando se encargará de instalar todos los módulos que se incluyen en el archivo `package.json`.

Una vez finalice el proceso de instalación, es necesario configurar algunos de los módulos haciendo modificaciones en los directorios android y ios. Para esto seguimos las guías de configuración que se detallan en la documentación de cada uno de ellos. Los módulos que necesitan configuración adicional son:

- oblador/react-native-vector-icons [48]
- zo0r/react-native-push-notification [49]
- react-native-camera/react-native-camera [50]
- itinance/react-native-fs [51]
- Módulos relativos a Firebase. Por motivos de seguridad, no se puede hacer un proyecto de Firebase público, ya que esto vulneraría las credenciales de este. Debido a esto, si se quiere replicar y compilar el proyecto, es necesario crear un nuevo proyecto de Firebase y vincularlo a nuestra aplicación.

  Para ello, se recomienda seguir los pasos señalados en la consola de Firebase [52] para crear un proyecto. Una vez creado, descargamos el archivo `google-services.json` y lo ubicamos en la carpeta `android/app/` de nuestro proyecto y modificamos los archivos `build.gradle` del proyecto Android para indicar a la aplicación las credenciales de nuestro proyecto React Native. Se puede encontrar información más detallada en la documentación de React Native Firebase [14].

Por último, nos queda configurar las funciones de Cloud. Nos aseguramos de tener instaladas las herramientas necesarias para este paso instalando las herramientas de Firebase con el siguiente comando:

```
npm install -g firebase-tools
```

A continuación, mediante el comando `npx firebase login`, autenticamos la herramienta de Firebase con el proyecto que habíamos creado:

```
PS C:\Users\Admin\FirebaseFunctions> npx firebase login
i  Firebase optionally collects CLI usage and error reporting information to help improve our products. Data is collecte
d in accordance with Google's privacy policy (https://policies.google.com/privacy) and is not used to identify you.

? Allow Firebase to collect CLI usage and error reporting information? No

Visit this URL on this device to log in:
https://accounts.google.com/o/oauth2/auth?client_id=563584335869-fgrhgmd47bqnekij5i8b5pr03ho849e6.apps.googleusercontent
.com&scope=email%20openid%20https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fcloudplatformprojects.readonly%20https%3A%2F%2Fwww
.googleapis.com%2Fauth%2Ffirebase%20https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fcloud-platform&response_type=code&state=96
369267&redirect_uri=http%3A%2F%2Flocalhost%3A9005

Waiting for authentication...

+  Success! Logged in as aurora.salgu@gmail.com
PS C:\Users\Admin\FirebaseFunctions>
```

Ilustración 144 Autenticación del proyecto de Firebase

Hecho esto, creamos el proyecto de Firebase con el comando `npx firebase init functions`:

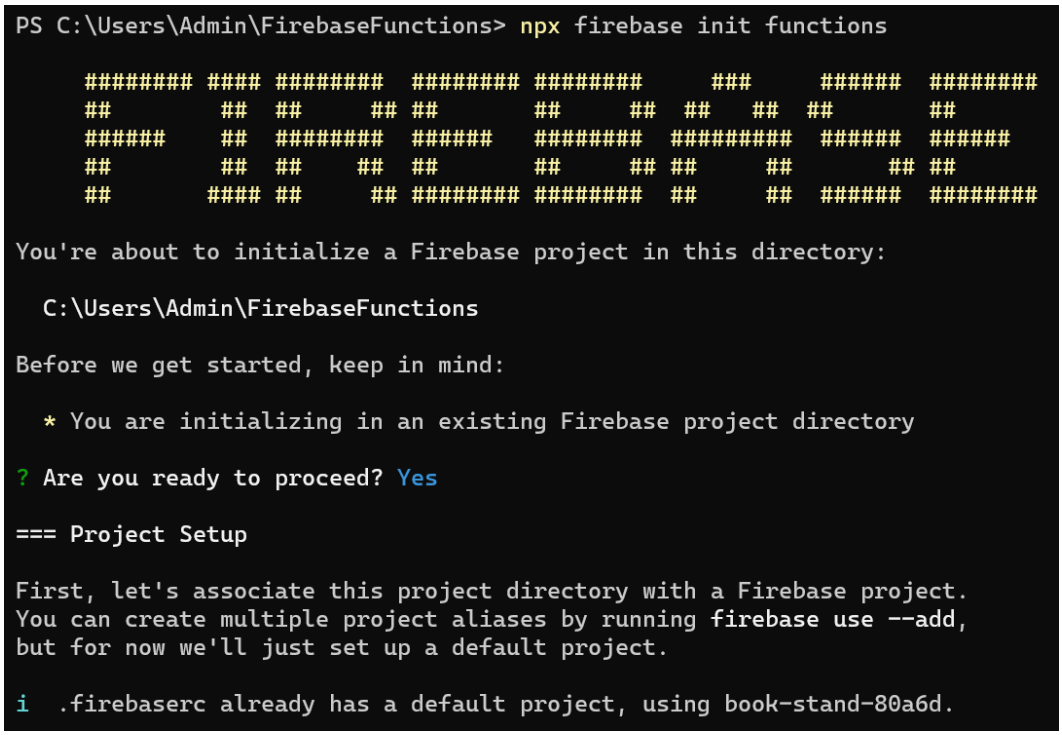

Ilustración 145 Creación del proyecto de Firebase Functions

Una vez finalizada la configuración, estamos listos para escribir el código de las funciones de Cloud en este directorio. En el repositorio de GitHub [35] se puede encontrar este código en el directorio "FirebaseFunctions". Para desplegar las funciones de Cloud, se hace uso del comando: `npx firebase deploy`:

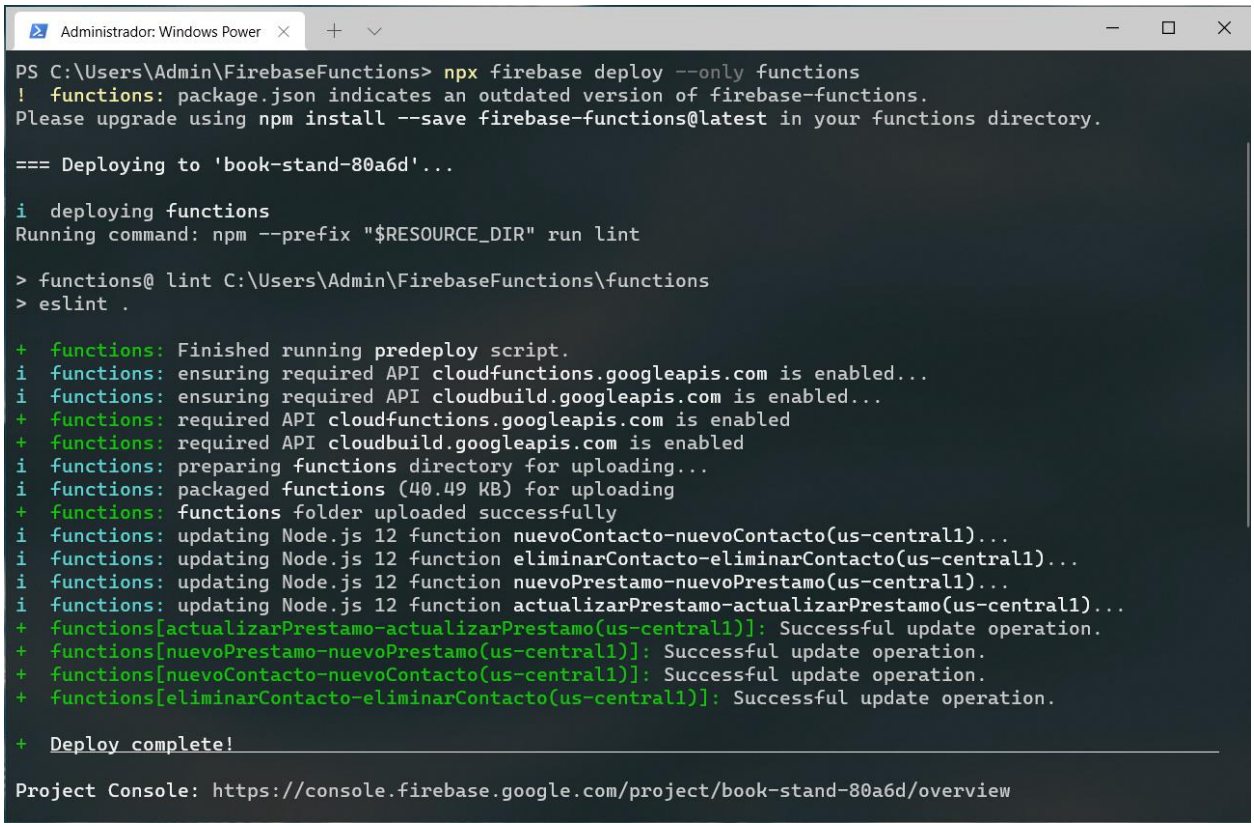

Ilustración 146 Despliegue de las funciones de Cloud

Tras configurar correctamente los módulos y el servidor, estamos en condiciones de ejecutar nuestra aplicación. Para ello, desde la raíz del proyecto ejecutamos el comando `npx react-native run-android`, que se encargará de compilar la aplicación y lanzar un emulador de Android para ejecutarla.

REFERENCIAS

[1] E. Hamilton, «Why Mobile Apps Are Important For Your Business?,» Tech Times, 4 mayo 2019. [En línea]. Available: https://www.techtimes.com/brandspin/242588/20190504/why-mobile-apps-are-important-for-your-business.htm. [Último acceso: 31 marzo 2021].

[2] Statista, «Mobile operating systems' market share worldwide from January 2012 to January 2021,» 8 febrero 2021. [En línea]. Available: https://www.statista.com/statistics/272698/global-market-share-held-by-mobile-operating-systems-since-2009/. [Último acceso: 31 marzo 2021].

[3] Facebook, «React Native,» 2021. [En línea]. Available: https://reactnative.dev/. [Último acceso: 6 abril 2021].

[4] Facebook, «React,» 2021. [En línea]. Available: https://reactjs.org/. [Último acceso: 6 abril 2021].

[5] Mozilla, «Classes,» 19 febrero 2021. [En línea]. Available: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Classes. [Último acceso: 6 abril 2021].

[6] React Native, «Using the State Hook,» Facebook, 2021. [En línea]. Available: https://reactjs.org/docs/hooks-state.html. [Último acceso: 6 abril 2021].

[7] React Native, «Using the Effect Hook,» Facebook, 2021. [En línea]. Available: https://reactjs.org/docs/hooks-effect.html. [Último acceso: 6 abril 2021].

[8] React, «State and Lifecycle,» Facebook, 2021. [En línea]. Available: https://reactjs.org/docs/state-and-lifecycle.html. [Último acceso: 6 abril 2021].

[9] React Native, «Core Components and Native Components,» Facebook, 2021. [En línea]. Available: https://reactnative.dev/docs/intro-react-native-components. [Último acceso: 6 abril 2021].

[10] Microsoft, «TypeScript: Typed JavaScript at Any Scale,» 2021. [En línea]. Available: https://www.typescriptlang.org/. [Último acceso: 6 abril 2021].

[11] npm, Inc, «npm,» 2014. [En línea]. Available: https://www.npmjs.com/. [Último acceso: 13 abril 2021].

[12] Expo & Software Mansion, «React Navigation | React Navigation,» 2021. [En línea]. Available: https://reactnavigation.org/. [Último acceso: 6 abril 2021].

[13] R. Dan Abramoc, «Redux - A predictable state conainer for JavaScript apps | Redux,» 2021. [En línea]. Available: https://redux.js.org/. [Último acceso: 6 abril 2021].

[14] Invertase, «React Native Firebase,» 2021. [En línea]. Available: https://rnfirebase.io/. [Último acceso: 6 abril 2021].

[15] Google, «Documentación,» 2021. [En línea]. Available: https://firebase.google.com/docs. [Último acceso: 7 abril 2021].

[16] Google, «Firebase Products,» 2021. [En línea]. Available: https://firebase.google.com/products-build. [Último acceso: 7 abril 2021].

[17] Google, «Firebase Pricing,» 2021. [En línea]. Available: https://firebase.google.com/pricing. [Último acceso: 7 abril 2021].

[18] Google Developers, «Firebase Authentication,» 17 noviembre 2020. [En línea]. Available: Firebase Authentication. [Último acceso: 7 abril 2021].

[19] Google Developers, «Cloud Firestore,» 16 diciembre 2020. [En línea]. Available: https://firebase.google.com/docs/firestore. [Último acceso: 7 abril 2021].

[20] Google Developers, «Cloud Functions for Firebase,» 17 febrero 2021. [En línea]. Available: https://firebase.google.com/docs/functions. [Último acceso: 7 abril 2021].

[21] Google Developers, «Cloud Storage for Firebase,» 17 febrero 2021. [En línea]. Available: Cloud Storage for Firebase. [Último acceso: 7 abril 2021].

[22] Google Developers, «Firebase Extensions,» 22 enero 2021. [En línea]. Available: https://firebase.google.com/docs/extensions. [Último acceso: 7 abril 2021].

[23] Google Developers, «Firebase Machine Learning,» 16 diciembre 2020. [En línea]. Available: https://firebase.google.com/docs/ml. [Último acceso: 7 abril 2021].

[24] Google Developers, «Barcode Scanning | ML Kit | Google Developers,» 19 enero 2021. [En línea]. Available: https://developers.google.com/ml-kit/vision/barcode-scanning?authuser=0&%3Bhl=es_419&hl=es_419. [Último acceso: 7 abril 2021].

[25] Google Developers, «Firebase Cloud Messaging,» 4 marzo 2021. [En línea]. Available: https://firebase.google.com/docs/cloud-messaging. [Último acceso: 7 abril 2021].

[26] Google Developers, «FCM Architectural Overview,» 22 enero 2021. [En línea]. Available: https://firebase.google.com/docs/cloud-messaging/fcm-architecture. [Último acceso: 7 abril 2021].

[27] Dan Abramov, Redux documentation authors, «Redux Essentials, Part 1: Redux Overview and Concepts,» 2021. [En línea]. Available: https://redux.js.org/tutorials/essentials/part-1-overview-concepts. [Último acceso: 6 abril 2021].

[28] Internet Archive, «Bienvenido a Open Library,» 2021. [En línea]. Available: https://openlibrary.org/. [Último acceso: 7 abril 2021].

[29] Internet Archive, «Internet Archive: Digital Library of Free & Borrowable Books, Movies, Music & Wayback Machine,» 2021. [En línea]. Available: https://archive.org/. [Último acceso: 7 abril 2021].

[30] A. Chitipothu, «Open Library Books API,» 2020 septiembre 2020. [En línea]. Available: https://openlibrary.org/dev/docs/api/books. [Último acceso: 7 abril 2021].

[31] A. Chitipothu, «Open Library Covers API,» 3 mayo 2011. [En línea]. Available: https://openlibrary.org/dev/docs/api/covers. [Último acceso: 7 abril 2021].

[32] C. H. T. M. Anand Chitipothu, «Open Library Search API,» 12 marzo 2020. [En línea]. Available: https://openlibrary.org/dev/docs/api/search. [Último acceso: 7 abril 2021].

[33] Wikipedia: La enciclopedia libre, «ISBN,» 8 enero 2021. [En línea]. Available: https://es.wikipedia.org/wiki/ISBN. [Último acceso: 7 abril 2021].

[34] Microsoft, «Visual Studio Code - Code editing. Redefined,» 2021. [En línea]. Available: https://code.visualstudio.com/. [Último acceso: 13 abril 2021].

[35] A. Salvador, «BookStand,» 25 septiembre 2021. [En línea]. Available: https://github.com/aursalgut/BookStand-Release. [Último acceso: 13 abril 2021].

[36] OpenJS Foundation, «Introduction to NodeJS,» 2021. [En línea]. Available: https://nodejs.dev/learn. [Último acceso: 21 abril 2021].

[37] Google Developers, «Download Android Studio and SDK Developer Tools,» 2021. [En línea]. Available: https://developer.android.com/studio/. [Último acceso: 13 abril 2021].

[38] VMware, «VMware Workstation Player,» 2021. [En línea]. Available: https://www.vmware.com/es/products/workstation-player.html. [Último acceso: 21 abril 2021].

[39] Apple Inc, «Xcode 12 - Apple Developer,» 2021. [En línea]. Available: https://developer.apple.com/xcode/. [Último acceso: 21 abril 2021].

[40] Apple Inc, «Gestiona tu ID de Apple,» 2021. [En línea]. Available: https://appleid.apple.com/es/. [Último acceso: 21 abril 2021].

[41] K. Limpitsouni, «unDraw,» 2021. [En línea]. Available: https://undraw.co/. [Último acceso: 13 abril 2021].

[42] diagrams.net, «Diagram Software and Flowchar Maker,» 2021. [En línea]. Available: https://www.diagrams.net/. [Último acceso: 13 abril 2021].

[43] Visual Paradigm, «Ideal Modeling & Diagramming Tool for Agile Team Collaboration,» 2021. [En línea]. Available: https://www.visual-paradigm.com/. [Último acceso: 13 abril 2021].

[44] I. Román Martínez, *Transparencias de la asignatura "Ingeniería de Software". Introducción a UML y repaso de conceptos OO.,* Sevilla, 2018.

[45] Microsoft, «TypeScript: Documentation - Everyday Types,» 2021. [En línea]. Available: https://www.typescriptlang.org/docs/handbook/2/everyday-types.html#type-aliases. [Último acceso: 4 mayo 2021].

[46] Google, «Help protect against harmful apps with Google Play Protect,» 2021. [En línea]. Available: https://support.google.com/googleplay/answer/2812853?hl=en#. [Último acceso: 12 mayo 2021].

[47] React Native, «Setting up the development environment,» 2021. [En línea]. Available: https://reactnative.dev/docs/environment-setup. [Último acceso: 25 mayo 2021].

[48] J. Arvidsson, «Customizable Icons for React Native with support for image source and full styling.,» [En línea]. Available: react-native-vector-icons. [Último acceso: 26 mayo 2021].

[49] J. Trujillo, «React Native Local and Remote Notifications,» [En línea]. Available: https://github.com/zo0r/react-native-push-notification. [Último acceso: 26 mayo 2021].

[50] Facebook Open Source, «React Native Camera,» [En línea]. Available: https://react-native-camera.github.io/react-native-camera/. [Último acceso: 26 mayo 2021].

[51] H. Hübel, «Native filesystem access for react-native,» [En línea]. Available: https://github.com/itinance/react-native-fs. [Último acceso: 26 mayo 2021].

[52] Google, «Firebase Console,» 2021. [En línea]. Available: https://console.firebase.google.com/. [Último acceso: 25 mayo 2021].

[53] Redux, «Redux Fundamentals, Part 2: Concepts and Data Flow,» 2021. [En línea]. Available: https://redux.js.org/tutorials/fundamentals/part-2-concepts-data-flow. [Último acceso: 6 abril 2021].