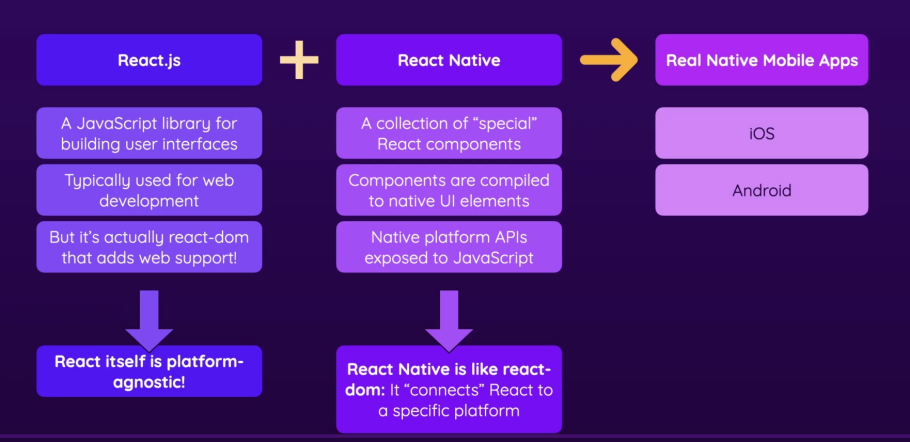
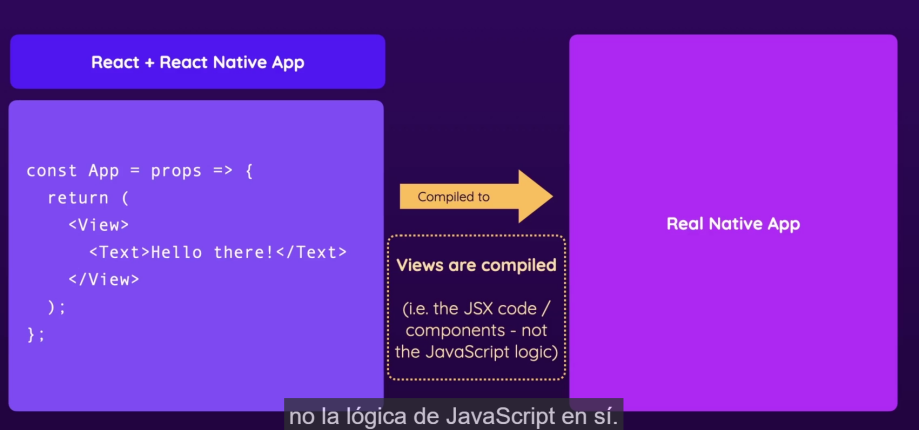
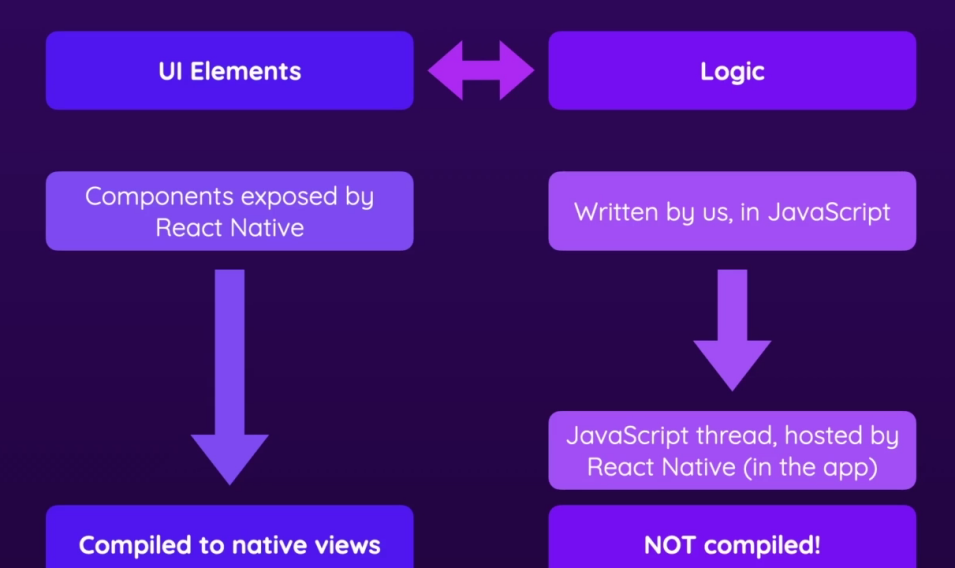
React Native:





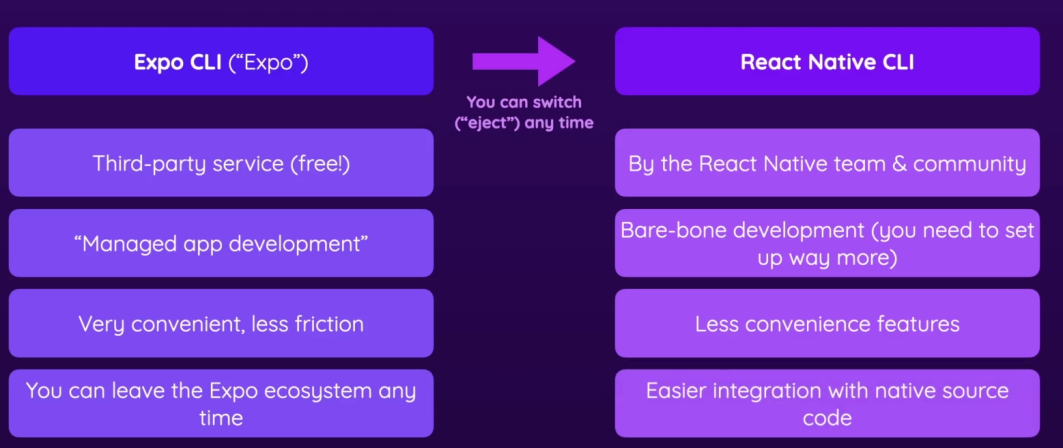
* View y Text son componentes especiales de RN que deben compilarse por RN para convertir en JS para Android o IOS, la lógica de JS no se compila.



Comparación WEB, Android, Ios y React Native (RN)



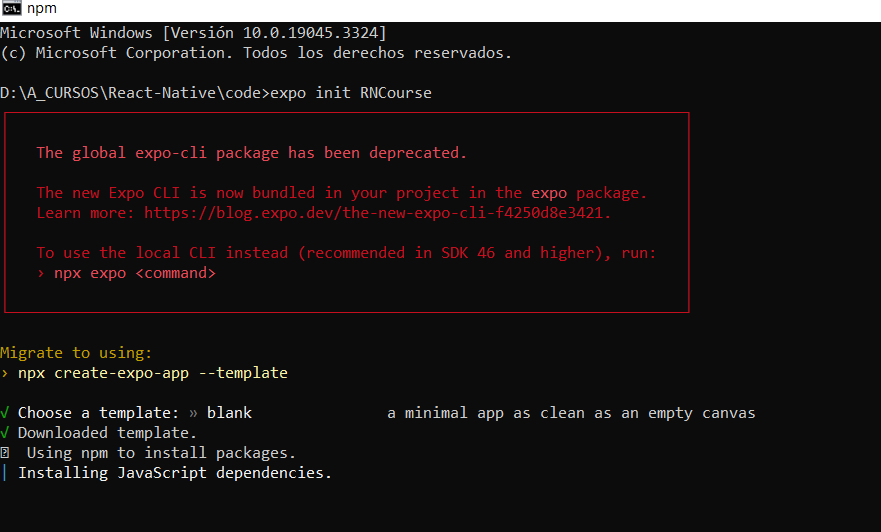
Proyectos RN



**Primer Programa**

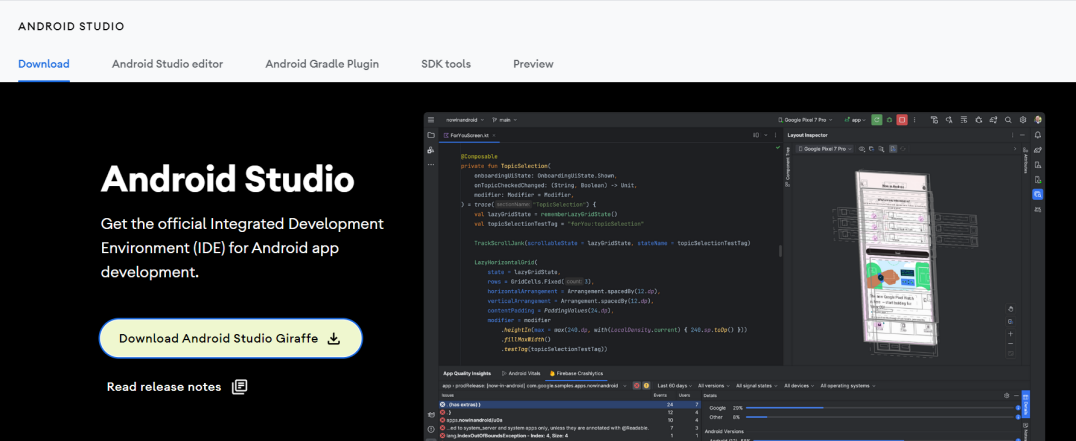
* Instalar NodeJS.
* Instalar expo-cli: [https://reactnative.dev/docs/environment-setup]
* npm cache clean --force
* npm cache verify
* npm -g uninstall expo-cli --save
* npm install expo-cli --global
* expo init guess-game
* cd app-name
* npm start
* Cear un proyecto Expo RN. En CMD de Windows

> expo init RNCourse

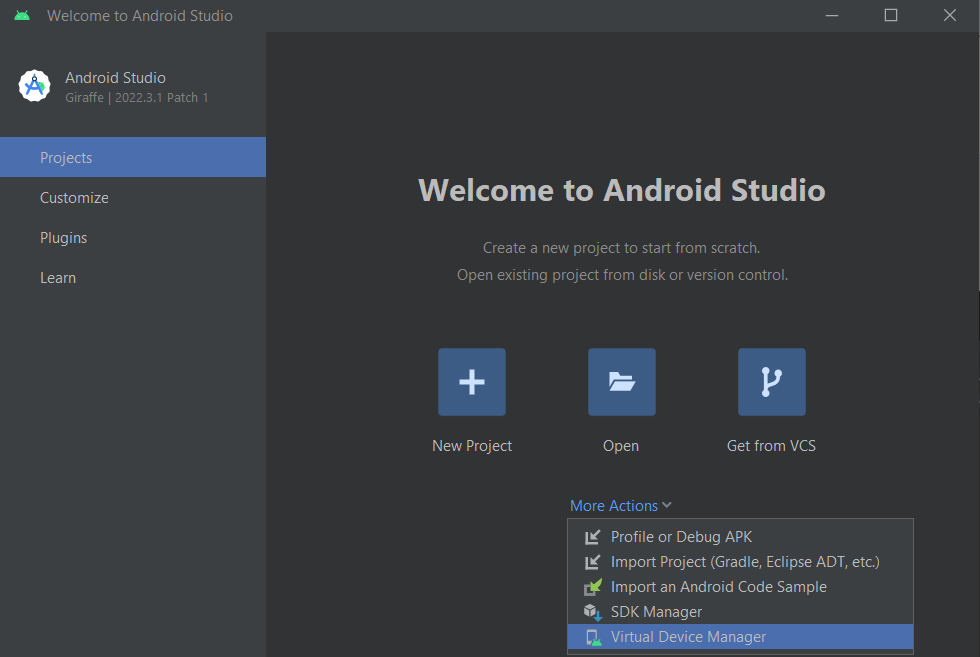


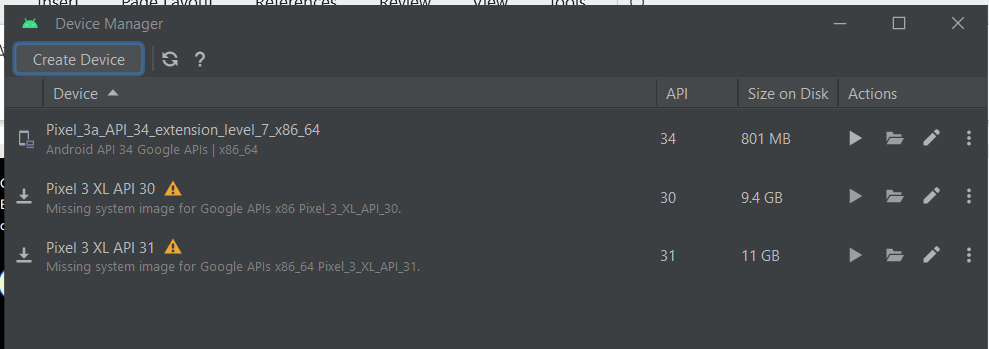


* Instalar en el celular expo go (Play Store) y escanear codigo de barra QR
* Instalar Android Studio

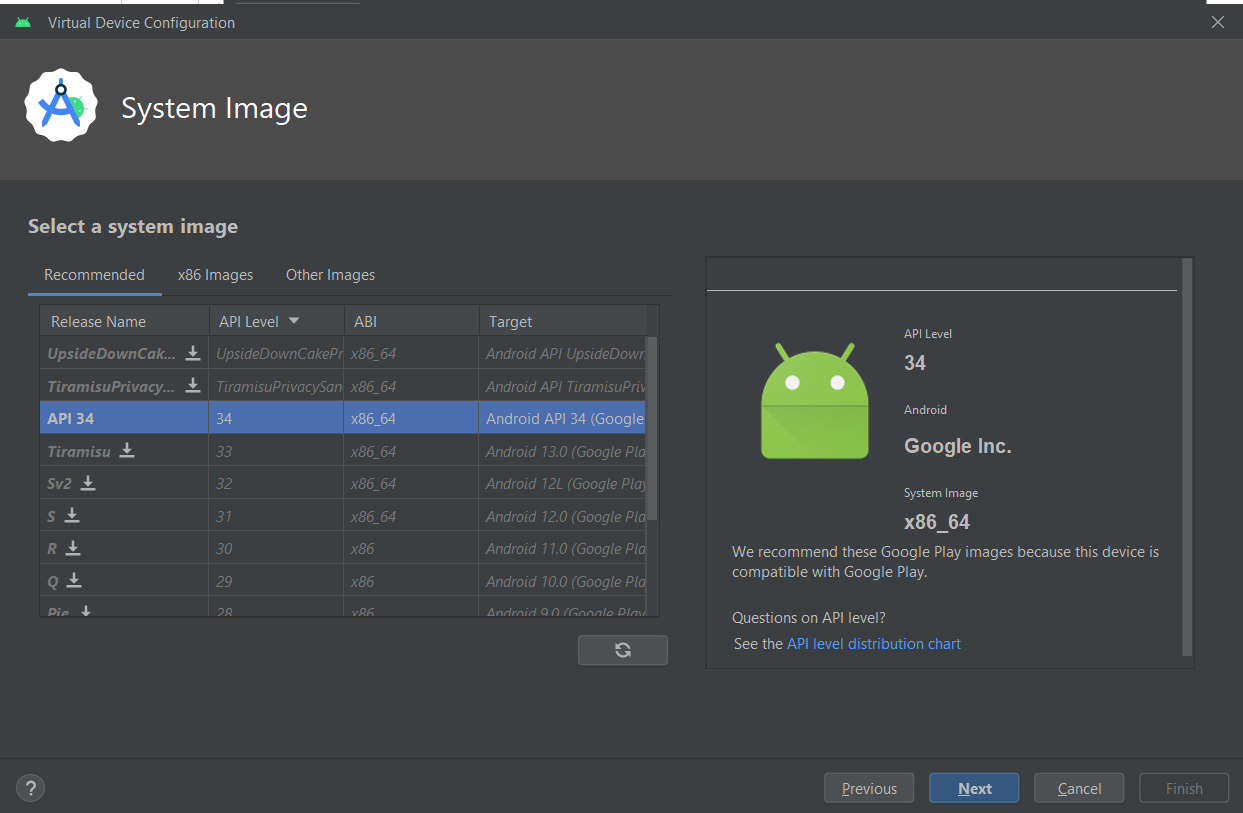


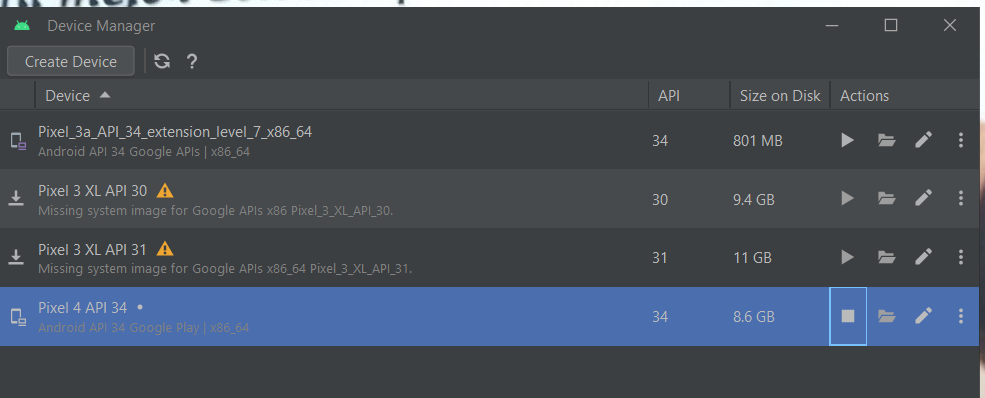
* Crear un dispositivo nuevo que tenga App Store



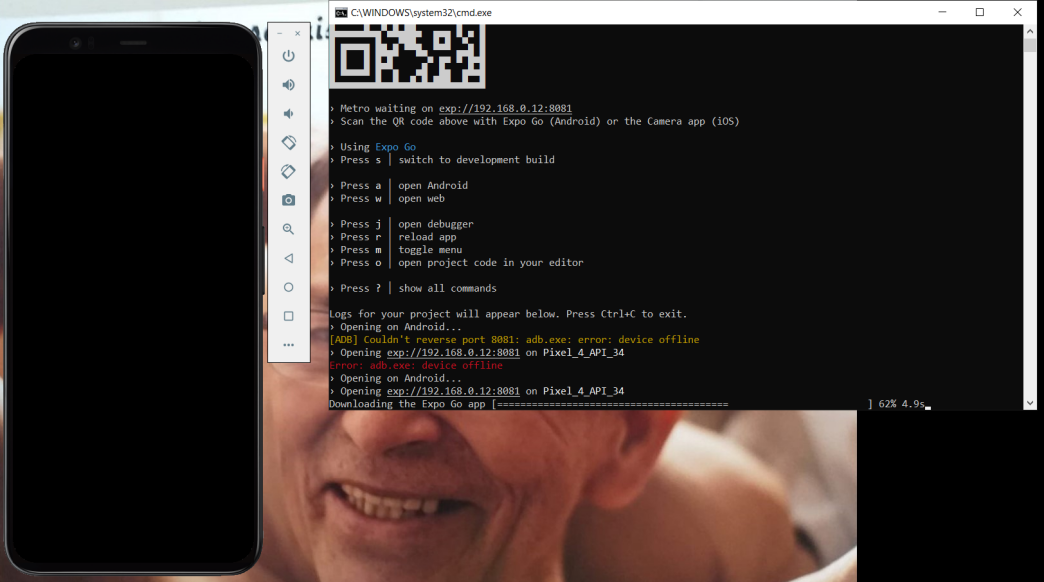


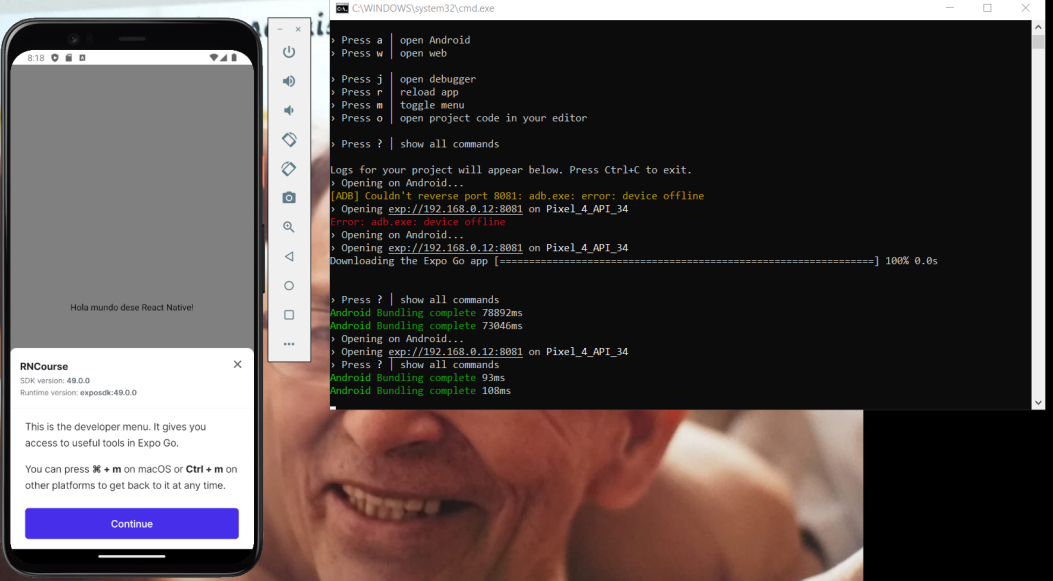
* Crear dispositivo. Pixel 4 Android. Instalar programas por defecto.



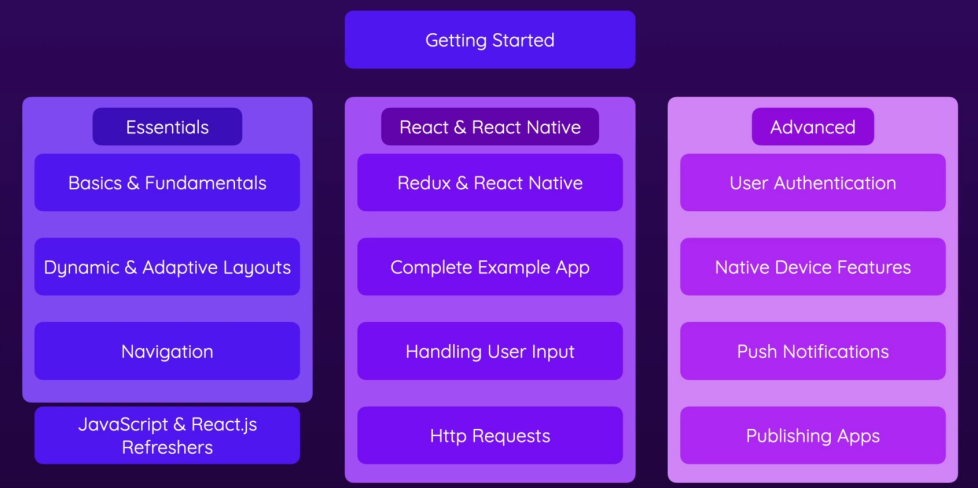


* Reiniciar App NR y seleciconar opcion a: buscar emulador Puxel 4 API 34.





* Para IOS Iphone 13. Solo en Equipos Apple.



* Al final del curso hay un repaso general de JS y ReactJS.

<https://github.com/academind/react-native-practical-guide-code>

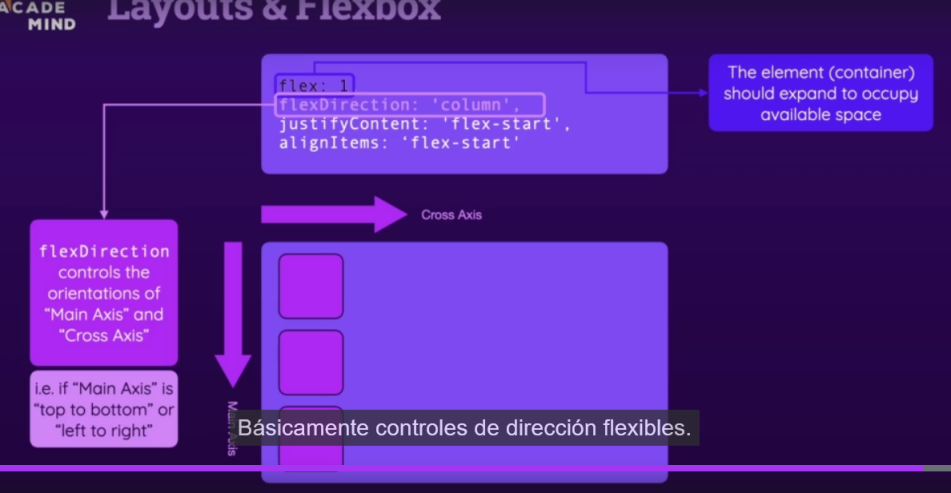
**Componentes**

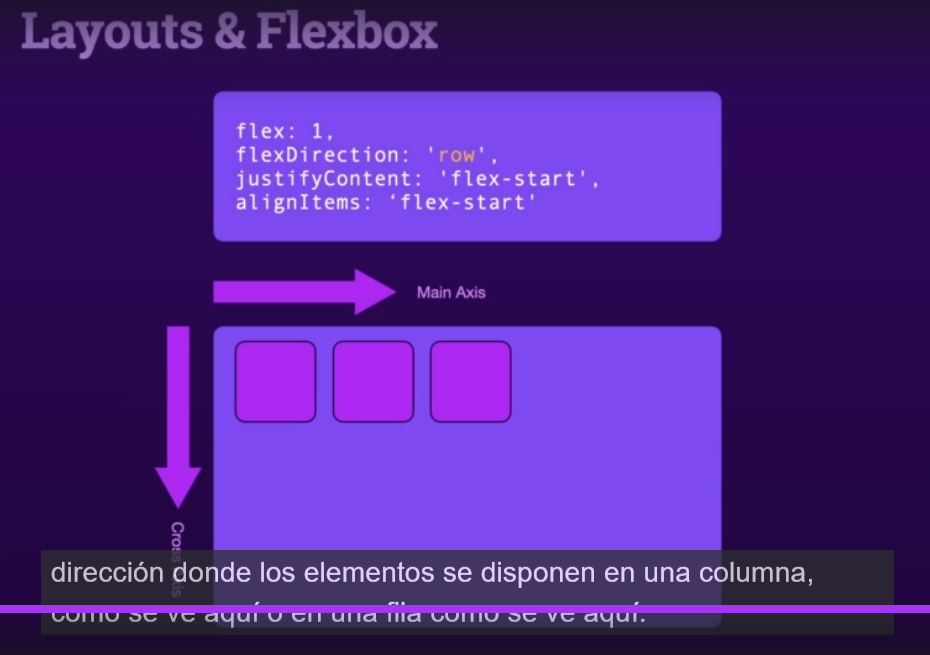
* <View>: Contenedor para mostrar nuestros componentes
* <Text>: Componente de Textos.

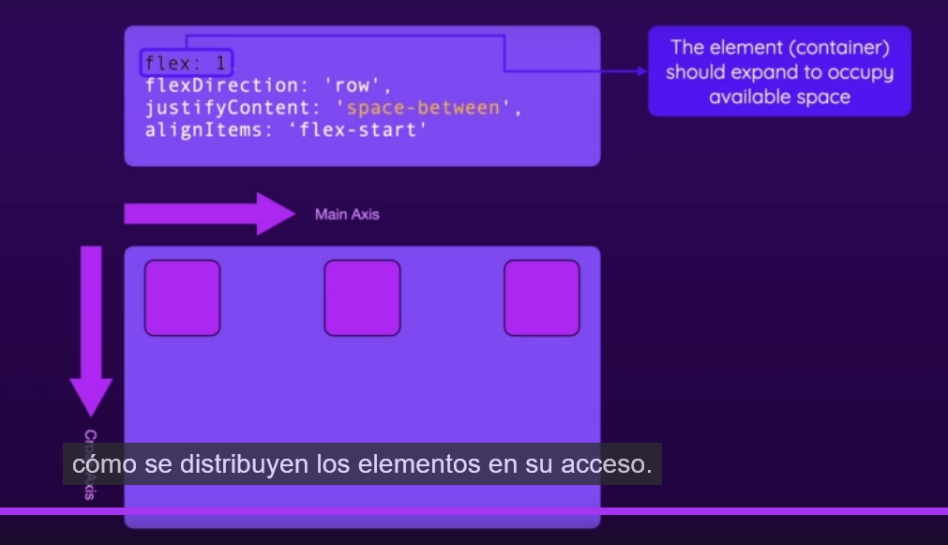


**Flexbox:** Permite organizar elementos secundarios en un espacio unidimensional.

* En React Native los elementos se organizan por defecto con Flexbox de arriba abajo y de izquierda a derecha (columnas). A diferencia de React WEB.







* Existe un eje principal y un eje transversal si es row o colum. Ejemplo: de izquierda a derecha eje principal y de arriba a abajo eje transversal si es row.
* Utilizar justifyContent para el eje principal
* Utiliza alignItems para el eje transversal
* Flex 1 implica que debe utilizar todo el espacio disponible en su eje. Si existen en un mismo eje varios flex para cada children se suma el total de flex y luego se distribuye el tamaño según los children. EJ: flex: 2 en un hijo, flex: 3 en otro hijo. Tamayo total 5. El primero 2 de 5 y el segundo 3 de 5

A: flex: 2 y B: flex: 3

|A|A|B|B|B| Total 5 espacios = dos de A y tres de B.

**Uso de Funciones**

  const goalInputHandler = () => {};

  return (

    <View style={styles.appContainer}>

      <View style={styles.inputContainer}>

        <TextInput

          style={styles.textInput}

          placeholder="Meta de este curso!"

          onChangeText={goalInputHandler}

        />

...

En este código el evento onChangeText={goalInputHandler} no se le coloca paréntesis a la llamada a la función goalInputHandler para que se ejecute cuando se introduce un valor sobre el TextInput, si agrego paréntesis se llamaría automáticamente al iniciar el return del componente.

**Android vrs IOS**

* Existen parámetros en style que pueden funcionar en Android pero no en IOS o viceversa para algunos componentes como el caso de <Text /> en estos casos puede servid}r agregar un componente <View /> que contenga al componente <Text/> y funciona. Ej: redondear bordes de un Text.
* Los estilos aplicados en un componente no caen en cascada a los componentes internos de este componente. Si una View contiene un Text y aplico color a View este no afecta al componente Text. Son componentes independientes. No existe herencia ni cascada en los estilos entre los componentes en RN
* Existen casos en los cuales se debe escribir códigos totalmente diferentes para cada caso.

RN tiene ScrollView para recorrer elementos en una vista pero si son muchos elementos genera problemas de rendimiento, siempre contiene todos los elementos de la lista. FlatList solo maneja los elementos actuales a amostrar en la lista , alto rendimiento y despliegue mas rápido. Los valorews de la lista solo se generan o utilizan cuando se vallan a mostrar

**Cambio de NodeJS:**

Versión que queremos tener disponible:

> nvm install [version]

Versión que queremos usar:

> nvm use [version]

> nvm use 14 (CCB)

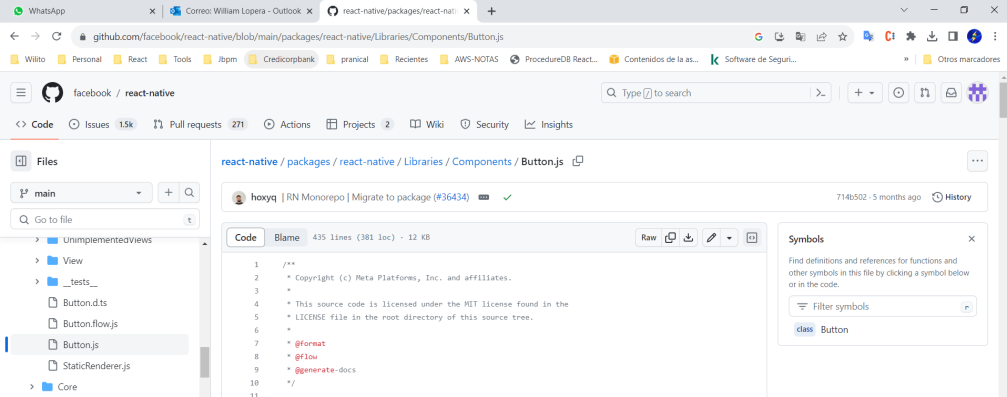
> nvm use 18 (RN)

**Para Monitorear consola de celular mas facilmente:**

Instalar una versión independiente de las herramientas

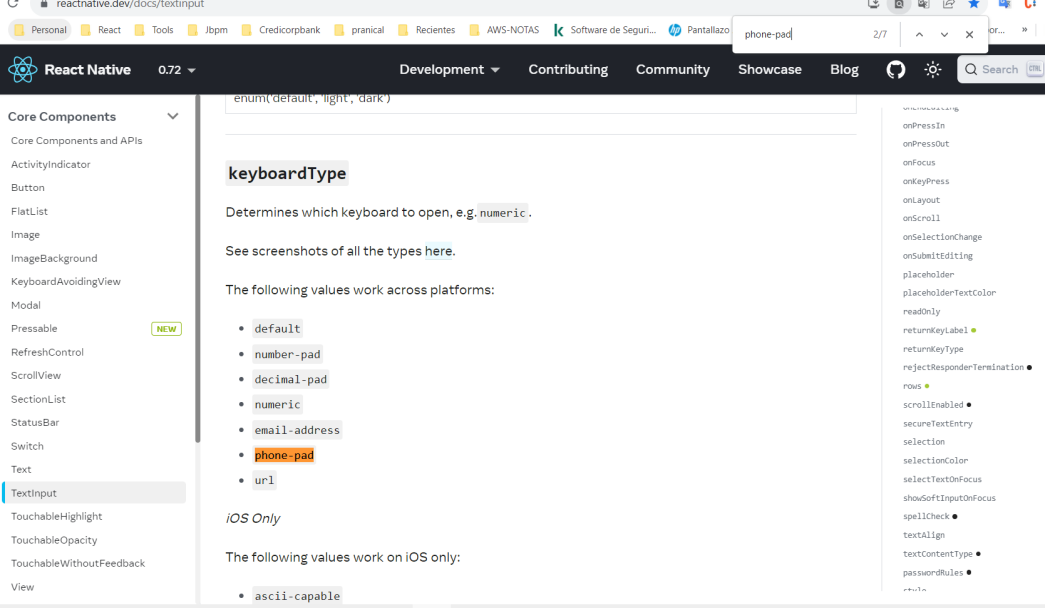
Npm install -g react-devtools

**Codigo fuente de RN**



* En la documentation oficial de RN podemos conseguir información sobre los componentes y su uso para IOS, Android o ambos:

> https://reactnative.dev/docs/textinput



* Las vistas <View> solo ocupan tanto espacio como necesitan para encajar su contenido en si mismas.
* LinearGradient de expo para RN

> expo install expo-linear-gradient

import { StyleSheet, View } from "react-native";

import { LinearGradient } from "expo-linear-gradient";

import StartGameScreen from "./screens/StartGameScreen";

export default function App() {

  return (

    <LinearGradient colors={["#4e0329", "#ddb52f"]} style={styles.rootScreen}>

      <StartGameScreen />

    </LinearGradient>

  );

}

const styles = StyleSheet.create({

  rootScreen: {

    flex: 1,

  },

});

**Uso de *Dimention* para controlar los tamaños de los componentes según las dimensiones de los equipos celulares**

import React from "react";

import { View, Text, StyleSheet, Dimensions } from "react-native";

import Colors from "../../constants/Colors";

const NumberContainer = ({ children }) => {

  return (

    <View style={styles.container}>

      <Text style={styles.numberText}>{children}</Text>

    </View>

  );

};

export default NumberContainer;

const deviceWidth = Dimensions.get('window').width

const styles = StyleSheet.create({

  container: {

    borderWidth: 4,

    borderColor: Colors.accent500,

    padding: deviceWidth < 380 ? 12 :24,

    marging: deviceWidth < 380 ? 12 :24,

    borderRadius: 8,

    justifyContent: 'center',

    alignContent:'center',

  },

  numberText: {

    color:Colors.accent500,

    fontSize: deviceWidth < 380 ? 28 :36,

    fontFamily:'open-sans-bold',

    //fontWeight: 'bold',

    textAlign: 'center'

  },

});

* Importamos librería o servicio ***Dimensions*** que permite consultar datos actuales del equipo móvil

import { Dimensions } from "react-native";

* Se puede consultar el ancho del móvil

const deviceWidth = Dimensions.get('window').width

* Se define el tamaño de ***padding*** y ***marging*** *según* el ancho del móvil. Si el ancho es menor a 380 se da un valor sino se define otro valor.

padding: deviceWidth < 380 ? 12 :24

marging: deviceWidth < 380 ? 12 :24

**Uso de *useWindowDimensions* para controlar los tamaños de los componentes según las dimensiones de los equipos celulares de manera dinámica**

import React from "react";

import {

  Image,

  StyleSheet,

  Text,

  View,

  ScrollView,

  useWindowDimensions,

} from "react-native";

import Colors from "../constants/Colors";

import PrimaryButton from "../components/ui/PrimaryButton";

import Title from "../components/ui/Title";

const GameOverScreen = ({ userNumber, roundsNumber, onStartNewGame }) => {

  const { width, height } = useWindowDimensions();

  let imageSize = 300;

  if (width < 380) {

    imageSize = 150;

  }

  if (height < 400) {

    imageSize = 80;

  }

  const imageStyle = {

    width: imageSize,

    height: imageSize,

    borderRadius: imageSize / 2,

  };

  return (

    <ScrollView style={styles.screen} >

      <View style={styles.rootContainer}>

        <Title>JUEGO TEMINADO!</Title>

        <View style={[styles.imageContainer, imageStyle]}>

          <Image

            style={styles.image}

            source={require("../assets/images/success.png")}

          />

        </View>

        <View>

          <Text style={styles.summaryText}>

            Necesitastes<Text style={styles.highlight}> {roundsNumber} </Text>

            rondas para adiviniar el número

            <Text style={styles.highlight}> {userNumber} </Text>

          </Text>

        </View>

        <PrimaryButton onPress={onStartNewGame}>Nuevo Juego</PrimaryButton>

      </View>

    </ScrollView>

  );

};

:::

* Importar librería o servicio ***useWindowDimensions*** que permite consultar datos actuales del equipo móvil

import { useWindowDimensions } from "react-native";

* Se generan valores dinamicamente tomando como base los valores retornados por el servicio **useWindowDimensions**

const { width, height } = useWindowDimensions();

  let imageSize = 300;

  if (width < 380) {

    imageSize = 150;

  }

  if (height < 400) {

    imageSize = 80;

  }

  const imageStyle = {

    width: imageSize,

    height: imageSize,

    borderRadius: imageSize / 2,

  };

* Se agregan los estilos a la vista agregando o sobre escribiendo algunos parámetros como ***width, height y borderRadius***, de manera dinámica

<View style={[styles.imageContainer, imageStyle]}>

**Uso de Platform para identificar el tipo de dispositivo móvil**

import React from "react";

import { Platform, StyleSheet, Text } from "react-native";

const Title = ({ children }) => {

  return <Text style={styles.title}>{children}</Text>;

};

export default Title;

const styles = StyleSheet.create({

  title: {

    fontFamily:'open-sans-bold',

    fontSize: 20,

    // fontWeight: "bold",

    color: "white",

    textAlign: "center",

    borderWidth: Platform.OS ==='android' ? 2: 0,

  // borderWidth: Platform.select({android:2, ios: 0}),

    borderWidth: 2,

    borderColor: "white",

    padding: 12,

    maxWidth: '80%',

    width: 300

  },

});

* Importar la librería o servicio Platform que permite consultar el tipo de dispositivo móvil

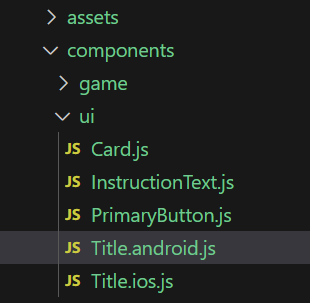
import { Platform} from "react-native";

* Generar valores de estilos según el tipo de plataforma actual

borderWidth: Platform.OS ==='android' ? 2: 0 ó

borderWidth: Platform.select({android:2, ios: 0})

**Otra forma de diferenciar y controlar los procesos y parámetros o estilos de las plataformas, es definir archivos con nombre que identifican la plataforma. Internamente solo se importa el componente sin incluir el nombre de la plataforma**



Ejemplo:

import Title from "../components/ui/Title";

<View style={styles.rootContainer}>

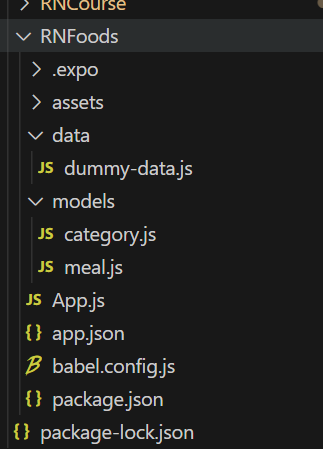
        <Title>JUEGO TEMINADO!</Title>

...

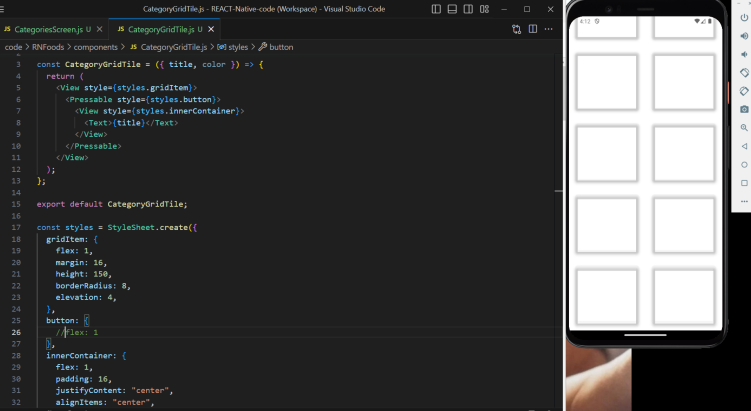
* RN determina automáticamente el tipo de dispositivo actual y toma según el caso
* Si es android toma el componente o archivo: Title.android.js
* Si es ios toma el componente o archivo: Title.ios.js

**Proyecto Platos de Comidas por países en React Native**

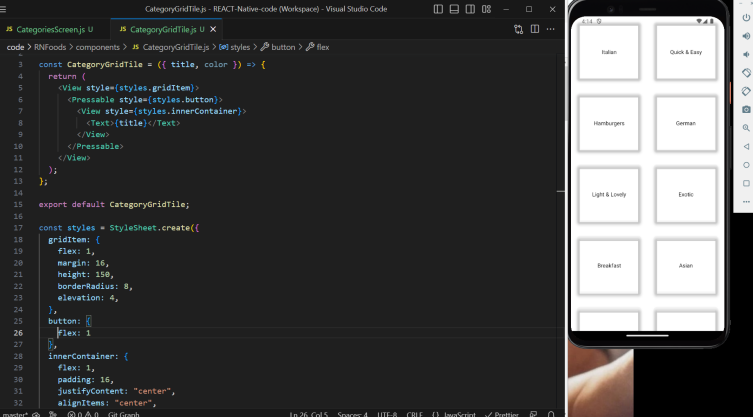
* Crear proyecto
* > expo init RNFoods



* Probar
* Si tengo un componente que no tiene definido area (Flex), no se muestra. En este ejemplo se comenta el flex:1 del estilo button. No existe area no se ocupa ningún espacio



* Si ahora le agrego el flex: 1



**Uso de React Navigation. Paquete de terceros.**

* Creado para agregar enrutamiento y navegación
* <https://reactnavigation.org/>
* Intalar el paqute recat-navigation
* npm install @react-navigation/native
* Como es un proyecto con expo, debemos instalar dependencias
* npx expo install react-native-screens react-native-safe-area-context
* Para un mejor rendimiento de los navegadores utilizar navigation stack. Se crea una pila nativa de navegación, utiliza elementos de plataforma nativos.
* npm install @react-navigation/stack
* npm install @react-navigation/native-stack

**Native-stack puede tener mas rendimiento que solo stack que emula comportamientos nativos**

**Configuración de la pantalla predeterminada**

Al configurar un Navegador (como <Stack.Navigator>) y registrar sus pantallas (a través de <Stack.Screen>), puede decidir qué pantalla se mostrará de forma predeterminada cuando se inicie la aplicación.

de fábrica, la pantalla superior (es decir, el primer elemento secundario dentro de <Stack.Navigator>) se utiliza como pantalla inicial.

Es decir, en el siguiente ejemplo, la pantalla Todos los productos se mostraría como pantalla inicial cuando se inicia la aplicación:

<Stack.Navigator>

<Stack.Screen name="AllProducts" component={AllProducts} /> // initial screen

<Stack.Screen name="ProductDetails" component={ProductDetails} />

</Stack.Navigator>

Por lo tanto, puede cambiar la pantalla inicial cambiando el orden de <Stack.Screen>. Alternativamente, también hay un accesorio initialRouteName que se puede configurar en el componente del navegador (es decir, en <Stack.Navigator> en este caso):

<Stack.Navigator initialRouteName="ProductDetails">

<Stack.Screen name="AllProducts" component={AllProducts} />

<Stack.Screen name="ProductDetails" component={ProductDetails} /> // initial screen

</Stack.Navigator>

**Navegar desde el interior de un componente no registrado como pantalla**

En el caso actual se utilizo native-stack y se registro en App.js las pantallas a navegar así se puede pasar el objeto navigation entre pantallas y así se navega

Otra forma es utilizar el utilizar el gancho *useNavigation* que viene en navigate y allí obtener la navegación sin tener que pasarla por las pantallas.

* Pasando función y objeto navigation entre pantallas

import { Pressable, StyleSheet, Text, View, Platform } from "react-native";

const CategoryGridTile = ({ title, color, onPress }) => {

  return (

    <View style={styles.gridItem}>

      <Pressable

        android\_ripple={{ color: "#ccc" }}

        style={(pressed) => [

          styles.button,

          pressed ? styles.buttonPressed : null,

        ]}

        onPress={onPress}

      >

        <View style={[styles.innerContainer, { backgroundColor: color }]}>

          <Text style={styles.title}>{title}</Text>

        </View>

      </Pressable>

    </View>

  );

};

* Obteniendo objeto navigation de gancho useNavigation

import { Pressable, StyleSheet, Text, View, Platform } from "react-native";

import { useNavigation } from "@react-navigation/native";

const CategoryGridTile = ({ title, color, onPress }) => {

  const navigation = useNavigation();

  return (

    <View style={styles.gridItem}>

      <Pressable

        android\_ripple={{ color: "#ccc" }}

        style={(pressed) => [

          styles.button,

          pressed ? styles.buttonPressed : null,

        ]}

        onPress={() => navigation.navigate("MealsOverview")}

      >

        <View style={[styles.innerContainer, { backgroundColor: color }]}>

          <Text style={styles.title}>{title}</Text>

        </View>

      </Pressable>

    </View>

  );

};

**Pasar parametros entre pantallas**

* LLamada a una pantalla

 navigation.navigate("MealsOverview");

* LLamada a una pantalla y agrego objeto con parámetros

 navigation.navigate("MealsOverview", { categoryId: itemData.item.id });

* Para extraer los parametros: route.params => Este viene del objeto navigation que se pasa en la llamada a la pantalla

const MealsOverviewScreen = ({ route }) => {

  const catId = route.params.categoryId;

  return (

    <View style={styles.container}>

      <Text>Resumen de comidas: {catId}</Text>

    </View>

  );

};

* También se puede utilizar un gancho que retorna los valores o parámetros

import {useRoute} from '@react-navigation/native'

const MealsOverviewScreen = () => {

  const route = useRoute()

  const catId = route.params.categoryId;

**Pasar parámetros a las pantallas**

export default function App() {

  return (

    <>

      <StatusBar style="dark" />

      <NavigationContainer>

        <Stack.Navigator

          screenOptions={{

            headerStyle: { backgroundColor: "#351401" },

            headerTintColor: "white",

            contentStyle: { backgroundColor: "#3f2f25" },

          }}

        >

          <Stack.Screen

            name="MealsCategories"

            component={CategoriesScreen}

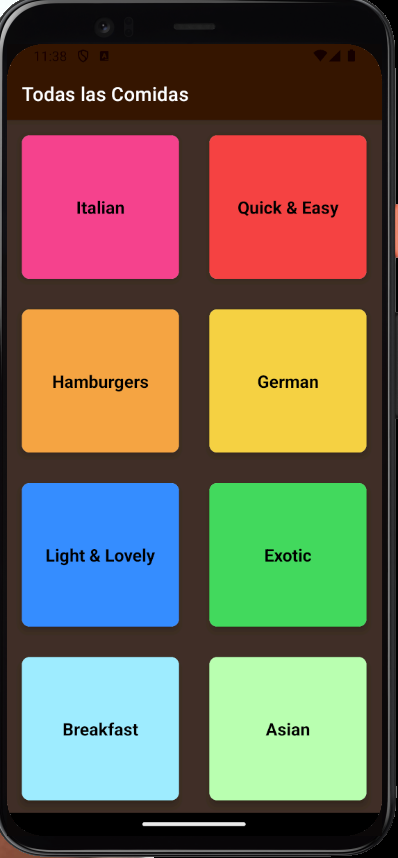
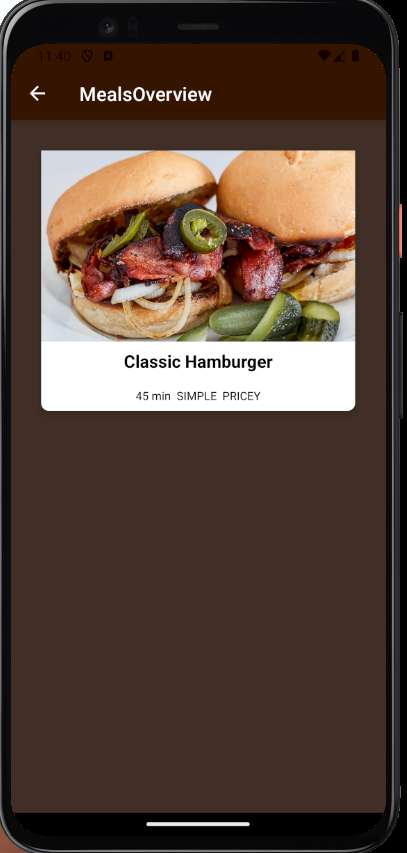
            options={{

              title: "Todas las Comidas",

            }}

          />

* **screenOptions={...}** en ***Stack.Navigation***: Parámetros a todas las vistas
* **options={{...** en **Stack.Screen** parámetros indivic}duales por vista

**Pasar parámetros a las pantallas de manera dinámica**

<Stack.Screen

            name="MealsOverview"

            component={MealsOverviewScreen}

            options={({route, navigation}) => {

              const catId = route.params.categoryId;

              return {

                title: catId,

              };

            }}

          />

* **options={({route, navigation}) =>** Se definen dos objetos route y navigation. De alli podemos tomar los valores de la pantalla de manera dinámica

import { MEALS, CATEGORIES } from "../data/dummy-data";

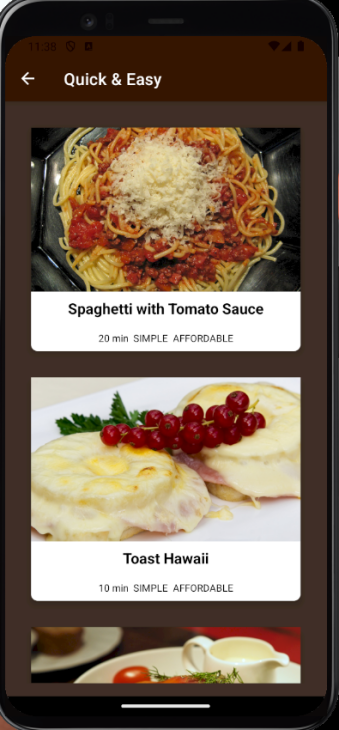
const MealsOverviewScreen = ({ route, navigation }) => {

  const catId = route.params.categoryId;

  const categoryTitle = CATEGORIES.find((category) => category.id === catId).title;

  navigation.setOptions({ title: categoryTitle });

* **const MealsOverviewScreen = ({ route, navigation }) =>** Se definen dos objetos route y navigation. De alli podemos tomar los valores de la pantalla de manera dinámica.
* **navigation.setOptions({ title: categoryTitle })** Se pasan los valores a la pantalla



Es mejor, utilizar

  useEffect(() => {

    const categoryTitle = CATEGORIES.find((category) => category.id === catId).title;

    navigation.setOptions({ title: categoryTitle });

    }, [catId, navigation])

***CATEGORIES (import)*** no se agrega al arreglo de dependencias del ***useEffect***, porque son funciones o valores externos pero no forman parte de la función componente como route y navigation

Se presenta un pequeño salto al momento de desplegar la imagen debido al **useEffect**, para evitar eso se utiliza **useLayoutEffect.** Este se utiliza en casos en los que se desea ejecutar algún efecto secundario mientras se ejecuta el **useEffect** y antes de que se allá renderizado el componente. O mejor que se ejecute simultaneamente con la función componente.

**Agregar botones, opciones de menú y/o textos a las cabeceras**

 <Stack.Screen

            name="MealDetails"

            component={MealDetailsScreen}

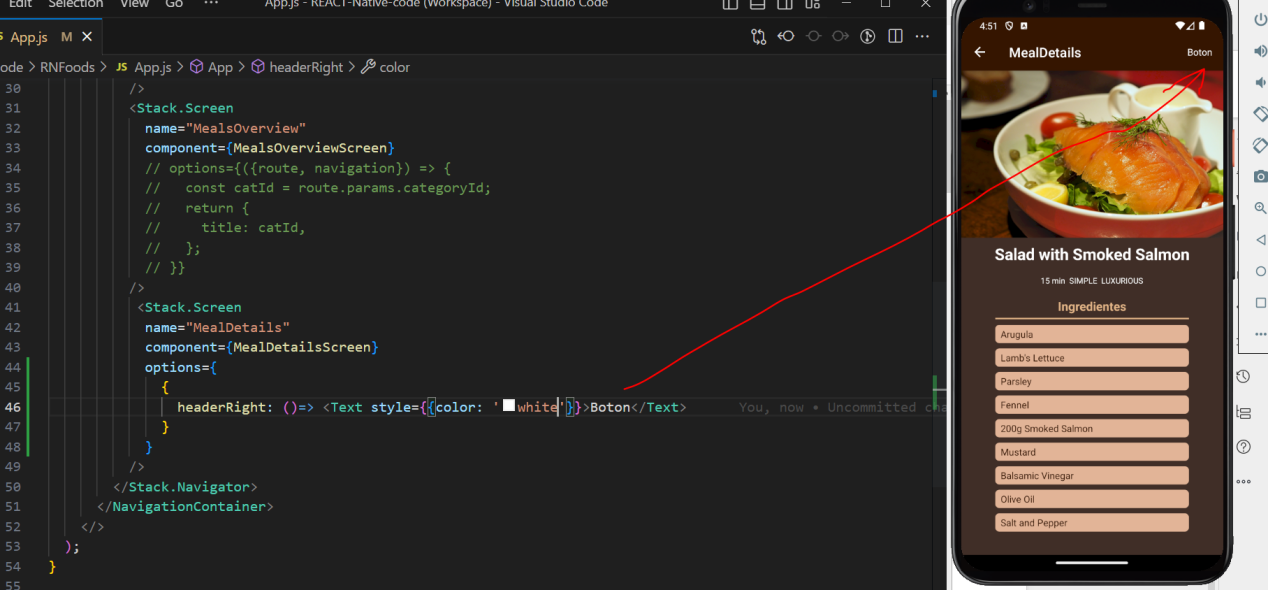
            options={

              {

                headerRight: ()=> <Text style={{color: 'white'}}>Boton</Text>

              }

            }



 <Stack.Screen

            name="MealDetails"

            component={MealDetailsScreen}

            options={

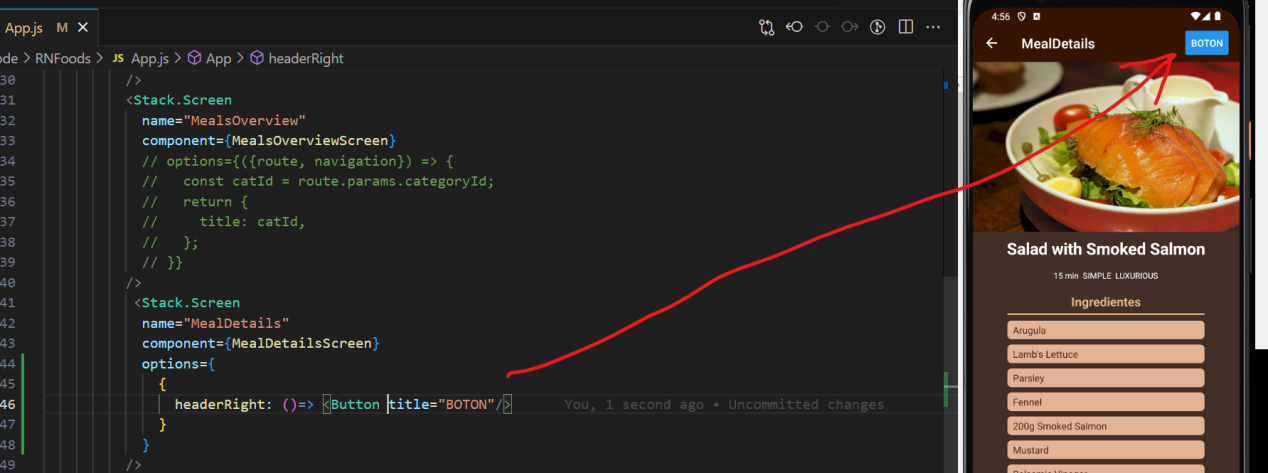
              {

                headerRight: ()=> <Button title="BOTON"/>

              }

            }

          />



**Si se requiere comunicación directa contra el componete a agregar en la cabecera se debe ir directamente contra la pantalla que tendrá el botón**

const MealDetailsScreen = ({ route, navigation }) => {

  const mealId = route.params.mealId;

  const selectedMeal = MEALS.find((meal) => meal.id === mealId);

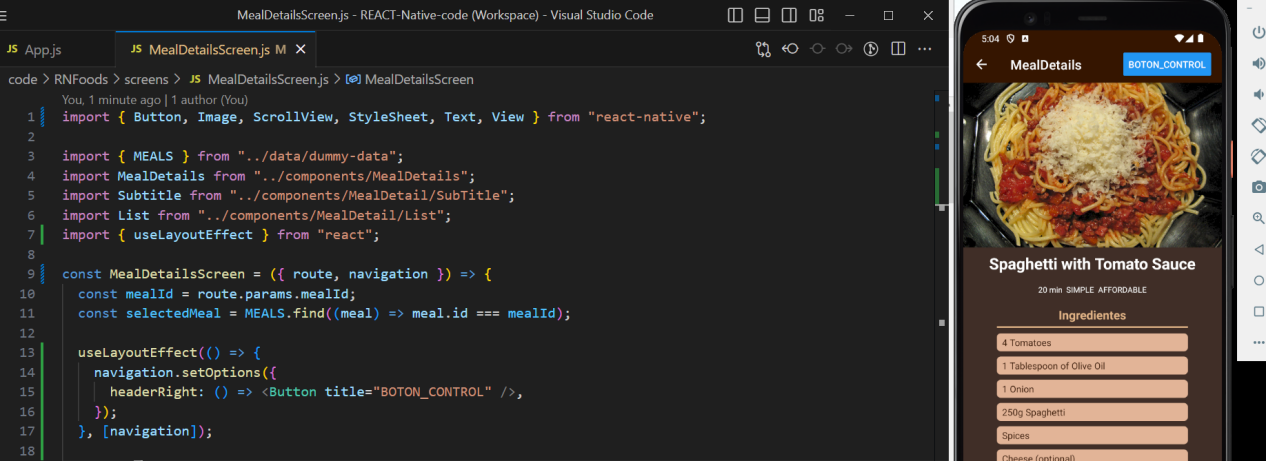
  useLayoutEffect(() => {

    navigation.setOptions({

      headerRight: () => <Button title="BOTON\_CONTROL" />,

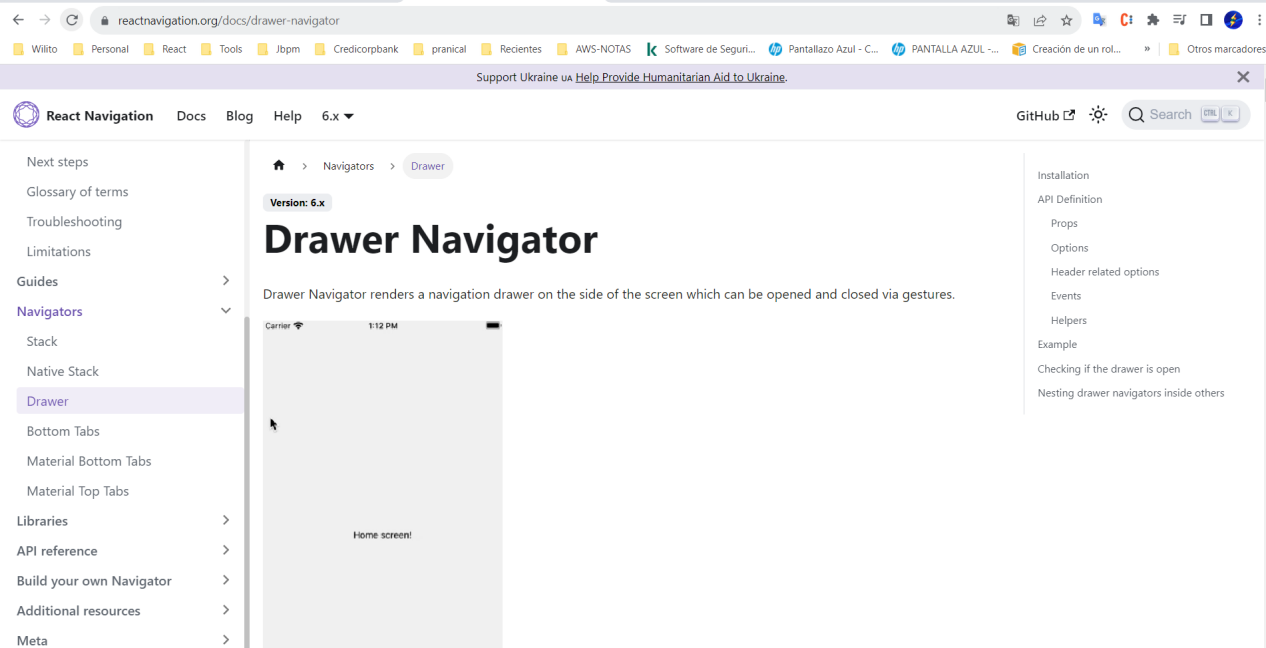
    });

  }, [navigation]);

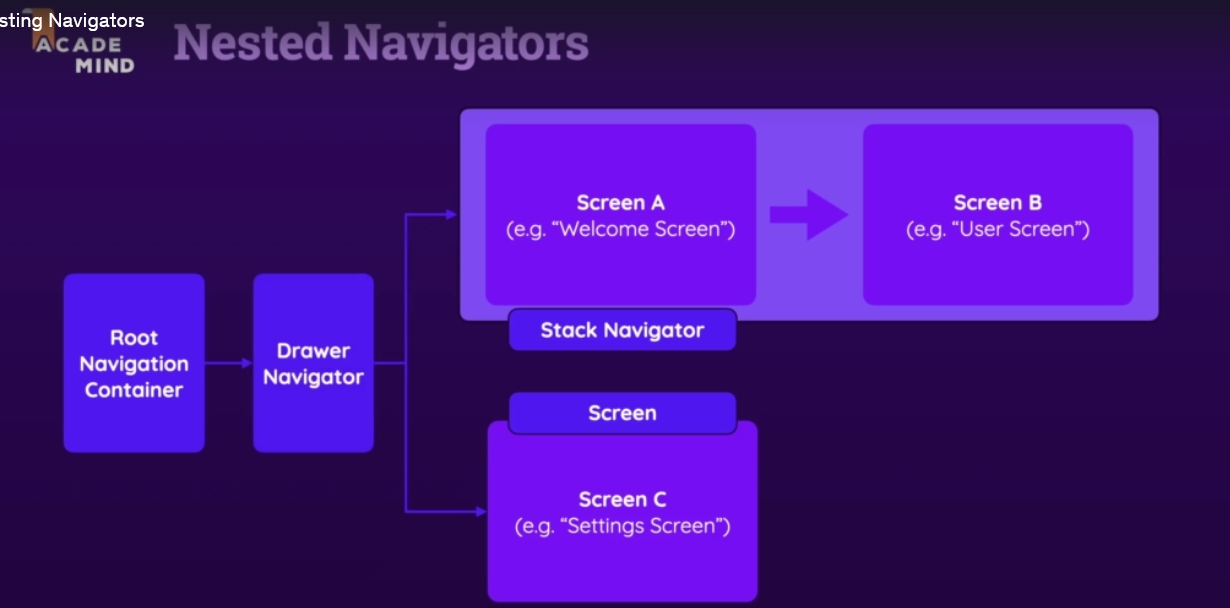


**Uso de Navegadores**

* https://reactnavigation.org/



Ver Documento ***React Native - Navigate-Drawer.doc***



Instalar

* npm install @react-navigation/drawer
* npx expo install react-native-gesture-handler react-native-reanimated
* Actualizar Babel.config.js

module.exports = function (api) {

  api.cache(true);

  return {

    presets: ["babel-preset-expo"],

    plugins: ["react-native-reanimated/plugin"],

  };

};

Agregar: plugins: ["react-native-reanimated/plugin"]

* **En caso de ser necesario:** Importar libreria en App.js, antes del import de los otros navegadores
* plugins: ["react-native-reanimated/plugin"]
* Correr aplicativo
* npx expo start --reset-cache

import { StatusBar } from "expo-status-bar";

import { StyleSheet } from "react-native";

import { NavigationContainer } from "@react-navigation/native";

import { createNativeStackNavigator } from "@react-navigation/native-stack";

import { createDrawerNavigator } from "@react-navigation/drawer";

import {Ionicons} from '@expo/vector-icons'

import CategoriesScreen from "./screens/CategoriesScreen";

import MealsOverviewScreen from "./screens/MealsOverviewScreen";

import MealDetailsScreen from "./screens/MealDetailsScreen";

import FavoritesScreen from "./screens/FavoritesScreen";

const Stack = createNativeStackNavigator();

const Drawer = createDrawerNavigator();

const DrawerNavigator = () => {

  return (

    <Drawer.Navigator

      screenOptions={{

        headerStyle: { backgroundColor: "#351401" },

        headerTintColor: "white",

        sceneContainerStyle: { backgroundColor: "#3f2f25" },

        drawerContentStyle: {backgroundColor: "#351401"},

        drawerInactiveTintColor: 'white',

        drawerActiveTintColor: "#351401",

        drawerActiveBackgroundColor: "#e4baa1"

      }}

    >

      <Drawer.Screen

        name="Categories"

        component={CategoriesScreen}

        options={{

          title: "Todas las Categorias",

          drawerIcon: ({color, size}) => <Ionicons name="list" color={color} size={size}/>

        }}

      ></Drawer.Screen>

      <Drawer.Screen

        name="Favorites"

        component={FavoritesScreen}

        options={{

          title: "Favoritas",

          drawerIcon: ({color, size}) => <Ionicons name="star" color={color} size={size}/>

        }}

      ></Drawer.Screen>

    </Drawer.Navigator>

  );

};

export default function App() {

  return (

    <>

      <StatusBar style="light" />

      <NavigationContainer>

        <Stack.Navigator

          screenOptions={{

            headerStyle: { backgroundColor: "#351401" },

            headerTintColor: "white",

            contentStyle: { backgroundColor: "#3f2f25" },

          }}

        >

          <Stack.Screen

            name="Drawer"

            component={DrawerNavigator}

            options={{

              title: "Todas las Comidas",

              headerShown: false,

            }}

          />

          <Stack.Screen

            name="MealsOverview"

            component={MealsOverviewScreen}

            // options={({route, navigation}) => {

            //   const catId = route.params.categoryId;

            //   return {

            //     title: catId,

            //   };

            // }}

          />

          <Stack.Screen name="MealDetails" component={MealDetailsScreen}   options={{

              title: "Acerca de las Comidas",

            }}/>

        </Stack.Navigator>

      </NavigationContainer>

    </>

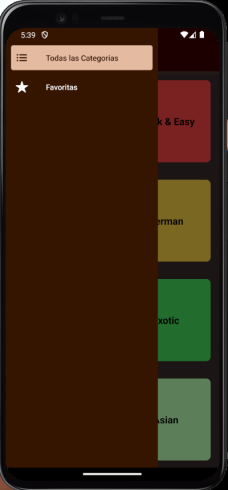
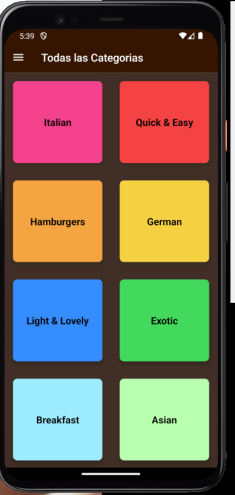
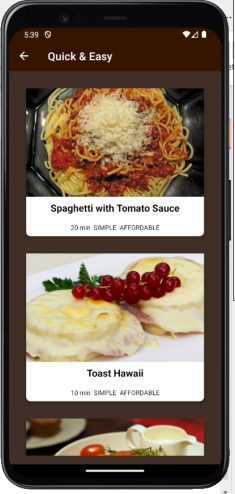
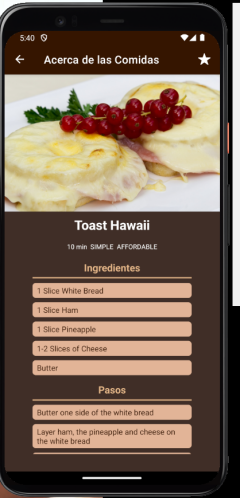
  );

}

const styles = StyleSheet.create({

  container: {},

});

**Uso de Context y Redux para pasar data entre componentes**



**Uso de Context React**

import { createContext, useState } from "react";

export const FavoritesContext = createContext({

  ids: [],

  addFavorite: (id) => {},

  deleteFavorite: (id) => {},

});

const FavoritesContextProvider = ({ children }) => {

  const [favoriteMealIds, setFavoriteMealIds] = useState([]);

  const addFavorite = (id) => {

    setFavoriteMealIds((currentFavIds) => [...currentFavIds, id]);

  };

  const deleteFavorite = (id) => {

    setFavoriteMealIds((currentFavlIds) =>

      currentFavlIds.filter((mealId) => mealId !== id)

    );

  };

  const value = {

    ids: favoriteMealIds,

    addFavorite: addFavorite,

    deleteFavorite: deleteFavorite,

  };

  return (

    <FavoritesContext.Provider value={value}>

      {children}

    </FavoritesContext.Provider>

  );

};

export default FavoritesContextProvider;

**App.js**

* Envolver nuestros componentes con Provider de Context

import FavoritesContextProvider from "./store/context/favorites-context";

export default function App() {

  return (

    <>

      <StatusBar style="light" />

      <FavoritesContextProvider>

        ...

      </FavoritesContextProvider>

    </>

  );

}

* Uso de Hook de React para tener cacceso al contexto: useContext

  const favoriteMealCtx = useContext(FavoritesContext);

**Uso de Redux/toolkit**

* npm install @reduxjs/toolkit react-redux

**App.js**

* Envolver nuestros componentes con Provider de redux

import { Provider } from "react-redux";

import { store } from "./store/redux/store";

export default function App() {

  return (

    <>

      <StatusBar style="light" />

      {/\* <FavoritesContextProvider> \*/}

      <Provider store={store} \>

       ...

      </Provider>

      {/\* </FavoritesContextProvider> \*/}

    </>

  );

}

* Crear carpetas de trabajo redux y store



* Crear favorites.js

import { createSlice } from "@reduxjs/toolkit";

const favoritesSlice = createSlice({

  name: "favorites",

  initialState: {

    ids: [],

  },

  reducers: {

    addFavorite: (state, action) => {

      state.ids.push(action.payload.id);

    },

    removeFavorite: (state, action) => {

      state.ids.splice(state.ids.indexOf(action.payload.id), 1);

    },

  },

});

export const addFavorite = favoritesSlice.actions.addFavorite;

export const removeFavorite = favoritesSlice.actions.removeFavorite;

export default favoritesSlice.reducer;

* Crear store.js

import { configureStore } from "@reduxjs/toolkit";

import favoriteReducer from "../redux/favorites";

export const store = configureStore({

  reducer: {

    favoriteMeals: favoriteReducer,

  },

});

* Actualizar pantalla MealDetailsScreen.js para utilizar redux

import { useDispatch, useSelector } from "react-redux";

…

import List from "../components/MealDetail/List";

import { useContext, useLayoutEffect } from "react";

import { addFavorite, removeFavorite } from "../store/redux/favorites";

const MealDetailsScreen = ({ route, navigation }) => {

  // const favoriteMealCtx = useContext(FavoritesContext);

  // Obtenermos los valores de ids

  const favoriteMealIds = useSelector((state) => state.favoriteMeals.ids);

  // Metodo requerido para enviar una solictud o action a redux (add/remove favorite)

  const dispatch = useDispatch();

  const mealId = route.params.mealId;

  const selectedMeal = MEALS.find((meal) => meal.id === mealId);

  // const mealIsFavorite = favoriteMealCtx.ids.includes(mealId);

  const mealIsFavorite = favoriteMealIds.includes(mealId);

  const changeFavoriteStatusHandler = () => {

    if (mealIsFavorite) {

      dispatch(removeFavorite({ id: mealId }));

      // favoriteMealCtx.deleteFavorite(mealId);

    } else {

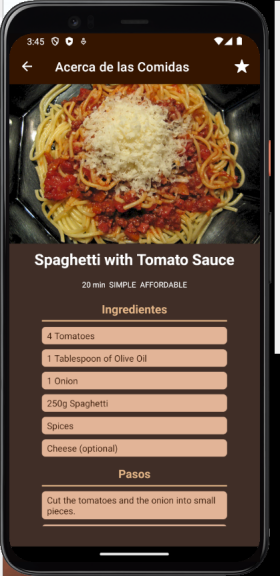
      dispatch(addFavorite({ id: mealId }));

      // favoriteMealCtx.addFavorite(mealId);

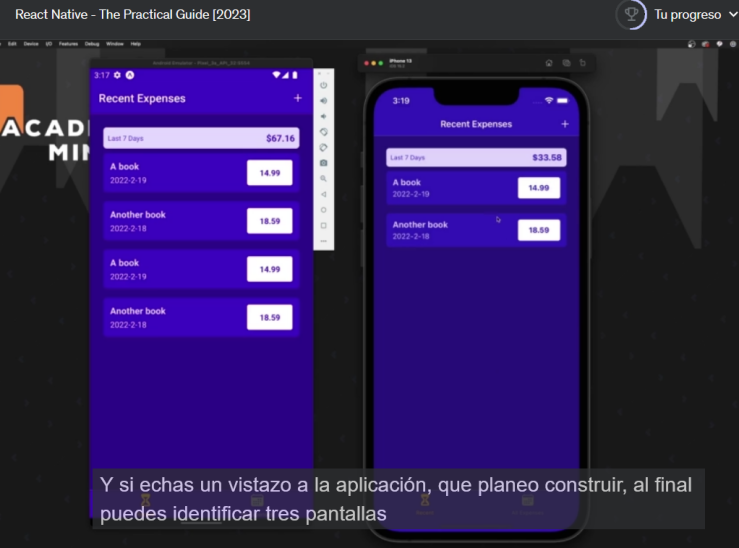
    }

  };

…

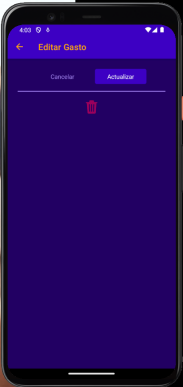
**RN Expenses: App RN de Gastos**



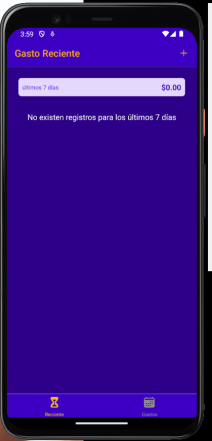
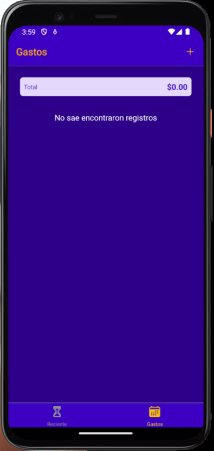
* App para mostrar gastos en React Native - Practica
* Crear App nueva: expo init RNExpense
* Lanzar App npx expo start --reset-cache
* Agregar Pantallas de inicio
* Instalar librerías de navegacion inicial
* npm install @react-navigation/native
* npx expo install react-native-screens react-native-safe-area-context
* npm install @react-navigation/native-stack
* npm install @react-navigation/bottom-tabs
* Se agrega todo el código para agregar, actualizar y borrar gastos, así como vistas para mostrar gastos y detalles de los mismos.

* Agregar, Editar y Borrar gastos

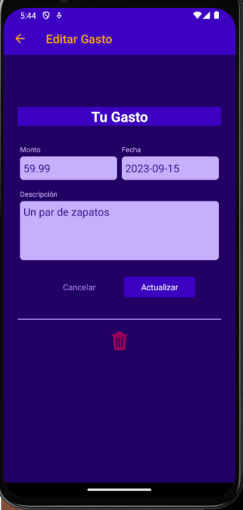
* Mensajes en caso de no existir data

Para manejo de datos se utiliza useReduce y useContext del API Context de React

* Se agrega formulario para agregar y modificar gastos
* Usod e TextInput y lógicas de JS

ver> https://reactnative.dev/docs/textinput



**Agregar validaciones**

* Cambiar las entradas por objetos

const ExpenseForm = ({

  submitButtonLabel,

  onCancel,

  onSubmit,

  defaultValues,

}) => {

  const [inputs, setInputs] = useState({

    amount: {

      value: defaultValues ? defaultValues.amount.toString() : "",

      isValid: true,

    },

    date: {

      value: defaultValues ? getFormattedDate(defaultValues.date) : "",

      isValid: true,

    },

    description: {

      value: defaultValues ? defaultValues.description : "",

      isValid: true,

    },

  });

* Validar Data de envio

const submitHandler = () => {

    const expenseData = {

      amount: +inputs.amount.value,

      date: new Date(inputs.date.value),

      description: inputs.description.value,

    };

    // if (dataIsValid(expenseData)) {

    //   onSubmit(expenseData);

    // }

    const amountIsValid =

      isNaN(expenseData.amount.value) && expenseData.amount > 0;

    const dateIsValid = expenseData.date.toString() !== "Invalid Date";

    const descriptionIsValid = expenseData.description.trim().length > 0;

    if (!amountIsValid || !dataIsValid || !descriptionIsValid) {

      setInputs((curInputs) => {

        return {

          amount: { value: curInputs.amount.value, isValid: amountIsValid },

          date: { value: curInputs.date.value, isValid: dateIsValid },

          description: {

            value: curInputs.description.value,

            isValid: descriptionIsValid,

          },

        };

      });

      return;

    }

    onSubmit(expenseData);

  };

* Retorno dle componente

return (

    <View style={styles.form}>

      <Text style={styles.title}>Tu Gasto</Text>

      <View style={styles.inputsRows}>

        <Input

          label="Monto"

          invalid={!inputs.amount.isValid}

          style={styles.rowInput}

          textInputConfig={{

            keyboardType: "decimal-pad",

            onChangeText: (value) => inputChangedHandler("amount", value),

            value: inputs.amount.value,

          }}

        />

        <Input

          label="Fecha"

          style={styles.rowInput}

          invalid={!inputs.date.isValid}

          textInputConfig={{

            placeholder: "YYYY-MM-DD",

            maxLength: 10,

            onChangeText: (value) => inputChangedHandler("date", value),

            value: inputs.date.value,

          }}

        />

      </View>

      <Input

        label="Descripción"

        invalid={!inputs.description.isValid}

        textInputConfig={{

          multiline: true,

          //autoCapitalize: 'none' //'words', default 'sentences', ...

          //autoCorrect:false, // default es true

          onChangeText: inputChangedHandler.bind(this, "description"),

          value: inputs.description.value,

          //name: "description",

        }}

      />

      {formIsInValid && <Text style={styles.errorText}> {formIsInValid}</Text>}

      <View style={styles.buttons}>

        <Button mode="flat" onPress={onCancel} style={styles.button}>

          Cancelar

        </Button>

        <Button onPress={submitHandler} style={styles.button}>

          {submitButtonLabel}

        </Button>

      </View>

    </View>

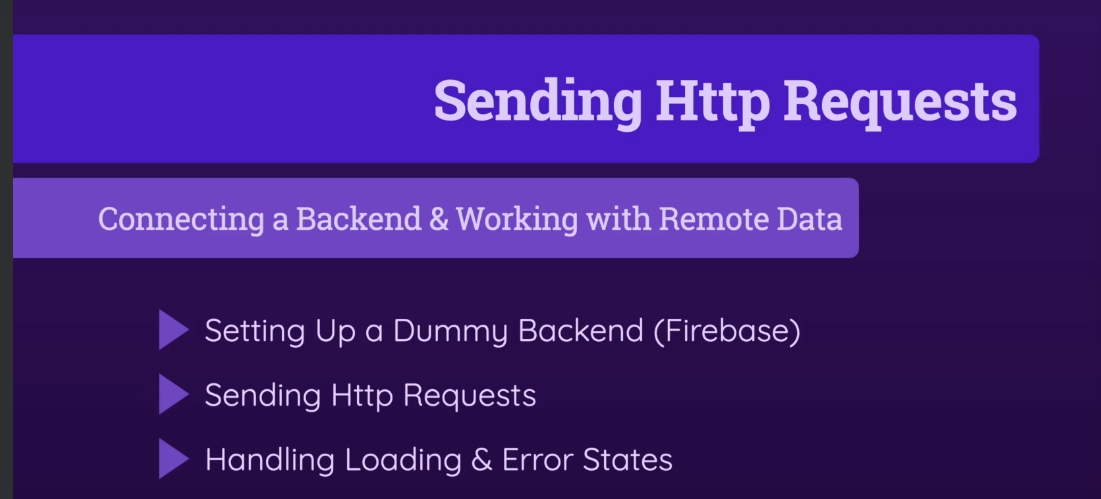
  );

};

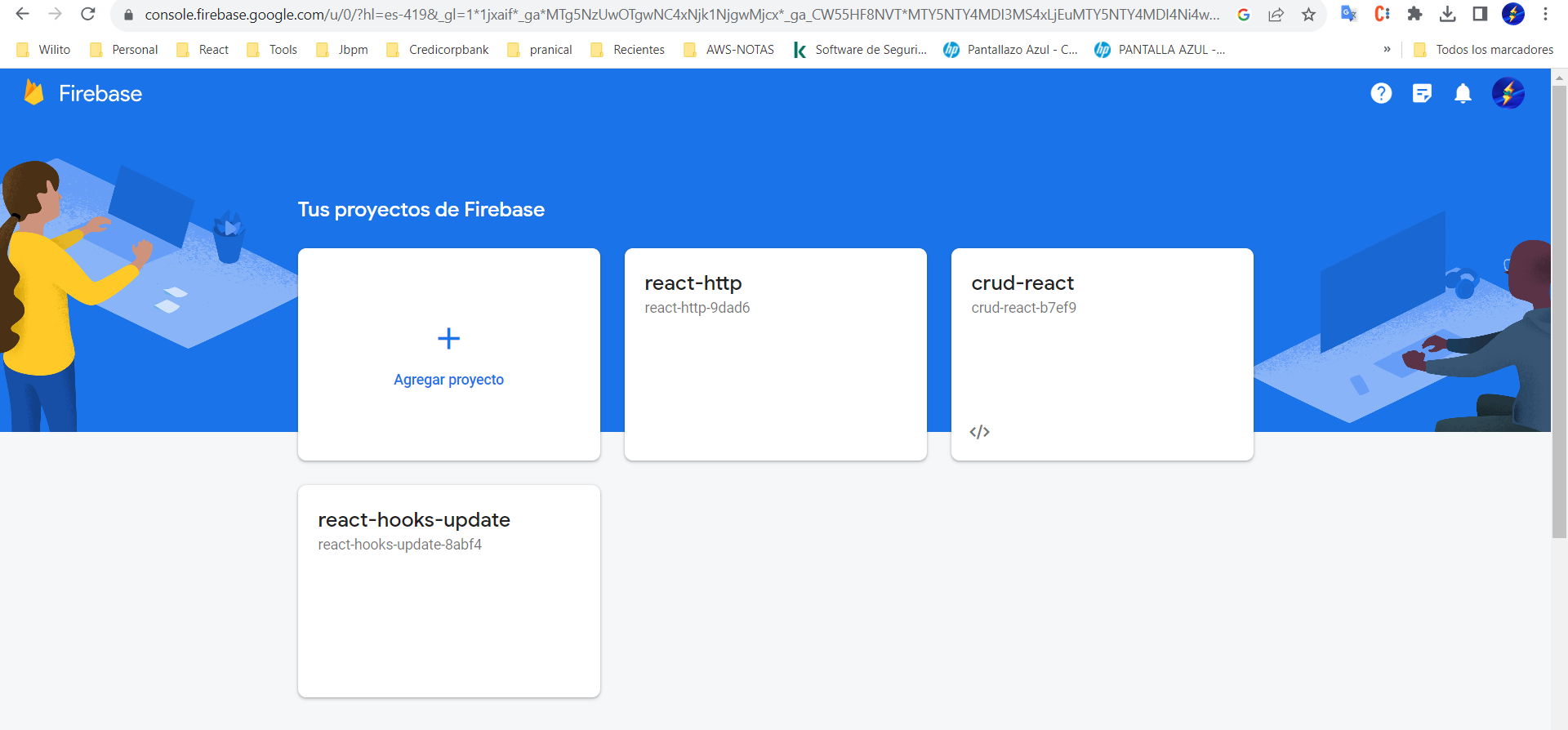
* Salida



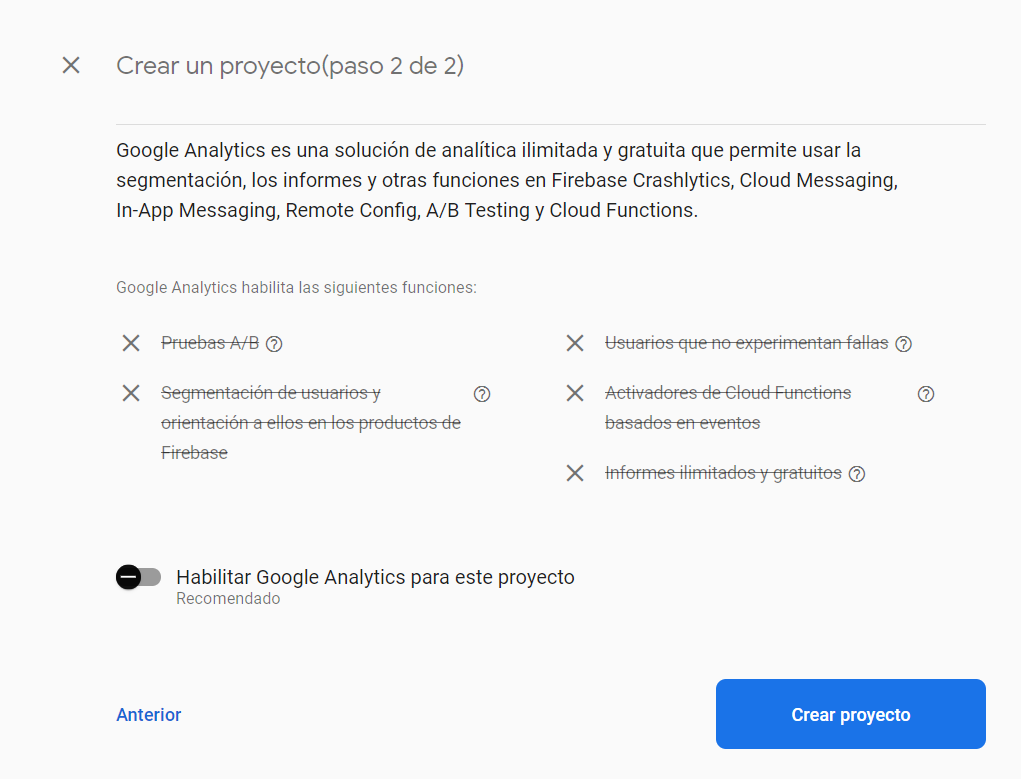
**Backend API para data remota**



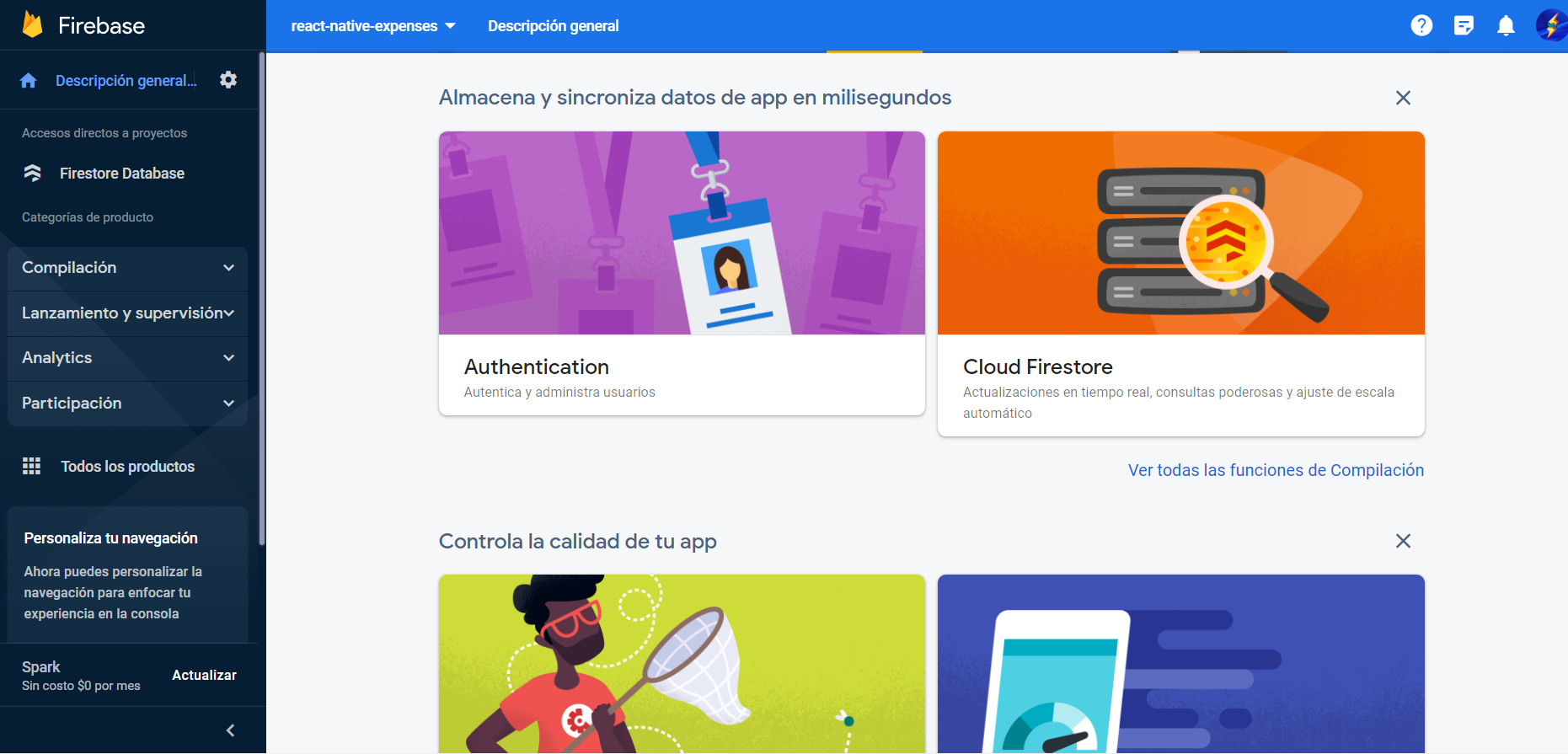
* Uso de **firebase**
* Comienza firebase

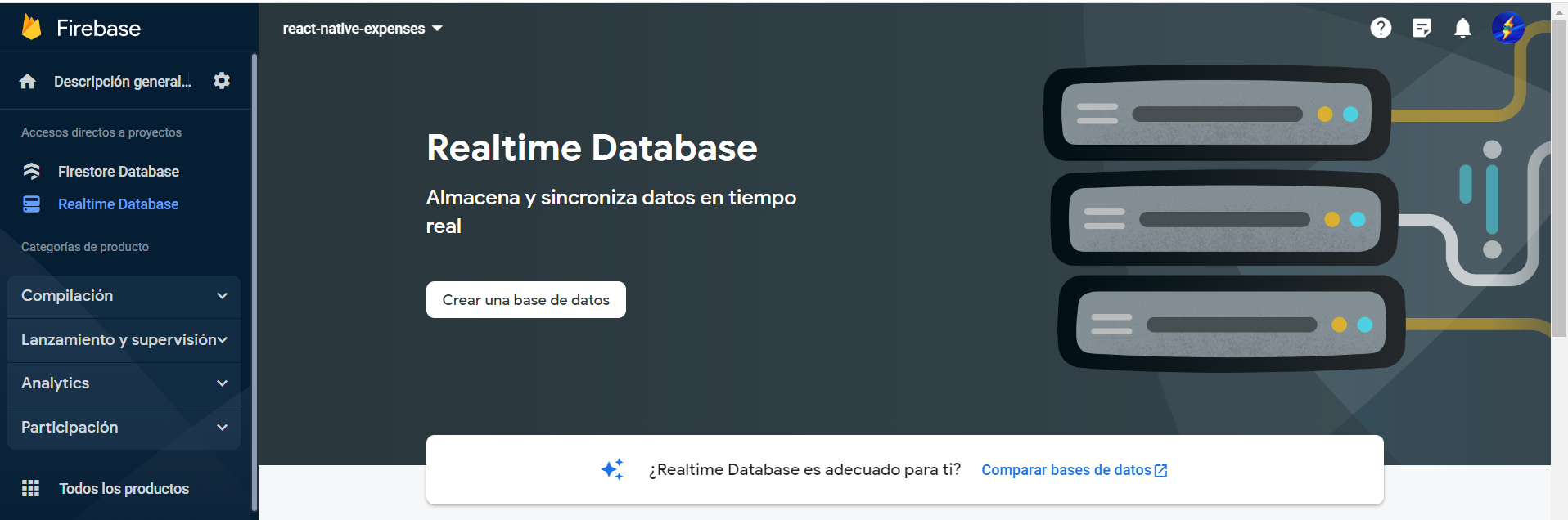


* Dar Nombre: react-native-experience

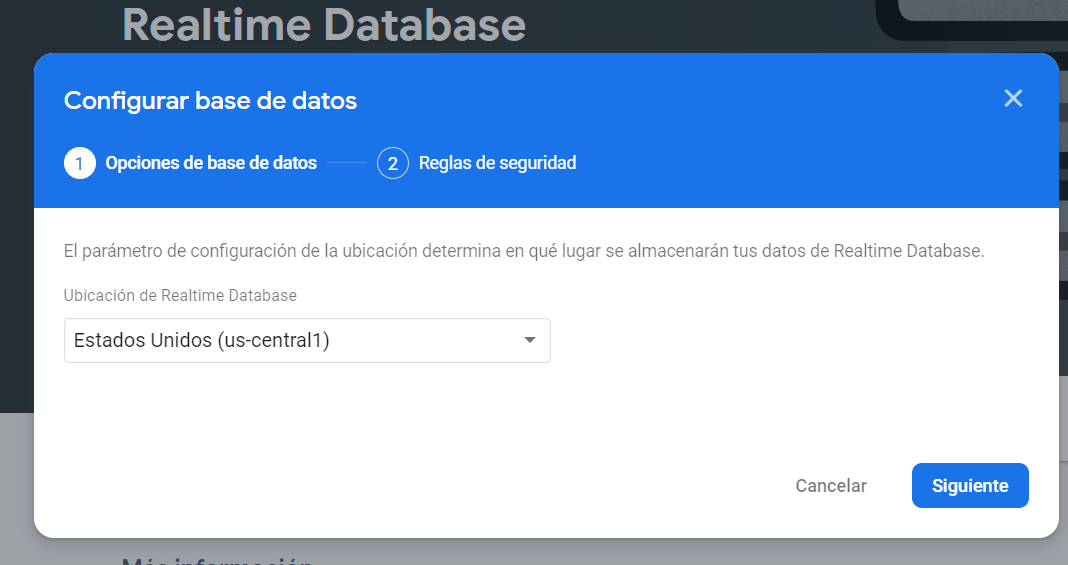


* Crear proyecto. Utilizaremos la base de datos en tiempo real. Detrás de escena se genera un API que facilita las pruebas

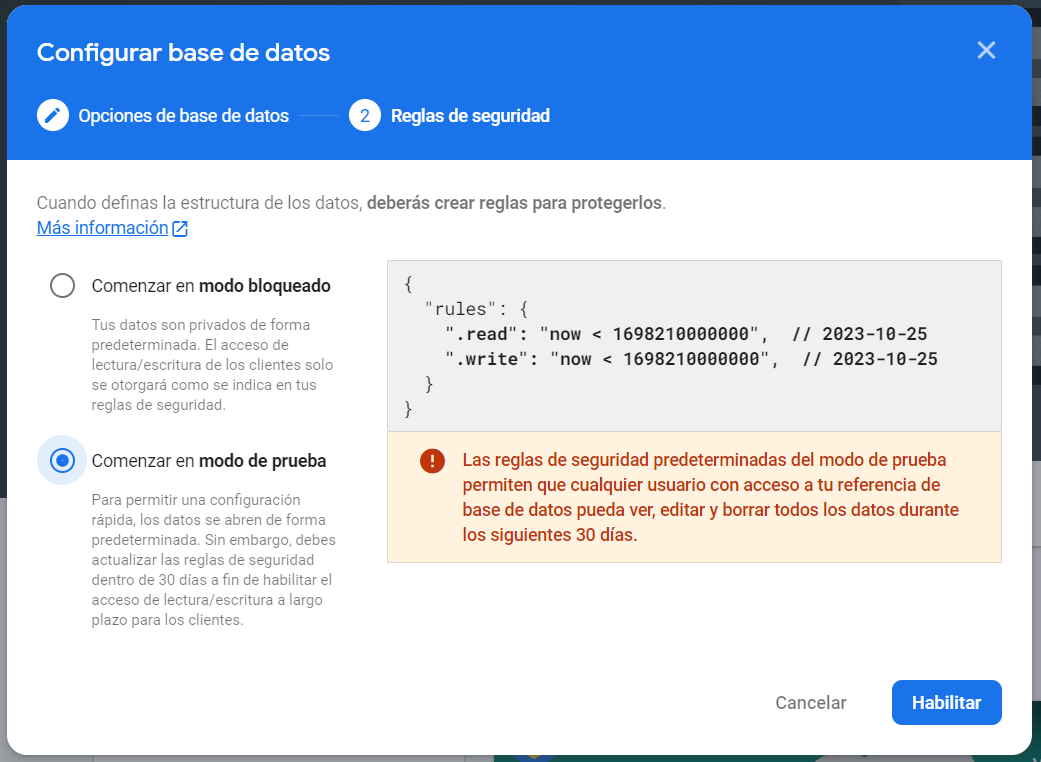




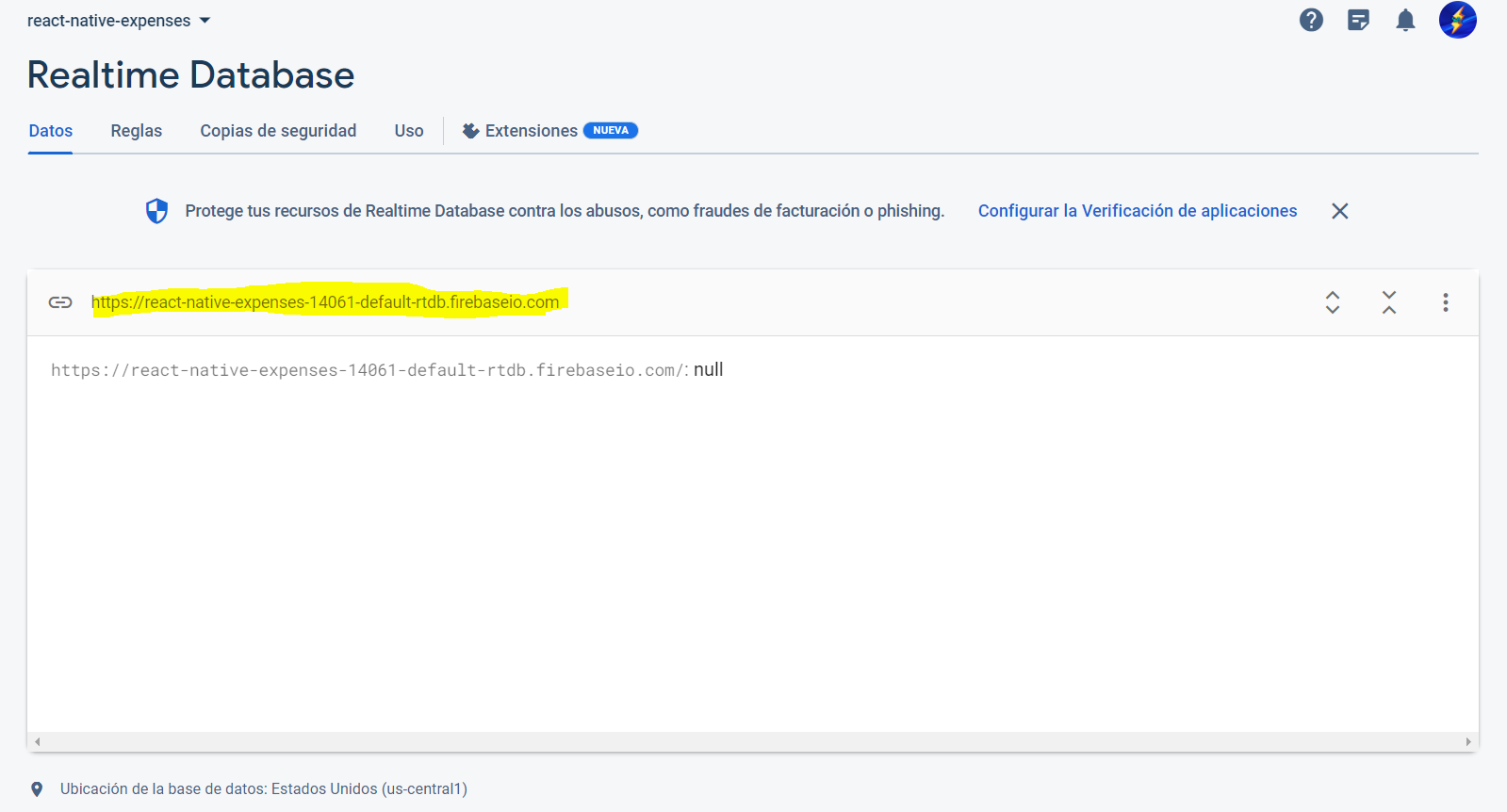
* Crear Base de datos



* Crearla en el modo prueba



* Crear la DB.



URL para peticiones a mi DB: [https://react-native-expenses-14061-default-rtdb.firebaseio.com](https://console.firebase.google.com/u/0/project/react-native-expenses-14061/database/react-native-expenses-14061-default-rtdb/data/~2F?hl=es-419)

* Podemos utilizar fetch para consumo de solicitudes o axios que es super compatible con React y React Native

> ..\A\_CURSOS\React-Native\code\RNExpenses> npm install axios

* Generar archivos para consumo de firebase
* import axios from "axios";
* export const storeExpenses = (expenseData) => {
* axios.post(
* "https://react-native-expenses-14061-default-rtdb.firebaseio.com/expenses.json",
* expenseData
* );
* };
* Se agrega a la ruta de firebase nombre.json (en este caso /expenses.json)
* Agregar Gastos

…

const confirmHandler = (expenseData) => {

    if (isEditing) {

      expensesCtx.updateExpense(editedExpenseId, expenseData);

    } else {

      storeExpenses(expenseData)

      expensesCtx.addExpense(expenseData);

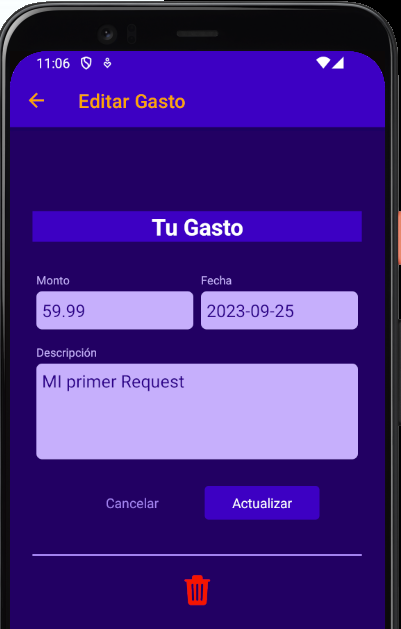
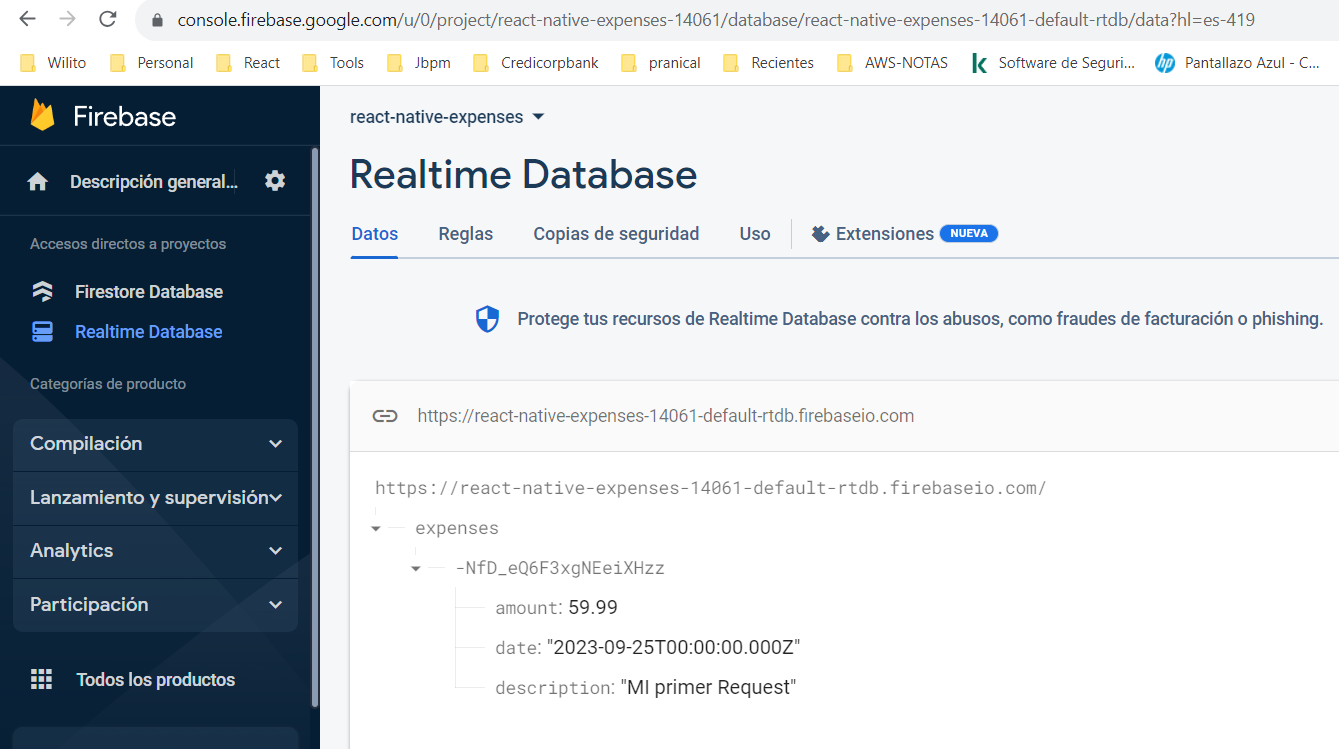
    }

    navigation.goBack();

  };

…

* Consultar la DB generada en firebase

* Uso de consultas **axios.get** se genera una promesa. Objeto que eventualmente dará acceso a otros Objetos. Uso de promesas async-await

export const fetchExpenses = async () => {

  const response = await axios.get(BACKEND\_URL + "/expenses.json");

  console.log("Datos de la consulta Firbase. Gastos: ", response);

  const expenses = [];

  const data = response.data;

  for (const key in data) {

    expenses.push({

      id: key,

      amount: data[key].amount,

      date: new Date(data[key].date),

      description: data[key].description,

    });

  }

* Consumo de función de consulta generado

export const fetchExpenses = async () => {

  const response = await axios.get(BACKEND\_URL + "/expenses.json");

  console.log("Datos de la consulta Firbase. Gastos: ", response);

  const expenses = [];

  const data = response.data;

  for (const key in data) {

    expenses.push({

      id: key,

      amount: data[key].amount,

      date: new Date(data[key].date),

      description: data[key].description,

    });

  }

* Ajustar el Context para actualizar la data a medida que genero y automáticamente consulto la data

export const ExpensesContext = createContext({

  expenses: [],

  addExpense: ({ description, amount, date }) => {},

  setExpenses: (expenses) => {},

  updateExpense: (id, { description, amount, date }) => {},

  deleteExpense: (id) => {},

});

const expensesReducer = (state, action) => {

  switch (action.type) {

    case "ADD":

      const id = new Date().toString() + Math.random().toString();

      return [{ ...action.payload, id: id }, ...state];

    case "SET":

      return action.payload;

    case "UPDATE":

…

 const setExpenses = (expenses) => {

    dispatch({ type: "SET", payload: expenses });

  };

  const updateExpense = (id, expenseData) => {

    dispatch({ type: "UPDATE", payload: { id: id, data: expenseData } });

  };

  const deleteExpense = (id) => {

    dispatch({ type: "DELETE", payload: id });

  };

  const value = {

    expenses: expensesState,

    addExpense: addExpense,

    setExpenses: setExpenses,

    updateExpense: updateExpense,

    deleteExpense: deleteExpense,

  };

…

* Genero y se actualiza la data a través del uso del Api de contexto de React

