

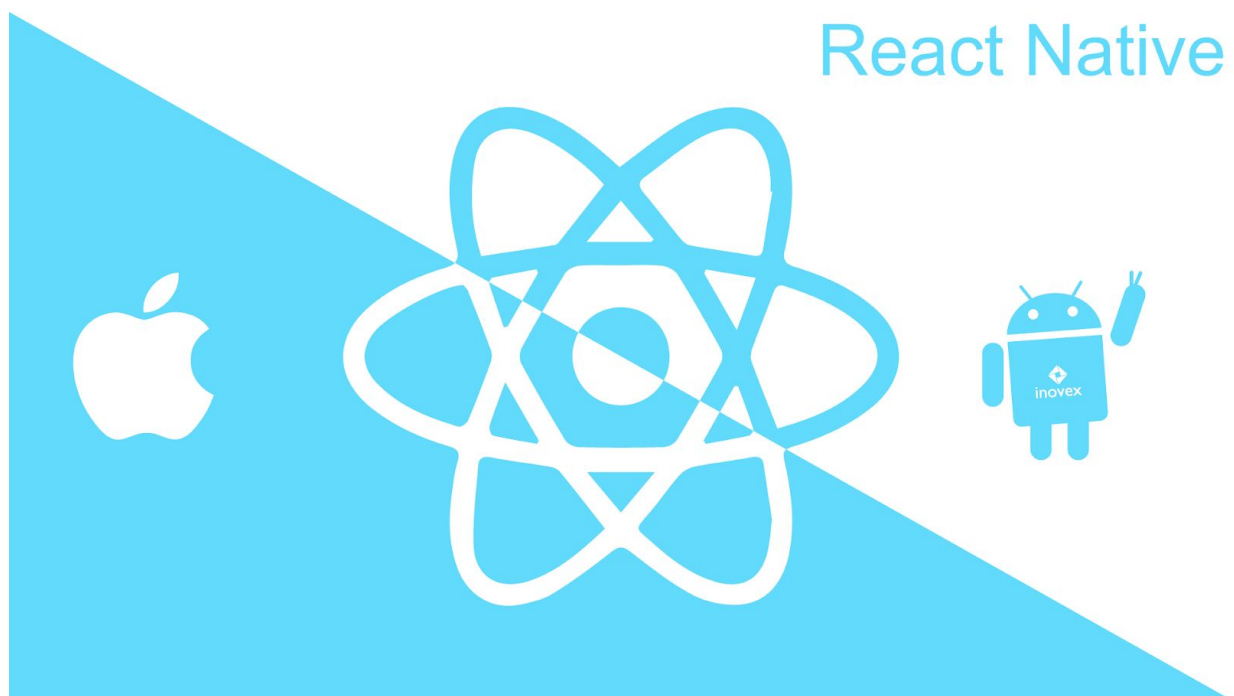
Práctica grupal.

React Native - Investigación y proyecto demo.

Sandra Carreras Masanet

Edgar Martínez Serrano

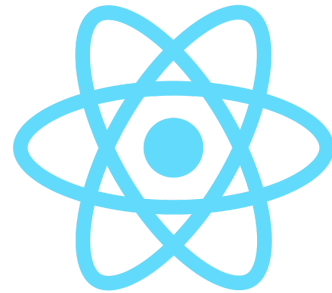
Gerard Soley López



¿Qué es React Native?	2
Alternativas a React Native	3
Flutter	3
Cordova	3
Xamarin	3
Ionic	4
Onsen UI	4
Quasar	4
¿Por qué usar React Native? Ventajas frente a otras alternativas	5
Facilidad de aprendizaje	5
Rendimiento y compatibilidad	5
Productividad del desarrollador	6
Comunidad	6
Coste	6
Estabilidad	6
Cualidades de React Native	7
Hooks	7
useState	7
useEffect	7
Redux	8
JSX	9
Virtual DOM	9
React Native vs código nativo	11
Nuestro proyecto	13
Introducción	13
Descripción de TopRestaurant	13
Vista de restaurantes	13
Vista Favoritos	16
Vista Top 5	16
Vista Buscar	17
Vista Cuenta	18
Navegación de la aplicación	19
Autenticación, Cloud Firestore y Storage	20
Hooks: useState y useEffect	22
MapView	22
Toast	25
Conclusiones del proyecto	26
Conclusiones finales sobre React Native	27
Referencias	28

¿Qué es React Native?

React Native es un framework JavaScript desarrollado por Facebook que sirve para crear aplicaciones reales nativas para multitud de plataformas. Este framework está basado en JavaScript React, pero a diferencia de React, React Native no está pensado para ser ejecutado en un navegador, sino que las aplicaciones desarrolladas se ejecutarán sobre las plataformas nativas. Como por ejemplo, Android o iOS. En vez de desarrollar, de hacer una aplicación híbrida o en HTML, obtenemos una aplicación nativa, que aparentemente no se puede distinguir de una aplicación realizada en Java o algún lenguaje de programación nativo.



En un principio, Facebook funcionaba mediante HTML5 en la versión móvil, esto hacía que la aplicación fuese demasiado inestable y recuperaba los datos de manera muy lenta. Entonces, Mark Zuckerberg, prometió desarrollar una mejor opción, de ahí surgió React.

React Native no utiliza HTML ni CSS, pero para interactuar dentro de la aplicación utiliza JavaScript para manipular las vistas. React Native también permite a los desarrolladores escribir código nativo en lenguajes como Java y Swift para Android e iOS respectivamente, lo que hace a react un framework flexible.

Alternativas a React Native

Flutter



Flutter es un framework de código abierto creado por Google y escrito en Dart. El código que genera es nativo para cada sistema operativo. Cuenta con Hot-Reload lo que permite ver a tiempo real en el emulador los cambios producidos en el código.

Cordova



Apache Cordova es un marco de desarrollo móvil de código abierto. Le permite utilizar tecnologías web como HTML5, CSS3 y JavaScript para el desarrollo multiplataforma. Las aplicaciones se ejecutan dentro de envoltorios dirigidos a cada plataforma y dependen de enlaces API que cumplen con los estándares para acceder a las capacidades de cada dispositivo, como sensores, datos, estado de la red, etc.

Xamarin



Xamarin es una plataforma de código abierto para compilar aplicaciones modernas y con mejor rendimiento para iOS, Android y Windows con .NET. Las aplicaciones de Xamarin se pueden escribir en PC o Mac, y compilar en paquetes de aplicación nativos, como un archivo .apk en Android o .ipa en iOS. Se ejecuta en entornos administrados y utiliza varias características como la asignación de memoria y la recolección de elementos no utilizados.

Ionic



Ionic es un framework gratuito y open source para desarrollar aplicaciones híbridas multiplataforma que utilizan HTML5, CSS (generado por SASS) y Cordova como base. Es uno de los framework del momento por utilizar AngularJS para gestionar las aplicaciones, lo que asegura aplicaciones rápidas y escalables.

Onsen UI



Onsen IU es un marco de código abierto que lo ayuda a crear aplicaciones híbridas con un rendimiento nativo. Permite a los desarrolladores crear aplicaciones móviles utilizando tecnologías web como CSS, HTML5 y JavaScript.

Quasar



Quasar es un marco de trabajo de código abierto basado en Vue.js, que le permite desarrollar rápidamente sitios web y aplicaciones, y desplegarlo en la Web como SPA, PWA, SSR, a una aplicación móvil, usando Cordova para iOS y Android, y a una aplicación de escritorio, usando Electron para Mac, Windows y Linux.

¿Por qué usar React Native? Ventajas frente a otras alternativas

Como hemos visto son muchas las alternativas disponibles, pero ahora vamos a hablar de las ventajas que presenta este framework y por qué elegirlo.

Facilidad de aprendizaje

Aprender los conceptos de React Native es fácil si somos desarrolladores de JavaScript con experiencia y tenemos algunas habilidades en React. Este framework es fácil de aprender y no requiere mucho esfuerzo, cuenta con una documentación oficial bastante muy explicativa, pero además es muy fácil encontrar tutoriales para iniciarse.

Rendimiento y compatibilidad

Con React Native podemos realizar una aplicación web progresiva. Este es un tipo de software que se entrega a través de web utilizando HTML, CSS y JavaScript. Está destinado para usar en cualquier plataforma.

A diferencia de los PWA, el código escrito en JavaScript es capaz de obtener los componentes visuales de forma nativa utilizando una capa que actúa en forma de puente 'React Native Bridge'. Esto lo que permite es trabajar con componentes nativos en un código JavaScript que conlleva un mejor rendimiento en la aplicación además de una mayor compatibilidad a la hora de utilizar recursos tales como efectos en los botones o propiedades específicas de componentes.

Además, se puede usar código fuente nativo en forma de módulos. Esto puede ser útil para desarrollar características especiales que no tienen soporte en React Native, por lo que se pueden llegar a desarrollar con un lenguaje nativo para la plataforma específica, y posteriormente importarlo en React Native en forma de módulo. También puede ser utilizado para mejorar los costes de rendimiento ya que en este caso el código ejecutado sí que sería 100% nativo.

Cordova o Ionic, por ejemplo, están basados en Angular por lo que finalmente obtenemos una aplicación en HTML y CSS, por lo tanto en rendimiento también es inferior.

Productividad del desarrollador

React Native nos brinda la flexibilidad de trabajar en cualquier editor de texto / IDE que deseemos. Se puede volver a cargar su aplicación sin volver a compilarla, lo que permite obtener más resultados en la mejora de la productividad del desarrollador. Además si queremos cambiar la plataforma, se puede reusar el 90% del código, mientras que en Flutter por ejemplo es entre el 50% y el 90%. En Xamarin se puede reusar el 96% del código pero únicamente se puede usar Visual Studio como IDE y requiere mucho más tiempo de compilación.

Comunidad

React Native tiene una gran popularidad y tiene una comunidad muy grande, lo usa mucha gente por lo que es muy fácil encontrar información, Además que podemos obtener ayuda en foros u otros sitios de forma más rápida. También, tiene APIs nativas listas para usarse de inmediato, así como muchas de las APIs que han sido desarrolladas por la propia comunidad. Flutter, por ejemplo, lleva menos tiempo en el mercado lo que provoca que su comunidad sea más pequeña aunque es verdad que está creciendo rápidamente.

Coste

Como ya hemos mencionado es un framework fácil de aprender y con una gran comunidad, esto hace que se reduzca el coste tanto económico como temporal para los desarrolladores nuevos.

También reduce la pérdida de tiempo al cargarse los cambios de la aplicación rápidamente sin necesidad de compilar. Y también como es multiplataforma esto ahorra costes de desarrollo.

Estabilidad

Este framework tiene mucha estabilidad, ya que está desarrollado Facebook que es una empresa con muchos recursos e importante. Esto desde el punto de vista empresarial, ofrece tranquilidad ya que te aseguras que este software tendrá un desarrollo estable.

Otras aplicaciones que usan React Native, como por ejemplo, Instagram, Discord, Netflix entre otras importantes. Que todas estas empresas grandes, trabajen con esta tecnología, como desarrolladores nos da confianza y estabilidad.

Cualidades de React Native

Hooks

Los hooks nos permiten usar el estado y características sin tener que crear una clase. De esta forma el código es más legible además que sigues un patrón de uso.

useState

En la definición de hooks, se ha mencionado que el código es más legible si usamos hooks, en este ejemplo se puede ver que declaramos un contador (count), con un método (setCount) y lo inicializamos a 0. Esto se hace con todos los hooks necesarios y siempre siguen el mismo patrón:

```
const [nombre, setNombre] = useState(inicializar)
```

De esta forma en una línea de código nos hemos ahorrado una clase además, todos los useState se recomienda inicializar al principio de la función para que así de un golpe de vista sabes los hooks de estado que hay y con qué valores se inicializan.

Una vez queramos modificar el estado, deberemos usar el setNombre y entre paréntesis pondremos el valor nuevo.

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

function Example() {
  // Declara una nueva variable de estado, la cual llamaremos "count"
  const [count, setCount] = useState(0);

  return (
    <div>
      <p>You clicked {count} times</p>
      <button onClick={() => setCount(count + 1)}>
        Click me
      </button>
    </div>
  );
}
```

useEffect

Este hook normalmente se usa para inicializar variables, llamadas a APIs.

Los useEffect, se puede usar para ejecutar cuando la función se renderiza por primera vez, o si se le pone un parámetro, cuando este parámetro se ejecutará.

Como se puede ver en el fragmento de código, actualiza el título del documento mediante un useEffect.


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

function Example() {
  const [count, setCount] = useState(0);

  // De forma similar a componentDidMount y componentDidUpdate
  useEffect(() => {
    // Actualiza el título del documento usando la API del navegador
    document.title = `You clicked ${count} times`;
  });

  return (
    <div>
      <p>You clicked {count} times</p>
      <button onClick={() => setCount(count + 1)}>
        Click me
      </button>
    </div>
  );
}
```

Redux

Igual que en React, podemos hacer uso de Redux. Redux es una herramienta para la gestión de estado en aplicaciones desarrolladas en Javascript.

El estado de una aplicación o de un componente se puede definir como conjunto de valores almacenados mediante propiedades o variables en un momento determinado de la ejecución que representan la situación actual de la aplicación o componente.

El estado puede incluir tanto respuestas del servidor como datos generados en local que son independientes del servidor.

El peligro que conlleva una aplicación de gran envergadura es juntar la mutabilidad de los datos con el asincronismo, es decir, una incertidumbre de cuando estos datos van a ser modificados.

Por tanto, Redux tiene como propósito hacer que estos cambios de estados sean predecibles, es decir, gestionar el estado de la aplicación. Para ello, Redux impone restricciones a la hora de cómo y cuándo producir la actualización de los datos.

Por otro lado, usar Redux se convierte en una buena práctica debido a que el proyecto mejora a nivel organizativo.

Entre los principios fundamentales que tiene Redux se encuentran que es una “fuente única de verdad”, es decir, un único objeto almacena el estado para toda la aplicación entera; trata la inmutabilidad, el objeto de estado es ‘read-only’ y no puede ser cambiado sin emitir

una acción de Redux que lo permita; y trata funciones puras, que es básicamente la definición del cambio de un estado a través de acciones (estas funciones son conocidas como reducers).

JSX

Es una extensión de JavaScript para poder desarrollar la interfaz de usuario. Aparentemente parece un lenguaje de plantillas, como podría ser HTML, pero funciona con JavaScript. La finalidad de React Native es unir la lógica de renderizado con la unidad lógica de interfaz de usuario. React Native al igual que React, junta estas dos lógicas en componentes.

Aunque los desarrolladores de React no quieren que se desarrollen las aplicaciones de esta forma, a muchos programadores les gusta usar etiquetas para desarrollar, por tanto se les ofrece esta opción.

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

ReactDOM.render(
  element,
  document.getElementById('root')
);
```

Virtual DOM

Antes que nada, DOM viene de “Document Object Model”. Esta es la interfaz del usuario en la aplicación. Cada vez que hay un estado en la interfaz, el DOM se debe de actualizar para representar el cambio. Ahora el problema es que modificar el DOM afecta de forma significativa al rendimiento. Por esto existe el virtual DOM.

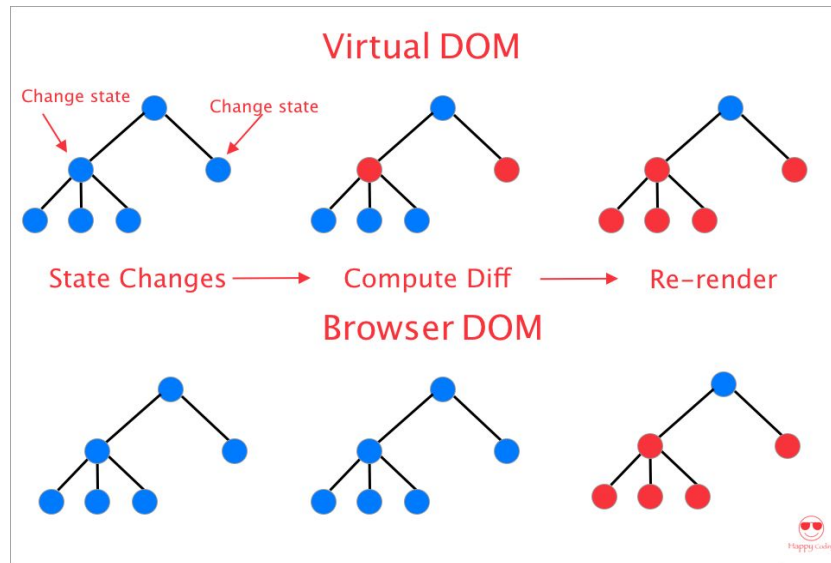
El virtual DOM funciona significativamente mejor que el DOM. El virtual DOM es una representación virtual del DOM, donde cada vez que cambia el estado de nuestra aplicación, el DOM virtual se actualiza.

Cuando agregamos elementos nuevos a la interfaz de usuario, se crea un DOM virtual que se representa con un árbol. Cada elemento es un nodo en el árbol. Si el estado de alguno de los elementos cambia, entonces se crea un nuevo árbol DOM virtual. Este se compara con el árbol virtual DOM anterior.

Una vez hecho esto, el virtual DOM calcula el mejor método posible para realizar estos cambios en el DOM. Esto asegura que haya operaciones mínimas en el DOM. Por tanto, el coste de rendimiento para actualizar el DOM es menor.

Los nodos rojos, son los nodos que se han modificado. Estos, representan los elementos de la interfaz que se han modificado. Tras esto, se calcula la diferencia entre la versión del DOM anterior y la actual. Después, el árbol principal se vuelve a representar para

proporcionar la interfaz de usuario actualizada. Finalmente, el DOM se actualiza con el árbol actualizado.



React Native vs código nativo

Como se ha mencionado ya anteriormente, React Native tiene una parte nativa en las aplicaciones debido a que gracias al uso de React Native Bridge los componentes que se utilizan son recogidos de forma nativa para la plataforma en la que se ejecuta.

Gracias a esto, al utilizar layouts nativos el resultado final del diseño de las interfaces no se diferencia respecto a una aplicación nativa, ya que los componentes son los propios de la plataforma.

De esta forma, se puede decir que React Native utiliza componentes visuales nativos mientras que la lógica de la aplicación se mantiene compartida para ambas plataformas, ejecutándose de forma no nativa en JavaScript. Y es por esto por lo que no se puede decir que React Native sea realmente una aplicación nativa, pues utiliza código no nativo que es compartido para distintas plataformas donde se desarrolla toda la lógica de la aplicación, y solo en el punto donde se obtienen elementos, componentes visuales, se obtienen de forma nativa a través de React Native Bridge.

Por una parte, el hecho de que los componentes se puedan tratar a nivel nativo, repercute en un mejor rendimiento de la aplicación que si estos no lo fueran, además de permitir un gran nivel de personalización, pues se pueden utilizar muchas más propiedades y efectos que en una aplicación típica de PWA no se podrían conseguir.

Sin embargo, no todo el funcionamiento nativo está soportado en React Native, puede haber casos concretos en los que un efecto o comportamiento deseado no se pueda representar con React Native debido a que hay características de los dispositivos que pueden no estar soportadas todavía. De nuevo, esta es una gran diferencia entre React Native y las posibilidades que traen un lenguaje realmente nativo, no todo se puede llegar a representar o conseguir.

No obstante, React Native ofrece una alternativa para conseguir de alguna forma hacer uso de esas características que solo pueden ser usadas con código nativo: los módulos.

El uso de módulos permite cargar en el proyecto código nativo de Android y código nativo de iOS, de forma que pueda ser usado en la aplicación con referencias. Este código que se ejecute sí será 100% nativos, con todas las ventajas que esto conlleva. Un desarrollador puede optar por este camino debido a la intención de usar características especiales de cierta plataforma donde no tiene alternativa en React Native, o incluso para mejorar la eficiencia y rendimiento, pues, al tratarse de código nativo es posible desarrollar de forma más óptima.

Sin embargo, esto es una alternativa para casos puntuales, ya que no sería lógico trabajar el 100% del código así, para eso sería más lógico desarrollar el código en una

plataforma en concreto. Trabajar así hace que perdamos todas las ventajas que nos aporta React Native en cuanto a eficiencia de trabajo y multiplataforma, por eso solo debe ser utilizado cuando no hay alternativa en React Native.

Nuestro proyecto

Introducción

Para esta práctica hemos desarrollado una aplicación en React Native. Nuestra aplicación se llama TopRestaurante. Esta, es una demo parecida a TripAdvisor, donde los usuarios pueden darse de alta en nuestra aplicación y subir restaurantes además de poder valorarlos. También, hay un top 5 de restaurantes, una vista de los restaurantes favoritos del cliente e incluso se pueden buscar restaurantes.

Cabe decir que es una demo, y hay apartados que no se han llegado a implementar, pero se han simulado. En esta documentación se explicará con detalle el funcionamiento de la aplicación y también mostraremos el código.

Descripción de TopRestaurant

La aplicación se divide en 5 vistas: una lista de restaurantes, la vista de favoritos, un Top 5, un buscador y la cuenta.

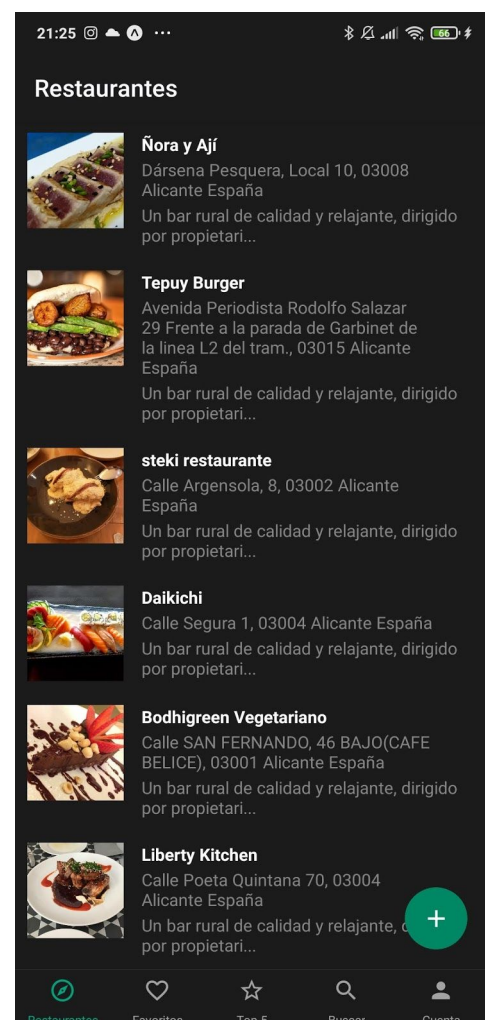
Vista de restaurantes

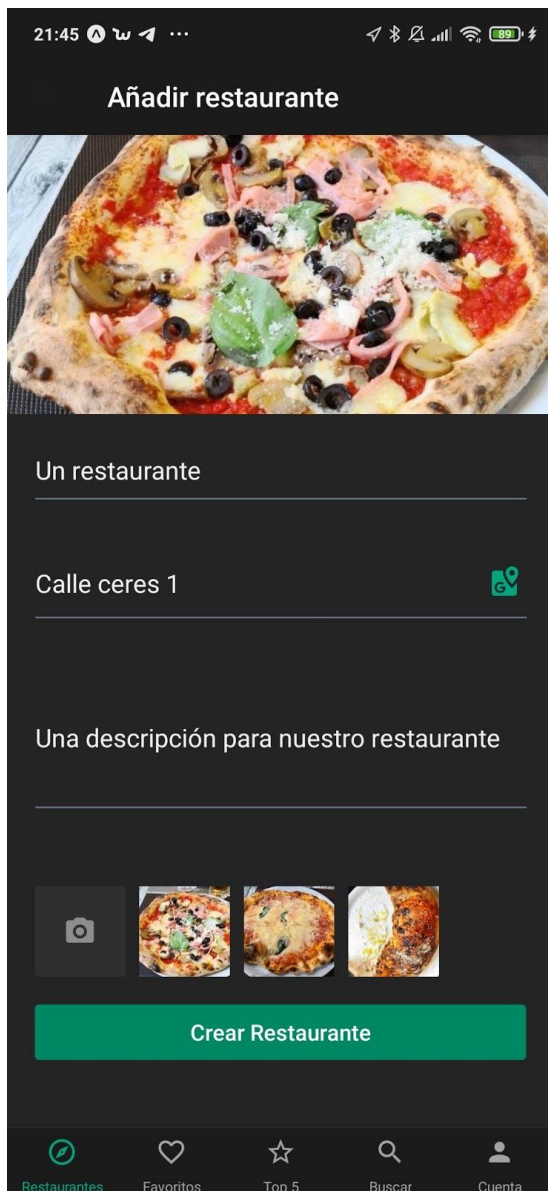
En esta vista, podemos observar un listado de todos los restaurantes de nuestra aplicación. Este listado es paginado, es decir, no se cargan todos los restaurantes, si no, que se cargan de 7 en 7. De esta forma, el dispositivo es capaz de renderizar los restaurantes sin tener problemas de memoria, además que es más eficiente a la hora de transmitir la información.

Para poder crear restaurantes, deberemos de estar registrados. Si no estás registrado, no podremos ver el botón de añadir restaurantes.

En esta vista, también se muestra un botón abajo a la derecha con el símbolo de '+'. Si pulsamos este botón nos redirigirá a una vista donde podremos añadir un restaurante nuevo.

La vista que nos aparecerá es la siguiente imagen.





En esta vista, vemos los datos que tiene un restaurante. Podemos observar que tiene una fotografía de portada, una descripción, una dirección, una ubicación y fotos.

La portada del restaurante, es la primera foto de la lista de fotos seleccionadas.

La máxima cantidad de fotos que se pueden seleccionar para el restaurante son cinco. Si pinchamos en el botón de ubicación, nos aparecerá un Modal con el mapa y nuestra ubicación. Antes nos pedirá permisos para poder usar el GPS.

Una vez que se crea el restaurante nos redirigirá al listado de restaurantes donde podremos ver todos los restaurantes y el que hemos creado se posiciona el primero.

Si pinchamos en un restaurante creado, podremos acceder a la vista restaurante, donde nos aparecerán los datos de este.

En esta ventana nos aparecerá el nombre del restaurante con la portada del restaurante, la descripción del restaurante, la ubicación, y la calle.

Además de esto aparece el número de teléfono, el correo y la calificación. Cabe decir que el número de teléfono no se puede introducir por el usuario ni el correo. Esto es una implementación que se ha quedado a mitad.

También, la calificación se muestra correctamente, solo que los usuarios no pueden calificar. Es decir, está almacenado que este restaurante tiene una puntuación de 4,1 estrella y se muestra porcentualmente con las estrellas.


Aquí, nos ha faltado incluir el botón de favoritos para asignar al usuario este restaurante.

Si pinchamos en el mapa, se nos abrirá el google maps con la posición del restaurante.

The screenshot shows a mobile application interface for a restaurant. At the top, the status bar displays the time 21:57, signal strength, and battery level at 96%. The app header features the restaurant's name, "Sale & Pepe Pizzeria - Barrio", in white text on a dark background. Below the header is a large, vibrant image of a pizza topped with pepperoni, fresh basil, and a side of bread and vegetables. Underneath the image, the restaurant's name is repeated, followed by a 4.1-star rating represented by five yellow stars. A descriptive paragraph follows, detailing the restaurant's location in Berkshire de Peasemore and its proximity to the A34 and M4 roads. The section titled "Informacion sobre el restaurante" contains a map snippet showing the location in Alicante (Alacant), Spain, with a red pin marking the spot. Below the map, the address "Calle Muñoz, 3, 03002 Alicante España" is listed, along with the phone number "689 61 30 51" and the email address "Sale & Pepe Pizzeria - Barrio@gmail.com". At the bottom of the screen is a navigation bar with five icons: a fork and knife for "Restaurantes", a heart for "Favoritos", a star for "Top 5", a magnifying glass for "Buscar", and a person icon for "Cuenta".

21:57 72 A ... 96%

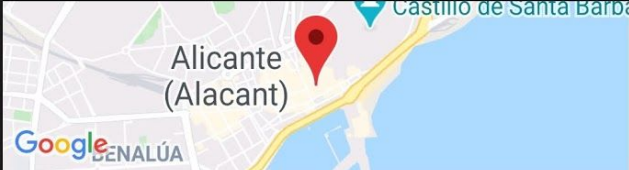
Sale & Pepe Pizzeria - Barrio



Sale & Pepe Pizzeria - Barrio ★★★★★

Un bar rural de calidad y relajante, dirigido por propietarios galardonados, en el hermoso pueblo de Berkshire de Peasemore, a poca distancia en coche de la A34 y de los cruces 13 y 14 de la M4. El edificio y la decoración son encantadores y rústicos, con elegantes toques modernos.

Informacion sobre el restaurante



Alicante (Alacant)

Google Maps

Castillo de Santa Bárbara

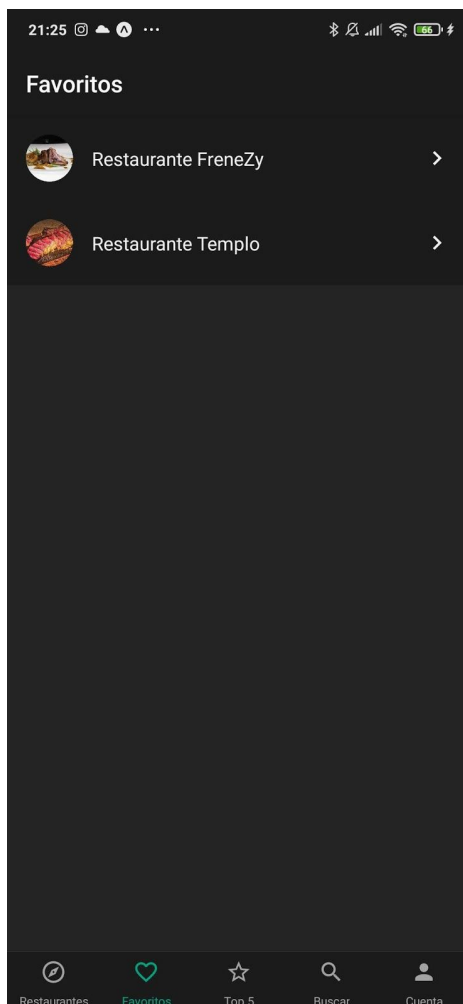
📍 Calle Muñoz, 3, 03002 Alicante España

☎ 689 61 30 51

@ Sale & Pepe Pizzeria - Barrio@gmail.com

Restaurantes Favoritos Top 5 Buscar Cuenta

Vista Favoritos



En esta vista, simplemente es un listado de los restaurantes favoritos que tendría un usuario. Como hemos explicado anteriormente, no se ha llegado a implementar la funcionalidad de favoritos.

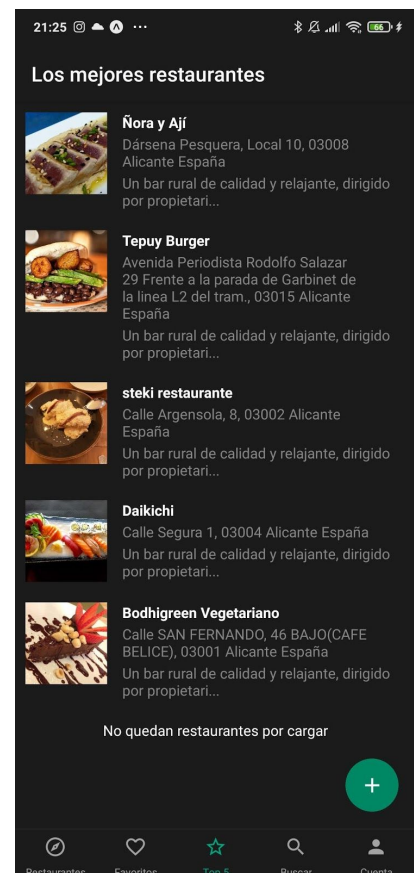
La lista de restaurantes se ha realizado mediante un FlatList.

Si pulsamos algún restaurante, nos dirigirá a la vista de este restaurante.

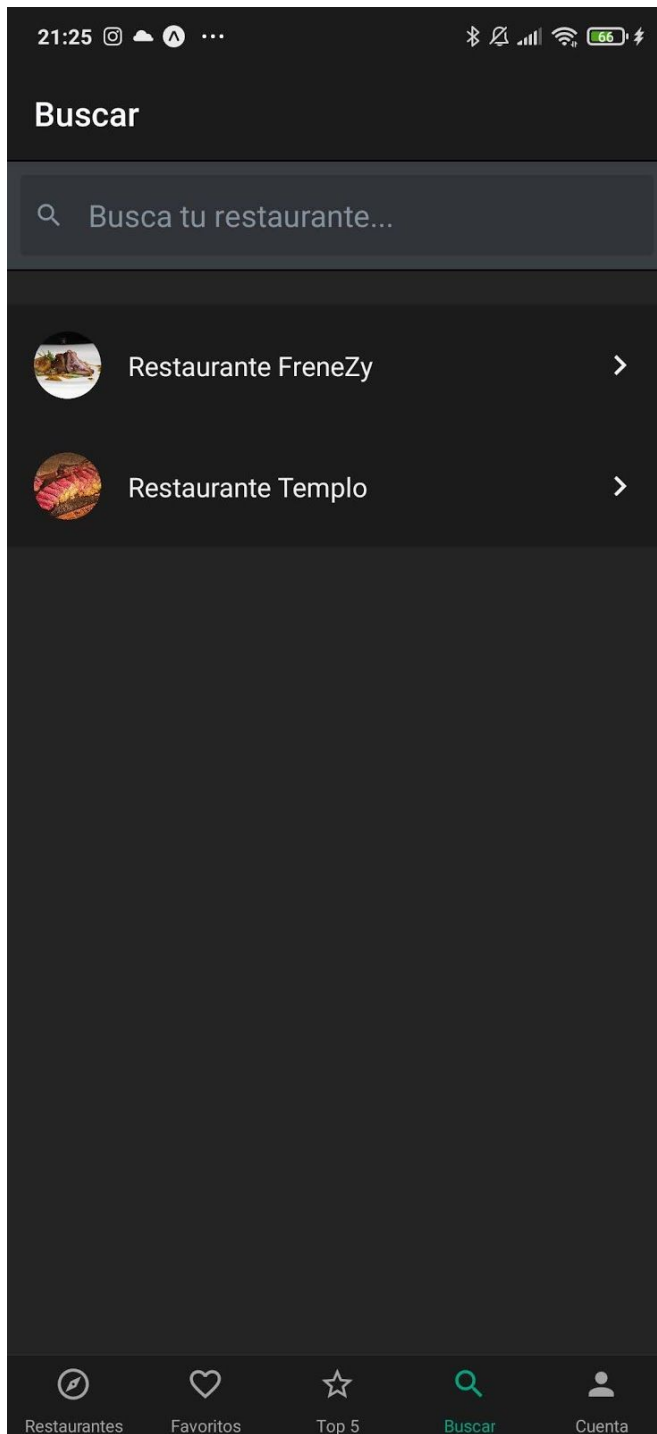
Vista Top 5

En esta vista, se ve un listado de los 5 mejores restaurantes. Esta vista tampoco la hemos llegado a implementar del todo. Simplemente, es un listado de 5 restaurantes. Nuestro objetivo era poder listar por puntuación, pero no lo hemos llegado a terminar.

Si pulsamos en algún restaurante, nos llevará a la vista del restaurante.



Vista Buscar

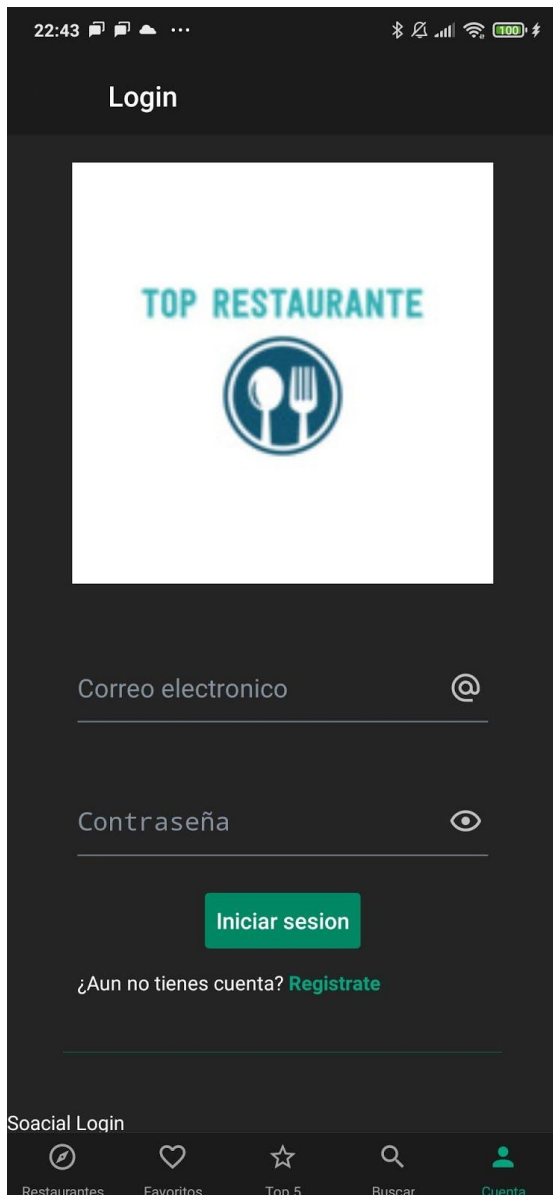


En esta vista, podemos ver que tenemos un buscador. Arriba escribimos el nombre del restaurante que queremos buscar y se nos listan.

Esta vista está completamente desarrollada. Y es funcional.

Si pulsamos, en alguno de los restaurantes, podemos observar que se nos abrirá ese restaurante.

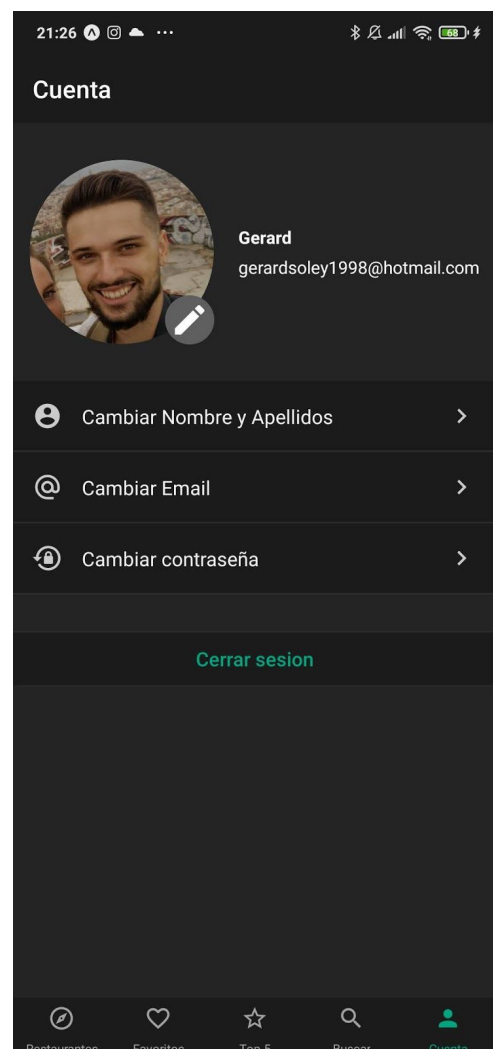
Vista Cuenta



En esta vista pueden ocurrir dos cosas. La primera es que no estamos logueados, entonces lo que sucede, es que nos pedirá login. Si estamos dados de alta, entonces solo nos hará falta iniciar sesión. Por otro lado, si no estamos registrados, nos podemos registrar en la aplicación.

Para registrarnos en la aplicación, deberemos pulsar en [Registrate](#) y nos redirigirá a la vista de Registro.

Una vez que tengamos nuestra cuenta, entonces no lograremos. Tras hacer login nos redirigirá a la vista de nuestra cuenta que es la siguiente que se muestra.



Los datos que nos aparecen en la vista de cuenta, se pueden modificar. Como se muestra en la imagen, tenemos una foto de usuario, y se puede cambiar.

Si pulsamos, por ejemplo. “Cambiar Nombre y Apellidos” entonces se nos abrirá un Model donde podremos modificar nuestro nombre. Esto se puede hacer igual con el resto.

También, se puede ver que está el botón de cerrar sesión.

Navegación de la aplicación

```
const Tab = createBottomTabNavigator();

export default function Navigation() {
  return (
    <NavigationContainer>
      <Tab.Navigator
        initialRouteName="restaurants"
        tabBarOptions={{
          inactiveTintColor: "#646464",
          activeTintColor: "#00a680"
        }}
        screenOptions={({route}) => ({
          tabBarIcon: ({color}) => screenOptions(route, color),
        })
      >
        <Tab.Screen
          name="restaurants"
          component={RestaurantsStack}
          options={{title: "Restaurantes"}}/>
        <Tab.Screen
          name="favorites"
          component={FavoritesStack}
          options={{title: "Favoritos"}}/>
        <Tab.Screen
          name="top-restaurants"
          component={TopRestaurantsStack}
          options={{title: "Top 5"}}/>
        <Tab.Screen
          name="search"
          component={SearchStack}
          options={{title: "Buscar"}}/>
        <Tab.Screen
          name="account"
          component={AccountStack}
          options={{title: "Cuenta"}}/>
      </Tab.Navigator>
    </NavigationContainer>
  )
}
```

Desde App.js hacemos un return de Navigations.js.

Esta imagen es un fragmento de código de Navigation.js. Donde se puede ver cómo se estructura la navegación.

Como se muestra arriba, se puede observar que se crea una barra navegación en el Bottom. Luego creamos las distintas screen que tendrá la navegación y asignamos los componentes a cada screen. Cada Screen tendrá un nombre y un título.

Dentro de cada Tab, tenemos el Stack de cada vista. Por ejemplo, si entramos dentro del componente de AccountStack, podemos observar lo siguiente.

Como se muestra en la imagen, tenemos tres stacks distintos: Account, Login, y Register. Una vez estamos dentro de Tab de Account, si nos movemos entre estas Stacks nos aparecerá todo el rato abajo en la barra de navegación que estamos visualizando dentro de Account.

Dentro de cada componente tenemos un return que devuelve la vista de cada vista. Por ejemplo, en Login tenemos la vista de este mismo además del código que gestiona la vista.

```
export default function AccountStack () {
  return (
    <Stack.Navigator>
      <Stack.Screen
        name = "account"
        component = {Account}
        options = {{title:"Cuenta"}}
      />
      <Stack.Screen
        name = "login"
        component = {Login}
        options = {{title:"Login"}}
      />
      <Stack.Screen
        name = "register"
        component = {Register}
        options = {{title:"Register"}}
      />
    </Stack.Navigator>
  );
}
```

Autenticación, Cloud Firestore y Storage

Hemos utilizado firebase para todo el tema de backend. Ya que de esta forma nos ha facilitado el desarrollo de la aplicación. En storage se almacenan las imágenes y los recursos, en el Cloud Firestore se almacenan todos los datos de la aplicación, como sería en una base de datos. Finalmente, en Authentication, se encuentran los datos de los usuarios para poder loguearse.

```
const onSubmit = () => {
  if(isEmpty(formData.email) || isEmpty(formData.password)){
    toastRef.current.show("Todos los campos son obligatorios");
  } else if (!validateEmail(formData.email)) {
    toastRef.current.show("El email no es correcto")
  } else {
    setLoading(true);
    firebase.auth().signInWithEmailAndPassword(formData.email,formData.password)
      .then(()=>{
        setLoading(false);
        navigation.navigate("account");
      })
      .catch (()=> {
        setLoading(false);
        toastRef.current.show("Email o contraseña incorrecta")
      })
  }
}
```

Como ejemplo de Authentication, tenemos el login. Como se muestra en la imagen, el login se realiza con firebase. Se le pasa el email y la contraseña.

```
const updatePhotoUrl = () => {
  firebase
    .storage()
    .ref(`avatar/${uid}`)
    .getDownloadURL()
    .then( async(response) => {
      const update = {
        photoURL: response
      };
      await firebase.auth().currentUser.updateProfile(update);
      setLoading(false);
    })
    .catch(() => {
      toastRef.current.show("Error al actualizar el avatar")
      setLoading(true);
    })
  });
};
```

Como ejemplo de Storage, podemos observar la siguiente foto. Aquí se puede observar, que se actualiza la imagen del usuario. Lo que se hace es obtener la referencia del usuario y luego se almacena en Storage con ese nombre.

```

useFocusEffect (
  useCallback(() => {
    db.collection("restaurants").get().then((snap) => {
      setTotalRestaurants(snap.size)
    })

    const resultRestaurants = [];

    //console.log(totalRestaurants);
    db.collection("restaurants")
      .orderBy("createAt", "desc")
      .limit(limitRestaurants).get()
      .then((response) => {
        setStartRestaurants(response.docs[response.docs.length - 1]);
        response.forEach((doc) => {
          const restaurant = doc.data();
          restaurant.id = doc.id;
          resultRestaurants.push(restaurant)
        });
        setRestaurants(resultRestaurants);
      }).catch(() => {
        console.log("error");
      })
    }, [])
  )
)

```

Finalmente, podemos observar esta imagen como ejemplo Firestore donde obtenemos el listado de restaurantes de forma paginada. Hay que aclarar que:

db = firebase.firestore(app)

Hooks: useState y useEffect

Hemos utilizado los hooks para simplificar el desarrollo de la aplicación. Ya que te ayuda a simplificar el código y lo hace más sencillo. Por ejemplo, si tuviéramos que crear un clases y métodos para cada una de las clases, el código se volvería más engorroso, para esto mismo sirven los useState. Para simplificar las clases. Son clases genéricas que te ayudan a ahorrar código y que sea más legible.

Los useState se declaran e inicializan, y luego más adelante, cuando tienes que actualizar el valor llamas al set y se actualiza. Como se puede observar en la imagen inferior, primero hemos declarado restaurants. Luego dentro del useEffect hemos utilizado el set: **setRestaurants(resultRestaurants);**

```
export default function Restaurants(props) {
  const {navigation} = props;
  const [user, setUser] = useState(null)
  const [restaurants, setRestaurants] = useState([])
  const [totalRestaurants, setTotalRestaurants] = useState(0)
  const [startRestaurants, setStartRestaurants] = useState(null)
  const [isLoading, setIsLoading] = useState(false);
  const limitRestaurants = 10;

  useEffect(() => {
    useCallbac(() => {
      db.collection("restaurants").get().then((snap) => {
        setTotalRestaurants(snap.size)
      })

      const resultRestaurants = [];

      //console.log(totalRestaurants);
      db.collection("restaurants")
        .orderBy("createAt", "desc")
        .limit(limitRestaurants).get()
        .then((response) => {
          setStartRestaurants(response.docs[response.docs.length - 1]);
          response.forEach((doc) => {
            const restaurant = doc.data();
            restaurant.id = doc.id;
            resultRestaurants.push(restaurant)
          });
          setRestaurants(resultRestaurants);
        }).catch(() => {
          console.log("error");
        })
      }, [])
    })
  }, [])
}
```

Cuando queremos realizar una acción, en un momento concreto, como por ejemplo cuando se actualiza un dato debemos usar los useEffect. Aquí lo que se hace es actualizar los datos de usuario cuando se modifica reloadUserInfo.

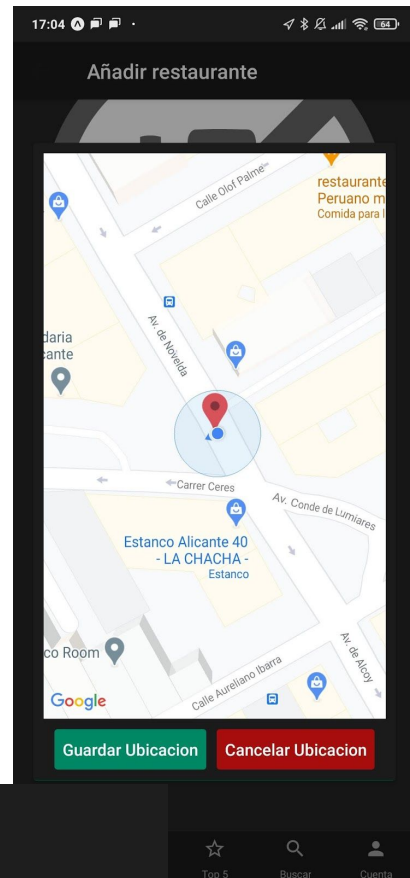
```
useEffect(() => {
  (async () => {
    const user = await firebase.auth().currentUser;
    setUserInfo(user);
  })()
  setReloadUserInfo(false);
}, [reloadUserInfo]);
```

MapView

En la aplicación, tenemos implementado un mapa para localizar el restaurante. El mapa aparece con un Modal y gestionamos que se vea con un `useState` que cuando queremos que se vea, lo ponemos a `true`.

Tenemos una función que se llama `Map`, que se encarga de encontrar la ubicación del terminal además de mostrar la posición en el mapa con un `marker`.

Como se muestra en la imagen de abajo, tenemos la función `Map`. Esta, tiene un `useEffect`, que al ejecutarse `Map()` obtenemos la ubicación del dispositivo. Si no existen permisos por parte del usuario, entonces se mostrará un `toast` diciendo que tenemos que aceptar los permisos de ubicación. En el momento que tengamos los permisos, entonces obtendremos la ubicación.



```
function Map(props) {
  const {isVisibleMap, setIsVisibleMap, setLocationRestaurant, toastRef} = props;
  const [location, setLocation] = useState(null)

  useEffect(() => {
    (async () => {
      const resultPermissions = await Permissions.askAsync(Permissions.LOCATION);
      const statusPermissions = resultPermissions.permissions.location.status;

      if(statusPermissions !== "granted"){
        toastRef.current.show("Tienes que aceptar los permisos de localizacion para crear un restaurante", 5000);
      } else {
        const loc = await Location.getCurrentPositionAsync({});
        console.log(loc);
        setLocation({
          latitude: loc.coords.latitude,
          longitude: loc.coords.longitude,
          latitudeDelta: 0.001,
          longitudeDelta: 0.001
        })
      }
    })()
  }, [])
}
```

A la derecha podemos observar el `return` del `Map`. Como se muestra, hay un `modal` con un `MapView`. Especificamos la localización al `MapView`, para que se ajuste el mapa a nuestra ubicación y luego dentro, tenemos un `Marker` que indica la posición exacta con una chincheta en el mapa.

Más abajo, observamos dos botones, que son para guardar los datos, o para cancelar la operación de obtener la ubicación.

```
const confirmLocation = () => {
  setLocationRestaurant(location);
  toastRef.current.show("Localizacion guardada correctamente")
  setIsVisibleMap(false)
}

return (
  <Modal isVisible={isVisibleMap} setIsVisible={setIsVisibleMap}>
    <View>
      {location && (
        <MapView>
          style={styles.mapStyles}
          initialRegion={location}
          showsUserLocation={true}
          onRegionChange={(region) => setLocation(region)}
        </MapView>
        <MapView.Marker>
          coordinate={{
            latitude: location.latitude,
            longitude: location.longitude
          }}
          draggable
        </MapView.Marker>
      )}
    </View>
    <View style={styles.viewMapBtn}>
      <Button>
        title="Guardar Ubicacion"
        containerStyle={styles.viewMapBtnContainerSave}
        buttonStyle={styles.viewMapBtnSave}
        onPress={confirmLocation}
      </Button>
      <Button>
        title="Cancelar Ubicacion"
        containerStyle={styles.viewMapBtnContainerCancel}
        buttonStyle={styles.viewMapBtnCancel}
        onPress={()=>setIsVisibleMap(false)}
      </Button>
    </View>
  </Modal>
)
```



```

export default function Map(props) {
  const {location, name, height} = props;

  const openAppMap = () => {
    const {latitude, longitude} = location;
    openMap({
      latitude: latitude,
      longitude: longitude,
      zoom: 19,
      query: name
    })
    console.log(latitude)
    console.log(longitude)
  }

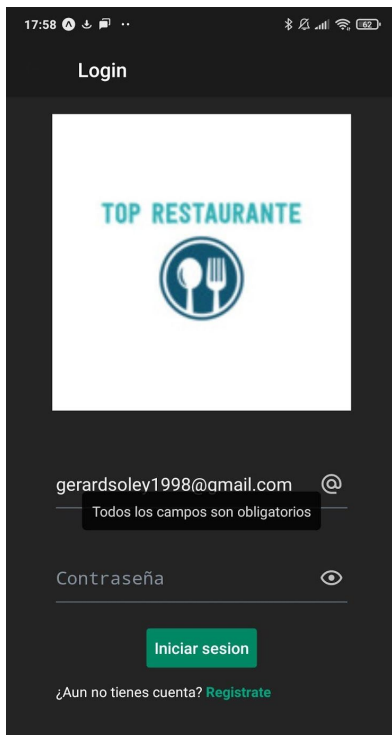
  return (
    <MapView
      style={{height:height, width: "100%"}}
      initialRegion={location}
      onPress={openAppMap}
    >
      <MapView.Marker
        coordinate={{
          latitude: location.latitude,
          longitude: location.longitude
        }}
      />
    </MapView>
  )
}

```

Para mostrar el mapa en la vista del restaurante, lo que hacemos es llamar a la función Map (que se muestra en la imagen de la izquierda).

A continuación, pasamos los parámetros de localización del restaurante, y ponemos el zoom que queremos que se muestre el mapa. Además, le añadimos un marker.

Toast



Para mostrar mensajes emergentes hemos utilizado toast.

Como ejemplo, vamos a ver el Toast que hemos puesto en Login. Como se ve, es obligatorio tener el campo del usuario y el de la contraseña, por tanto hemos añadido un toast a la vista de Login.

Como se ve en el código de abajo, tenemos un View compuesto por LoginForm y CreateAccount. En LoginForm, comprobaremos que los campos están rellenos. Si no es así recuperaremos el toast y lo mostraremos con el mensaje indicado.

En la última imagen de esta página hay un fragmento de código que pertenece a LoginForm. Aquí podemos ver que tenemos varios toast, pero el que se ha lanzado es el primero, donde comprobamos que todos los campos están rellenos.

```
export default function Login () {
  const toastRef = useRef();

  return (
    <ScrollView>
      <Image
        source={require("../assets/img/logo_size.jpg")}
        resizeMode="contain"
        style={styles.logo}
      />
      <View style={styles.viewContainer}>
        <LoginForm toastRef={toastRef}/>
        <CreateAccount/>
      </View>
      <Divider style={styles.divider}/>
      <Text>Social Login</Text>
      <Toast ref={toastRef} position="center" opacity={0.9}/>
    </ScrollView>
  );
}

const onSubmit = () => {
  if(isEmpty(formData.email) || isEmpty(formData.password)){
    toastRef.current.show("Todos los campos son obligatorios");
  } else if (!validateEmail(formData.email)) {
    toastRef.current.show("El email no es correcto")
  } else {
    setLoading(true);
    firebase.auth().signInWithEmailAndPassword(formData.email, formData.password)
      .then(()=>{
        setLoading(false);
        navigation.navigate("account");
      })
      .catch (()=> {
        setLoading(false);
        toastRef.current.show("Email o contraseña incorrecta")
      })
  }
}
```

Conclusiones del proyecto

A todos los integrantes del proyecto nos ha gustado este framework por su agilidad a la hora de desarrollar, ya que partimos de Xamarin y cada vez que quieres ver el resultado de una vista, estas obligado a compilar, y estas compilaciones son bastante costosas. En cambio con React Native, utilizando Expo Cli se genera al instante la modificación y de manera automática al guardar el fichero.

Cabe destacar, que hemos encontrado muchos recursos acerca de React Native, esto nos ha hecho que aprender acerca de este entorno de trabajo sea mucho más sencillo de lo que esperábamos. La propia documentación que nos ofrece su página web es muy buena.

Además, React Native facilita mucho el desarrollo del front end, ya que no tienes porque tocar html como en PWAs. Gracias a esto, nos ha facilitado el desarrollo de las vistas sin tener que estar perdiendo tiempo en ir declarando plantillas html. Simplemente, en el mismo return de la vista, solo debes añadir botones u objetos envueltos.

Para concluir, recomendamos usar este entorno de trabajo para desarrollar aplicaciones multiplataforma ya que también hay una gran cantidad de APIs que pueden proporcionar una gran funcionalidad.

Conclusiones finales sobre React Native

Como hemos visto en el desarrollo de la práctica, podemos decir que consideramos React Native una de las mejores opciones para desarrollar aplicaciones. Pensando desde un punto de vista empresarial y valorando el resto de opciones, React Native es la que mejor cumple con los requisitos de marketing, es decir, que sea flexible, rentable de producir y además teniendo la seguridad y tranquilidad en que el framework va a ser duradero y mantenido por muchos tiempo, ya que tiene a Facebook como creador y muchas otras empresas grandes en el sector tecnológico que utilizan esta tecnología.

Como hemos explicado anteriormente en sus ventajas, este framework tiene una flexibilidad muy alta, ya que podemos hacerla más o menos nativa según los requisitos de nuestro proyecto. Es decir, si necesitamos acceder mucho a los recursos del dispositivo, quizá nos interesará realizar la aplicación más nativa, acercándonos más al lenguaje propio del sistema. Si por lo contrario, queremos realizar una aplicación multiplataforma, con costes bajos, podremos desarrollarla de forma completa con JavaScript sin tener ningún problema.

También, como se usa JavaScript, si en un futuro fuese necesario migrar a otro framework, nos aseguramos que podemos reusar un porcentaje elevado de nuestro código.

Cabe destacar que, desde el punto de vista del desarrollador, es bastante ágil desarrollar en este entorno ya que no tienes que estar de manera tediosa compilando el proyecto cada vez.

Finalmente, podemos concluir que desde el punto de vista del desarrollador y desde una empresa, React Native es muy buena opción tanto por los costes de desarrollo como la facilidad de uso.

Referencias

React Native

<https://openwebinars.net/blog/react-native-que-es-para-que-sirve/>

Flutter

<https://www.itdo.com/blog/flutter-el-nuevo-framework-de-google/>

Apache Cordova

<https://cordova.apache.org/docs/en/latest/guide/overview/index.html>

Xamarin

<https://docs.microsoft.com/es-es/xamarin/get-started/what-is-xamarin>

Ionic

[https://en.wikipedia.org/wiki/Ionic_\(mobile_app_framework\)](https://en.wikipedia.org/wiki/Ionic_(mobile_app_framework))

Onsen UI

<https://onsen.io/v2/guide/>

Quasar

<https://quasar.dev/introduction-to-quasar>

Ventajas frente a otras alternativas

<https://www.huenei.com/ventajas-de-react-para-construir-apps/>

<https://openwebinars.net/blog/comparativa-react-native-y-diferentes-frameworks/#:~:text=React%20Native%20tiene%20algunas%20ventajas,el%20frontend%20y%20el%20backend.>

<https://blog.nubecollectiva.com/5-populares-aplicaciones-que-usan-tecnologia-react/>

Hooks

<https://es.reactjs.org/docs/hooks-intro.html>

JSX

<https://reactnative.dev/docs/intro-react>

Redux

<http://blog.enriqueoriol.com/2018/08/que-es-redux.html>

Virtual DOM

<https://es.reactjs.org/docs/faq-internals.html>

<https://programmingwithmosh.com/react/react-virtual-dom-explained/>

vs. Desarrollo nativo

<https://www.estudioyobo.com/blog/2020/01/react-native-vs-desarrollo-nativo/>

<https://hub.packtpub.com/react-native-really-native-framework/>

<https://stackoverflow.com/questions/41813824/what-is-native-and-what-is-not-in-a-react-native-project>

<https://matiashernandez.dev/agregar-modulos-nativos-a-una-aplicacion-react-native>