REACT-NATIVE

**SEC-2: REFORZAMIENTO DE LAS BASES**

**INICIO DE PROYECTO – REACT CON TYPESCRIPT**

Crear el proyecto de react con typescript con el comando:

**npx create-react-app my-app --template typescript**

**npx react-native init AwesomeTSProject --template react-native-template-typescript**

Y a continuación, en el archivo **public/index.html** importar Bootstrap:

...

<link

      href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0-alpha3/dist/css/bootstrap.min.css"

      rel="stylesheet"

      integrity="sha384-KK94CHFLLe+nY2dmCWGMq91rCGa5gtU4mk92HdvYe+M/SXH301p5ILy+dN9+nJOZ"

      crossorigin="anonymous"

    />

<title>Reforzamiento TS - React</title>

...

**PRIMEROS FUNCTIONALS COMPONENTS EN TS**

Borramos todo lo que no utilicemos como el index.css, etc.

Creamos la carpeta **Typescript** dentro de la carpeta **src**, y creamos el archivo **TiposBasicos.tsx**.

Y dentro del archivo **index.html** le damos la **class=”container”** al body.

<body class="container">

**OBJETOS LITERALES E INTERFACES**

Creamos el archivo **typescript/ObjetosLiterales.tsx:**

interface Persona {

  nombre: string;

  edad: number;

  aísción: Direccion;

}

interface Direccion {

  aís: string;

  casaNo: number;

}

export const ObjetosLiterales = () => {

  const persona: Persona = {

    nombre: “Joaquín”,

    edad: 25,

    aísción: {

      aís: “Argentina”,

      casaNo: 420,

    },

  };

  return (

    <>

      <h3>Objetos Literales</h3>

      <code>

        <pre>{JSON.stringify(persona, null, 2)}</pre>

      </code>

    </>

  );

};

**FUNCIONES, RETORNO Y ARGUMENTOS**

Creamos el archivo **typecript/Funciones.tsx:**

export const Funciones = () => {

  const sumar = (a: number, b: number): number => {

    return a + b;

  };

  return (

    <>

      <h3>Funciones</h3>

      <span>El resultado es {sumar(1, 2)}</span>

    </>

  );

};

**HOOK – useState**

Creamos la carpeta **components** dentro de **src**, con el archivo **Contador.tsx**

import { useState } from "react";

export const Contador = () => {

  const [value, setValue] = useState(0);

  const acumular = (numero: number) => {

    setValue(value + numero);

  };

  return (

    <>

      <h3>

        Contador: <small>{value}</small>

      </h3>

      <button className="btn btn-primary" onClick={() => acumular(1)}>

        +1

      </button>

      &nbsp;

      <button className="btn btn-primary" onClick={() => acumular(-1)}>

        -1

      </button>

    </>

  );

};

**CUSTOM HOOK – useCounter**

Creamos la carpeta **hooks** con el archivo **useCounter.tsx** en donde va a recibir toda la lógica del useState que utilizamos en el componente **Contador.tsx**:

import { useState } from "react";

export const useCounter = (inicial: number = 10) => {

  const [value, setValue] = useState(inicial);

  const acumular = (numero: number) => {

    setValue(value + numero);

  };

  return {

    value,

    acumular,

  };

};

Y ahora en la carpeta **components** creamos el archivo **ContadorConHook.tsx** e importamos este **useCounter.tsx:**

import { useCounter } from "../hooks/useCounter";

export const ContadorConHook = () => {

  const { value, acumular } = useCounter(0);

  return (

    <>

      <h3>

        Contador con hook: <small>{value}</small>

      </h3>

      <button className="btn btn-primary" onClick={() => acumular(1)}>

        +1

      </button>

      &nbsp;

      <button className="btn btn-primary" onClick={() => acumular(-1)}>

        -1

      </button>

    </>

  );

};

**COMPONENTE LOGIN – ANTESALA AL useReducer**

Creamos el componente **components/Login.tsx**:

export const Login = () => {

  return (

    <>

      <h3>Login</h3>

      <div className="alert alert-info">Validando...</div>

      <div className="alert alert-danger">No autenticado</div>

      <div className="alert alert-success">Autenticado</div>

      <button className="btn btn-primary">Login</button>

      <button className="btn btn-danger">Log out</button>

    </>

  );

};

**HOOK – useReducer**

Ahora importamos dentro del componente **Login.tsx** el **useReducer** y creamos la **interface** para el state, el **initialState**, el **type** para el action y la **función** para ejecutar en el useReducer:

import { useReducer } from "react";

interface AuthState {

  validando: boolean;

  token: string | null;

  userName: string;

  name: string;

}

const initialState: AuthState = {

  validando: true,

  token: null,

  userName: "",

  name: "",

};

type AuthAction = {

  type: "logout";

};

const authReducer = (state: AuthState, action: AuthAction): AuthState => {

  switch (action.type) {

    case "logout":

      return { validando: true, token: null, userName: "", name: "" };

    default:

      return state;

  }

};

export const Login = () => {

  const [state, dispatch] = useReducer(authReducer, initialState);

  return (

    <>

      <h3>Login</h3>

      <div className="alert alert-info">Validando...</div>

      <div className="alert alert-danger">No autenticado</div>

      <div className="alert alert-success">Autenticado</div>

      <button className="btn btn-primary">Login</button>

      <button className="btn btn-danger">Log out</button>

    </>

  );

};

**AUTH-REDUCER**

Implementamos la lógica para que dependiendo el valor de **validando** en el state, muestre una u otra cosa mediante un **useEffect** el cual después de 1 segundo y medio ejecuta el **dispatch** del reducer:

import { useEffect, useReducer } from "react";

interface AuthState {

  validando: boolean;

  token: string | null;

  userName: string;

  name: string;

}

const initialState: AuthState = {

  validando: true,

  token: null,

  userName: "",

  name: "",

};

type AuthAction = {

  type: "logout";

};

const authReducer = (state: AuthState, action: AuthAction): AuthState => {

  switch (action.type) {

    case "logout":

      return { validando: false, token: null, userName: "", name: "" };

    default:

      return state;

  }

};

export const Login = () => {

  const [state, dispatch] = useReducer(authReducer, initialState);

  useEffect(() => {

    setTimeout(() => {

      dispatch({ type: "logout" });

    }, 1500);

  }, []);

  if (state.validando) {

    return (

      <>

        <h3>Login</h3>

        <div className="alert alert-info">Validando...</div>

      </>

    );

  }

  return (

    <>

      <h3>Login</h3>

      <div className="alert alert-danger">No autenticado</div>

      <div className="alert alert-success">Autenticado</div>

      <button className="btn btn-primary">Login</button>

      <button className="btn btn-danger">Log out</button>

    </>

  );

};

**LOGIN y LOGOUT**

Creamos las funciones de Login y Logout para pasarle a los button y ejecutamos el payload que tiene un tipado específico de lo que tiene que recibir para logearse:

import { useEffect, useReducer } from "react";

interface AuthState {

  validando: boolean;

  token: string | null;

  userName: string;

  name: string;

}

const initialState: AuthState = {

  validando: true,

  token: null,

  userName: "",

  name: "",

};

type LoginPayload = {

  userName: string;

  name: string;

};

type AuthAction = { type: "logout" } | { type: "login"; payload: LoginPayload };

const authReducer = (state: AuthState, action: AuthAction): AuthState => {

  switch (action.type) {

    case "logout":

      return { validando: false, token: null, userName: "", name: "" };

    case "login":

      const { name, userName } = action.payload;

      return {

        validando: false,

        token: "abc123",

        userName,

        name,

      };

    default:

      return state;

  }

};

export const Login = () => {

  const [state, dispatch] = useReducer(authReducer, initialState);

  useEffect(() => {

    setTimeout(() => {

      dispatch({ type: "logout" });

    }, 1500);

  }, []);

  const login = () => {

    dispatch({

      type: "login",

      payload: { userName: "joaquincaggiano", name: "Joaquín" },

    });

  };

  const logout = () => {

    dispatch({ type: "logout" });

  };

  if (state.validando) {

    return (

      <>

        <h3>Login</h3>

        <div className="alert alert-info">Validando...</div>

      </>

    );

  }

  return (

    <>

      <h3>Login</h3>

      {state.token ? (

        <>

          <div className="alert alert-success">

            Autenticado com: {state.name}

          </div>

          <button className="btn btn-danger" onClick={logout}>

            Log out

          </button>

        </>

      ) : (

        <>

          <div className="alert alert-danger">No autenticado</div>

          <button className="btn btn-primary" onClick={login}>

            Login

          </button>

        </>

      )}

    </>

  );

};

**PETICIONES HHTP – AXIOS**

Creamos el componente **components/Usuarios.tsx** y la carpeta **api/reqRes.tsx** la cual contendrá el llamado a la api que vamos a utilizar para traer usuarios.

**reqRes.tsx:**

import axios from "axios";

export const reqResApi = axios.create({

    baseURL: "https://reqres.in/api"

});

**Usuarios.tsx:**

import { useEffect } from "react";

import { reqResApi } from "../api/reqRes";

export const Usuarios = () => {

  useEffect(() => {

    reqResApi

      .get("/users")

      .then((resp) => {

        console.log(resp.data.data);

      })

      .catch((err) => console.log(err));

  }, []);

  return (

    <>

      <h3>Usuarios</h3>

      <table className="table">

        <thead>

          <tr>

            <th>Avatar</th>

            <th>Nombre</th>

            <th>Email</th>

          </tr>

        </thead>

        <tbody></tbody>

      </table>

    </>

  );

};

**ESTABLECER EL TIPO DE LAS RESPUESTAS HTTP**

Vamos a crear la carpeta **interfaces** con el archivo **reqRes.tsx** en donde va a tener la interface con el tipado especifico de la respuesta http que estamos utilizando en el componente **Usuarios.tsx**.

Para hacer el tipado más rápido vamos a utilizar la página <https://app.quicktype.io/> en la cual debemos copiar y pegar la respuesta JSON y nos hará el tipado automáticamente.

**reqRes.tsx:**

export interface ReqResListado {

    page:        number;

    per\_page:    number;

    total:       number;

    total\_pages: number;

    data:        User[];

    support:     Support;

}

export interface User {

    id:         number;

    email:      string;

    first\_name: string;

    last\_name:  string;

    avatar:     string;

}

export interface Support {

    url:  string;

    text: string;

}

**Usuarios.tsx**: importamos esta interface y le decimos al axios.get que es de este tipo la respuesta. Además creamos un **state** y le decimos que es de tipo **User[]**.

import { useEffect, useState } from "react";

import { reqResApi } from "../api/reqRes";

import { ReqResListado, User } from "../interfaces/reqRes";

export const Usuarios = () => {

    const [users, setUsers] = useState<User[]>([])

  useEffect(() => {

    reqResApi

      .get<ReqResListado>("/users")

      .then((resp) => {

        setUsers(resp.data.data)

      })

      .catch((err) => console.log(err));

  }, []);

  return (

    <>

      <h3>Usuarios</h3>

      <table className="table">

        <thead>

          <tr>

            <th>Avatar</th>

            <th>Nombre</th>

            <th>Email</th>

          </tr>

        </thead>

        <tbody></tbody>

      </table>

    </>

  );

};

**MOSTRAR USUARIOS EN PANTALLA**

Ahora que ya tenemos los users en el state con el tipado especificado podemos mostrarlos en pantalla. Para eso vamos a crear una función que renderice un elemento JSX con los datos de cada usuario:

import { useEffect, useState } from "react";

import { reqResApi } from "../api/reqRes";

import { ReqResListado, User } from "../interfaces/reqRes";

export const Usuarios = () => {

  const [users, setUsers] = useState<User[]>([]);

  useEffect(() => {

    reqResApi

      .get<ReqResListado>("/users")

      .then((resp) => {

        setUsers(resp.data.data);

      })

      .catch((err) => console.log(err));

  }, []);

  const renderItem = (user: User) => {

    return (

      <tr key={user.id.toString()}>

        <th>

          <img

            src={user.avatar}

            alt={user.first\_name}

            style={{ width: "50px", borderRadius: "100px" }}

          />

        </th>

        <th>

          {user.first\_name} {user.last\_name}

        </th>

        <th>{user.email}</th>

      </tr>

    );

  };

  return (

    <>

      <h3>Usuarios</h3>

      <table className="table">

        <thead>

          <tr>

            <th>Avatar</th>

            <th>Nombre</th>

            <th>Email</th>

          </tr>

        </thead>

        <tbody>{users.map((user) => renderItem(user))}</tbody>

      </table>

      <button className="btn btn-primary">Next</button>

    </>

  );

};

**CREAR UNA PEQUEÑA PAGINACIÓN**

Creamos la función **loadUsers** la cual le pasamos como parámetro la **page**, el cual va a tener el valor de un useRef que creamos con el valor de 1.

import { useEffect, useRef, useState } from "react";

import { reqResApi } from "../api/reqRes";

import { ReqResListado, User } from "../interfaces/reqRes";

export const Usuarios = () => {

  const [users, setUsers] = useState<User[]>([]);

  const pageRef = useRef(1)

  useEffect(() => {

    loadUsers();

  }, []);

  const loadUsers = async () => {

    const resp = await reqResApi.get<ReqResListado>("/users", {

        params: {

            page: pageRef.current

        }

    });

    if(resp.data.data.length > 0) {

        setUsers(resp.data.data);

        pageRef.current ++;

    } else {

        alert("No hay más registros")

    }

  };

  const renderItem = (user: User) => {

    return (

      <tr key={user.id.toString()}>

        <th>

          <img

            src={user.avatar}

            alt={user.first\_name}

            style={{ width: "50px", borderRadius: "100px" }}

          />

        </th>

        <th>

          {user.first\_name} {user.last\_name}

        </th>

        <th>{user.email}</th>

      </tr>

    );

  };

  return (

    <>

      <h3>Usuarios</h3>

      <table className="table">

        <thead>

          <tr>

            <th>Avatar</th>

            <th>Nombre</th>

            <th>Email</th>

          </tr>

        </thead>

        <tbody>{users.map((user) => renderItem(user))}</tbody>

      </table>

      <button className="btn btn-primary" onClick={loadUsers}>

        Next

      </button>

    </>

  );

};

**CUSTOM HOOK – useUsers**

Creamos dentro de la carpeta **hooks** el archivo **useUser.tsx**:

import { useEffect, useRef, useState } from "react";

import { reqResApi } from "../api/reqRes";

import { ReqResListado, User } from "../interfaces/reqRes";

export const useUser = () => {

  const [users, setUsers] = useState<User[]>([]);

  const pageRef = useRef(1);

  useEffect(() => {

    loadUsers();

  }, []);

  const loadUsers = async () => {

    const resp = await reqResApi.get<ReqResListado>("/users", {

      params: {

        page: pageRef.current,

      },

    });

    if (resp.data.data.length > 0) {

      setUsers(resp.data.data);

      pageRef.current++;

    } else {

      alert("No hay más registros");

    }

  };

  return {

    users,

    loadUsers,

  };

};

Y ahora en el componente **Usuarios.tsx** lo importamos y desestructuramos la información que necesitamos:

import { useUser } from "../hooks/useUser";

export const Usuarios = () => {

  const { users, loadUsers } = useUser();

...

**SIGUIENTES Y ANTERIORES REGISTROS – PAGINACIÓN**

Ya no vamos a utilizar la función **loadUsers()** en el componente, sino que vamos a crear dos funciones, una para ir a la página siguiente y otra para volver de página.

**useUser.tsx:**

import { useEffect, useRef, useState } from "react";

import { reqResApi } from "../api/reqRes";

import { ReqResListado, User } from "../interfaces/reqRes";

export const useUser = () => {

  const [users, setUsers] = useState<User[]>([]);

  const pageRef = useRef(1);

  useEffect(() => {

    loadUsers();

  }, []);

  const loadUsers = async () => {

    const resp = await reqResApi.get<ReqResListado>("/users", {

      params: {

        page: pageRef.current,

      },

    });

    if (resp.data.data.length > 0) {

      setUsers(resp.data.data);

    } else {

      pageRef.current--;

      alert("No hay más registros");

    }

  };

  const nextPage = () => {

    pageRef.current++;

    loadUsers();

  };

  const prevPage = () => {

    if (pageRef.current > 1) {

      pageRef.current--;

      loadUsers();

    }

  };

  return {

    users,

    nextPage,

    prevPage,

  };

};

**Usuarios.tsx:**

...

const { users, nextPage, prevPage } = useUser();

...

<button className="btn btn-primary" onClick={prevPage}>

  Prev

</button>

&nbsp;

<button className="btn btn-primary" onClick={nextPage}>

  Next

</button>

**FORMULARIOS**

Creamos el componente **Formularios.tsx** el cual va a tener dos inputs donde se manejan sus value con un onChange:

import { useState } from "react";

export const Formularios = () => {

  const [form, setForm] = useState({

    email: "test@gmail.com",

    password: "123456",

  });

  const onChange = (value: string, campo: string) => {

    setForm({

      ...form,

      [campo]: value,

    });

  };

  return (

    <>

      <h3>Formularios</h3>

      <input

        type="text"

        className="form-control"

        placeholder="Email"

        value={form.email}

        onChange={(e) => onChange(e.target.value, "email")}

      />

      <input

        type="password"

        className="form-control my-2"

        placeholder="Password"

        value={form.password}

        onChange={(e) => onChange(e.target.value, "password")}

      />

    </>

  );

};

**CUSTOM HOOK – useForm<Genéricos>**

Creamos dentro de la carpeta **hooks** el archivo **useForm.tsx**, el cual va a recibir un formulario que va a ser un objeto de tipo genérico que va a recibir como parámetro.

import { useState } from "react";

export const useForm = <T extends Object>(formulario: T) => {

  const [state, setState] = useState(formulario);

  const onChange = (value: string, campo: keyof T) => {

    setState({

      ...state,

      [campo]: value,

    });

  };

  return {

    state,

    onChange,

  };

};

Y ahora en el componente **Formulario.tsx** es donde mandamos ese objeto:

import { useForm } from "../hooks/useForm";

export const Formularios = () => {

  const {state, onChange} = useForm({

    email: "test@gmail.com",

    password: "123456",

  });

  return (

    <>

      <h3>Formularios</h3>

      <input

        type="text"

        className="form-control"

        placeholder="Email"

        value={state.email}

        onChange={(e) => onChange(e.target.value, "email")}

      />

      <input

        type="password"

        className="form-control my-2"

        placeholder="Password"

        value={state.password}

        onChange={(e) => onChange(e.target.value, "password")}

      />

    </>

  );

};

**SEC-3: INSTALACIÓN Y CONFIGURACIÓN DE REACT NATIVE CON EMULADORES Y DISPOSITIVOS FÍSICOS**

**DIFERENCIA ENTRE EXPO CLI y REACT NATIVE CLI**

|  |  |
| --- | --- |
| **Expo CLI** | **React Native CLI** |
| Expo se encarga de controlar los builds para IOS y Android. | Genera un proyecto con lo mínimo necesario para correr. |
| Provee muchas funcionalidades nativas incluidas. | Genera los proyectos nativos para IOS y Android de forma independiente. |
| Es excelente para alguien que empieza el desarrollo móvil. | Te permite escribir código nativo por plataforma (en caso de ser necesario). |
| Permite correr y probar la aplicación en IOS aunque no dispongas de una MAC. |  |

**Expo CLI:** usa el **EJECT** en caso de que un módulo nativo no sea soportado por Expo SDK.

NO HAGAS UN EJECT SI:

* Lo que deseas es distribuir en las App stores.
* Estás inconforme con el código nativo.
* Disfrutas de las actualizaciones que vienen con Expo.
* Si usas Expo Push Notificacion Services.
* Dependes de la comunidad de Expo.

Contras:

|  |  |
| --- | --- |
| **Expo CLI** | **React Native CLI** |
| No puedes extender una funcionalidad nativa no soportada por expo (a menos que uses el EJECT) | Necesitas los ambientes de desarrollo por separado. Android Studio (las herramientas al menos) y Xcode. |

Pros comunes en ambos:

* La mayoría de los paquetes soportan ambos ambientes con instalaciones independientes.
* Cambios en caliente (hot reload).
* Desarrollar y ver cambios en desarrollo en dispositivos físicos.
* Utilizar el conocimiento que tienes de React en React Native.
* Desplegar en las App stores.

**WINDOWS: INSTALACIONES NECESARIAS**

Ir a la documentación en:

<https://reactnative.dev/docs/environment-setup?guide=native>

Luego clickear en **Chocolatey** y en la powershell correr los comandos:

* Get-ExecutionPolicy.
* Si retorna **Restricted**, correr el comando:
* **Set-ExecutionPolicy AllSigned**
* Si no dice Restricted, ejecutamos el comando:
* **Set-ExecutionPolicy Bypass -Scope Process**

Una vez finalizado esos comandos, copiamos y pegamos la siguiente línea:

Set-ExecutionPolicy Bypass -Scope Process -Force; [System.Net.ServicePointManager]::SecurityProtocol = [System.Net.ServicePointManager]::SecurityProtocol -bor 3072; iex ((New-Object System.Net.WebClient).DownloadString('https://community.chocolatey.org/install.ps1'))

MIRAR LA DOCUMENTACIÓN ACÁ POR LAS DUDAS: <https://chocolatey.org/install>

Una vez instalado todo, cerramos la powershell y la volvemos a abrir, y pegamos el comando que aparece en la documentación de React Native:

**choco install -y nodejs-lts microsoft-openjdk11**

**……. NO CONTINUE PORQUE NO ME ANDABA BIEN EN WINDOWS**

**MAC OSX: INSTALACIONES NECESARIAS – ANDROID**

Vamos a la documentación oficial en: <https://reactnative.dev/docs/environment-setup?guide=native&platform=android>

Seleccionamos en Development **MacOs** y en **Target** Android.

Luego si no tenemos instalado **Homebrew**, vamos a <https://brew.sh/> y copiamos y pegamos el comando que nos aparece en la consola.

Por ejemplo este:

/bin/bash -c "$(curl -fsSL <https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh>)"

Una vez terminado la instalación, verificamos que con el comando **brew –version** nos de una respuesta afirmativa, y seguimos.

Ahora corremos el comando **brew install watchman**.

Y una vez finalizado eso corremos los comandos:

* brew tap homebrew/cask-versions
* brew install --cask zulu11
* brew info --cask zulu11

Luego ya debemos abrir **Android Studio** e ir a **SDK Manager** y verificar en la sección SDK Platforms si tenemos descargado (en el video utiliza el Android 10) Android 13.

Luego clickear en **Show package details** (esta debajo a la derecha) y verificar que tengamos descargado **Android SDK platform 33** y **Google APIs ARM 64 v8a System Image**.

Ahora debemos configurar variables de entorno, por lo que abrimos el **Finder** y clickeamos en nuestro usuario. A continuación, apretamos **command + shift + .** para abrir los archivos ocultos y buscamos el archivo **zshrc** o el archivo **zprofile**, y copiamos las siguientes líneas de código:

export ANDROID\_HOME="$HOME/Library/Android/sdk"

export PATH="$PATH:$ANDROID\_HOME/emulator"

export PATH="$PATH:$ANDROID\_HOME/platform-tools"

**MAC OSX: EMULADOR DE ANDROID**

Vamos a **Android Studio** y clickeamos en **Virtual Device Manager>Create device**.

Creamos un device a nuestro placer o elegimos algunos de los que están por defecto.

Y luego ya está, le damos next hasta que abra el emulador.

**INSTALACIONES NECESARIAS PARA IOS**

Vamos a la documentación oficial de React Native:

<https://reactnative.dev/docs/environment-setup?platform=ios>

Y elegimos Macos e IOs, luego instalar node y watchman, como dice la documentación.

A continuación, abrir XCode y vamos en el menú de arriba de todo en la mac a **settings** del XCode.

Luego seleccionamos **Locations** y verificar que command line tools corresponda la versión con la de xcode.

Ahora debemos instalar cocoapods con el comando: **brew install cocoapods**

**CREAR PROYECTO DE REACT NATIVE**

Vamos a crear el proyecto con el comando:

**npx react-native init MiProyecto --template react-native-template-typescript**

Una vez terminado la descarga abrir el emulador de Android y correr el comando dentro de la carpeta del proyecto:

**npx react-native run-android**

Este comando nos va abrir otra terminal la cual se llama **Metro**, si presionamos la tecla **r**, reinicia la aplicación.

Luego abrimos la carpeta del proyecto en VSC y en el **App.tsx** borramos todo y creamos el siguiente componente:

import React from 'react'

import { Text, View } from 'react-native'

const App = () => {

return (

<View>

<Text>Hola Mundo</Text>

</View>

)

}

export default App;

**CORRER SIMULADOR DE IOS**

Dentro de la carpeta del Proyecto corremos el comando: **npx react-native run-ios**

Además si queremos correr el proyecto con un emulador determinado agregamos al comando de arriba las siguientes opciones:

* --simulator=”iPhone 8”
* --simulator=”iPhone 8 Plus”
* --simulator=”iPhone 11”
* --simulator=”iPhone 11 Pro”
* --simulator=”iPhone 11 Pro Max”
* --simulator=”iPhone 12”
* --simulator=”iPhone 12 Pro”
* --simulator=”iPhone 12 Pro Max”
* Etc.

El comando para ver la listas de dispositivos es:

**xcrun simctl list devices**

**SEC-4: MI PRIMER APP EN REACT NATIVE – COUNTER APP**

**DESHABILITAR PRETTIER**

Nos dirigimos al archivo **.eslintrc.js** y agregamos la siguiente línea en el **module.exports:**

rules: {

'prettier/prettier': 0

}

**HOLA MUNDO**

En el **App.tsx** borrar todo y escribir lo siguiente:

import React from 'react';

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

const App = () => {

return (

<View

style={{

flex: 1,

justifyContent: 'center',

}}>

<Text style={{

fontSize: 45,

textAlign: 'center'

}}>

Hola Mundo

</Text>

</View>

);

};

export default App;

En React Native no temenos un <div> o un <p>, pero tenemos los componentes **<View>** y **<Text>**.

**CREAR PANTALLAS INDEPENDIENTES**

En la raíz del proyecto creamos la carpeta **src** y dentro de la misma, las carpeta **screens** y **components**.

Ahora dentro de la carpeta **screens** creamos la pantalla **HolaMundoScreen.tsx**, la cual va a contener todo el código que estaba en el **App.tsx**, y luego importamos esta screen en el App.tsx.

**CREAR UN CONTADOR**

En la carpeta **screens** creamos el archivo **ContadorScreen.tsx**, e implementamos la lógica para mostrar dos botones que aumenten o disminuyan el contador:

import React, {useState} from 'react';

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

export const ContadorScreen = () => {

const [counter, setCounter] = useState(0);

return (

<View style={{flex: 1, justifyContent: 'center'}}>

<Text

style={{

textAlign: 'center',

fontSize: 40,

}}>

Contador: {counter}

</Text>

<Button title="Aumentar" onPress={() => setCounter(counter + 1)} />

<Button title="Decrementar" onPress={() => setCounter(counter - 1)} />

</View>

);

};

Acá está la documentación de los componentes que podemos usar de React-Native:

<https://reactnative.dev/docs/components-and-apis>

**TOUCHABLE OPACITY**

La documentación recomienda no usar tanto el componente Button, en cambio, debemos usar el **TouchableOpacity**:

import React, {useState} from 'react';

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

export const ContadorScreen = () => {

const [counter, setCounter] = useState(0);

return (

<View style={{flex: 1, justifyContent: 'center'}}>

<Text

style={{

textAlign: 'center',

fontSize: 40,

}}>

Contador: {counter}

</Text>

<TouchableOpacity onPress={() => setCounter(counter + 1)}>

<View>

<Text>+1</Text>

</View>

</TouchableOpacity>

</View>

);

};

**STYLE-SHEET**

En React Native se acostumbra a importar **StyleSheet** y crear un objeto el cual va a contener los estilos de los componentes que utilicemos:

import React, {useState} from 'react';

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

export const ContadorScreen = () => {

const [counter, setCounter] = useState(0);

return (

<View style={styles.container}>

<Text style={styles.title}>Contador: {counter}</Text>

<TouchableOpacity onPress={() => setCounter(counter + 1)}>

<View>

<Text>+1</Text>

</View>

</TouchableOpacity>

</View>

);

};

const styles = StyleSheet.create({

container: {

flex: 1,

justifyContent: 'center',

},

title: {

textAlign: 'center',

fontSize: 40,

},

});

**BUTTON PERSONALIZADO FLOTANTE**

Vamos hacer que los TouchableOpacity queden debajo de todo, uno a la derecha y el otro a la izquierda:

import React, {useState} from 'react';

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

export const ContadorScreen = () => {

const [counter, setCounter] = useState(0);

return (

<View style={styles.container}>

<Text style={styles.title}>Contador: {counter}</Text>

<TouchableOpacity

style={styles.fabLocationRight}

onPress={() => setCounter(counter + 1)}>

<View style={styles.fab}>

<Text style={styles.fabText}>+1</Text>

</View>

</TouchableOpacity>

<TouchableOpacity

style={styles.fabLocationLeft}

onPress={() => setCounter(counter - 1)}>

<View style={styles.fab}>

<Text style={styles.fabText}>-1</Text>

</View>

</TouchableOpacity>

</View>

);

};

const styles = StyleSheet.create({

container: {

flex: 1,

justifyContent: 'center',

},

title: {

textAlign: 'center',

fontSize: 40,

},

fabLocationRight: {

position: 'absolute',

bottom: 25,

right: 25,

},

fabLocationLeft: {

position: 'absolute',

bottom: 25,

left: 25,

},

fab: {

backgroundColor: '#5856D6',

width: 60,

height: 60,

borderRadius: 100,

justifyContent: 'center',

},

fabText: {

color: 'white',

fontSize: 25,

fontWeight: 'bold',

alignSelf: 'center',

},

});

**COMPONENTE PERSONALIZADO: FAB – FLOATING ACTION BUTTON**

En la carpeta **components** creamos el archivo **Fab.tsx**, el cual va a ser nuestro botón reutilizable:

import React from 'react';

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

interface Props {

title: string;

}

export const Fab = ({title}: Props) => {

return (

<TouchableOpacity

style={styles.fabLocationRight}

onPress={() => console.log('Click')}>

<View style={styles.fab}>

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

</View>

</TouchableOpacity>

);

};

const styles = StyleSheet.create({

fabLocationRight: {

position: 'absolute',

bottom: 25,

right: 25,

},

fabLocationLeft: {

position: 'absolute',

bottom: 25,

left: 25,

},

fab: {

backgroundColor: '#5856D6',

width: 60,

height: 60,

borderRadius: 100,

justifyContent: 'center',

},

fabText: {

color: 'white',

fontSize: 25,

fontWeight: 'bold',

alignSelf: 'center',

},

});

Como vemos estamos diciendole que va a recibir **props** y además le decimos que tipado tienen esas Props.

En el **screen** ContadorScreen.tsx importamos este componente y le pasamos la prop title:

…

<Fab title='+1' />

…

**ESTILO CONDICIONAL**

Debemos hacer ahora que un botón este a la derecha y el otro la izquierda, por lo que le mandamos en Props la propiedad **position**, y le especificamos en la interface que puede ser **‘bl’** o **‘br’**, y que por defecto es br.

Además ya no usamos el componente TouchableOpacity, en vez de eso usamos **TouchableNativeFeedback**, el cual tiene una propiedad **background** la cual produce el efecto de un shadow cuando clickeamos en el botón (por ahora solo nos va a servir en Android, ver el próximo video para ver cómo se soluciona en IOS).

**Fab.tsx:**

import React from 'react';

import {

StyleSheet,

Text,

TouchableNativeFeedback,

TouchableOpacity,

View,

} from 'react-native';

interface Props {

title: string;

position?: 'br' | 'bl';

onPress: () => void;

}

export const Fab = ({title, onPress, position = 'br'}: Props) => {

return (

<View

style={[

styles.fabLocation,

position === 'br' ? styles.right : styles.left,

]}>

<TouchableNativeFeedback

onPress={onPress}

background={TouchableNativeFeedback.Ripple('#28425B', false, 30)}>

<View style={styles.fab}>

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

</View>

</TouchableNativeFeedback>

</View>

);

};

const styles = StyleSheet.create({

fabLocation: {

position: 'absolute',

bottom: 25,

},

right: {

right: 25,

},

left: {

left: 25,

},

fab: {

backgroundColor: '#5856D6',

width: 60,

height: 60,

borderRadius: 100,

justifyContent: 'center',

shadowColor: '#000',

shadowOffset: {

width: 0,

height: 4,

},

shadowOpacity: 0.3,

shadowRadius: 4.65,

elevation: 8,

},

fabText: {

color: 'white',

fontSize: 25,

fontWeight: 'bold',

alignSelf: 'center',

},

});

**ContadorScreen.tsx:**

...

<Fab title="+1" onPress={() => setCounter(counter + 1)} />

<Fab title="-1" position="bl" onPress={() => setCounter(counter - 1)} />

...

**CODIGO ESPECIFICO PARA PLATAFORMA**

En el componente **Fab.tsx** importamos **Platform** de react-native y creamos dos funciones, una **ios** y otra **android**, las cuales van a retornar un JSX.Element:

import React from 'react';

import {

StyleSheet,

Text,

TouchableNativeFeedback,

TouchableOpacity,

View,

Platform,

} from 'react-native';

interface Props {

title: string;

position?: 'br' | 'bl';

onPress: () => void;

}

export const Fab = ({title, onPress, position = 'br'}: Props) => {

const ios = () => {

return (

<TouchableOpacity

activeOpacity={0.75}

onPress={onPress}

style={[

styles.fabLocation,

position === 'br' ? styles.right : styles.left,

]}>

<View style={styles.fab}>

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

</View>

</TouchableOpacity>

);

};

const android = () => {

return (

<View

style={[

styles.fabLocation,

position === 'br' ? styles.right : styles.left,

]}>

<TouchableNativeFeedback

onPress={onPress}

background={TouchableNativeFeedback.Ripple('#28425B', false, 30)}>

<View style={styles.fab}>

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

</View>

</TouchableNativeFeedback>

</View>

);

};

return Platform.OS === 'ios' ? ios() : android();

};

...

De esta forma en Android tendremos ciertos estilos y en ios otros.

**SEC-5: FLEX, POSITION y BOX OBJECT MODEL**

**CONTINUACIÓN DEL PROYECTO – DISEÑOS y FLEXBOX**

Creamos en la carpeta **screens** el archivo BoxObjectModelScreen.tsx y lo importamos en el **App.tsx** donde lo envolvemos en el componente **SafeAreaView** con el objetivo de que en ios aparezca por debajo del notch de la pantalla, asi no aparece donde esta la hora arriba de todo y no se ve nada:

import React from 'react';

import { BoxObjectModelScreen } from './src/screens/BoxObjectModelScreen';

import { SafeAreaView } from 'react-native';

const App = () => {

return (

<SafeAreaView>

<BoxObjectModelScreen />

</SafeAreaView>

);

};

export default App;

**PADDING, MARGIN, BORDER, WIDTH y HEIGHT**

Vamos a configurar nuestro snippet de StyleSheet, para ello en la paleta de comandos vamos a **user snippets>typescriptreact** y pegamos lo siguiente:

"React Native Styles": {

"prefix": "stles",

"body": [

"const styles = StyleSheet.create({",

" $1",

"});"

]

}

Luego en el video empieza a jugar con el width, border, padding, y margin de los contenedores. No me pareció necesario escribirlo acá. Mirar el video cualquier cosa.

**HEIGHT, WIDTH PORCENTUAL y DIMENSIONES DE LA PANTALLA**

Podemos sacar las medidas de la pantalla con el hook **useWindowDimensions** de react native o con **Dimensions** también de react native. La diferencia es que el hook te permite saber la pantalla cuando estamos rotando el celular, el Dimension saca las dimensiones una sola vez, entonces cuando rotamos el celular sigue apareciendo las primeras medidas tomadas y no está bien.

Creamos dentro de la carpeta **screens** el archivo **DimensionesScreen.tsx**:

import React from 'react';

import {

Dimensions,

StyleSheet,

Text,

View,

useWindowDimensions,

} from 'react-native';

// const {width, height} = Dimensions.get('window')

export const DimensionesScreen = () => {

const {width, height} = useWindowDimensions();

return (

<View>

<View style={styles.container}>

<View style={{...styles.cajaMorada, width: width \* 0.5}} />

<View style={styles.cajanaranja} />

</View>

<Text style={styles.title}>

W: {width}, H: {height}

</Text>

</View>

);

};

const styles = StyleSheet.create({

container: {

width: '100%',

height: 200,

backgroundColor: 'red',

},

cajaMorada: {

backgroundColor: '#5856D6',

// width: '50%',

height: '50%',

},

cajanaranja: {

backgroundColor: '#F0A23B',

},

title: {

fontSize: 30,

textAlign: 'center',

},

});

**POSICIÓN RELATIVA**

Todos los elementos tienen position relative por defecto, si le especificas un top en 0 por ejemplo, mantendría su lugar de origen, porque depende del elemento padre, esto no pasaría en un position absolute.

Creamos el screen **PositionScreen.tsx**:

import React from 'react';

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

export const PositionScreen = () => {

return (

<View style={styles.container}>

<View style={styles.cajaMorada} />

<View style={styles.cajaNaranja} />

</View>

);

};

const styles = StyleSheet.create({

container: {

flex: 1,

justifyContent: 'center',

alignItems: 'center',

backgroundColor: '#28C4D9',

},

cajaMorada: {

width: 100,

height: 100,

backgroundColor: '#5856D6',

borderWidth: 10,

borderColor: 'white',

},

cajaNaranja: {

width: 100,

height: 100,

backgroundColor: '#F0A23B',

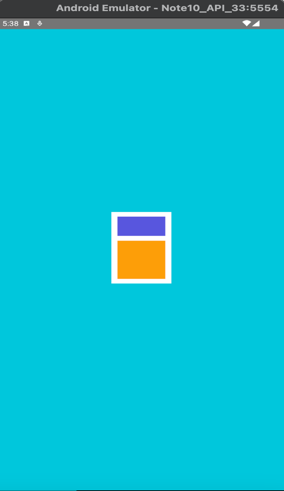
borderWidth: 10,

borderColor: 'white',

top: -50,

},

});



**POSICIÓN ABSOLUTA**

La posición absoluta es dependiendo el elemento padre.

import React from 'react';

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

export const PositionScreen = () => {

return (

<View style={styles.container}>

<View style={styles.cajaVerde} />

<View style={styles.cajaMorada} />

<View style={styles.cajaNaranja} />

</View>

);

};

const styles = StyleSheet.create({

container: {

flex: 1,

// justifyContent: 'center',

// alignItems: 'center',

backgroundColor: '#28C4D9',

},

cajaMorada: {

width: 100,

height: 100,

backgroundColor: '#5856D6',

borderWidth: 10,

borderColor: 'white',

position: 'absolute',

right: 0,

},

cajaNaranja: {

width: 100,

height: 100,

backgroundColor: '#F0A23B',

borderWidth: 10,

borderColor: 'white',

position: 'absolute',

bottom: 0,

right: 0,

},

cajaVerde: {

backgroundColor: 'green',

borderWidth: 10,

borderColor: 'white',

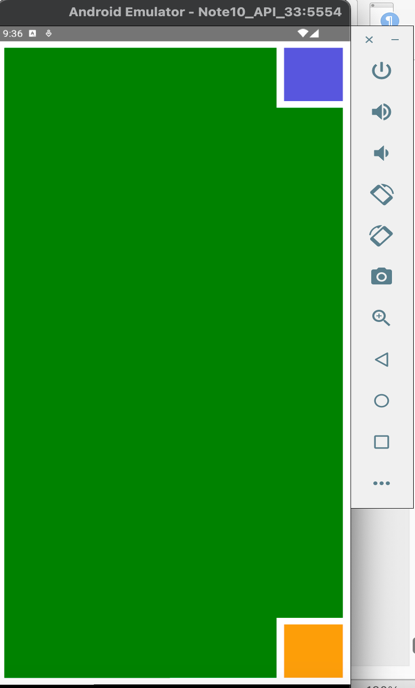
...StyleSheet.absoluteFillObject,

},

});

El **absoluteFillObject** es lo mismo que poner position absolute, top 0, bottom 0, left 0 y right 0. Es decir, estira todo el elemento en sus 4 puntas.

La caja verde se estiró en sus 4 puntas.



**FLEX**

Podemos ver la documentación oficial acá: <https://reactnative.dev/docs/flexbox>

Si le especificamos a los elementos hijos que los 3 van a tener un **flex: 1**, van a compartir el mayor tamaño posible entre los 3, siendo iguales en su tamaño.

Pero si le decimos por ejemplo que dos elementos tienen un flex: 4, y el tercero un flex: 2, los elementos de 4 van a compartir el 40% del elemento padre cada uno, y el tercer elemento un 20%.

import React from 'react';

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

export const FlexScreen = () => {

return (

<View style={styles.container}>

<Text style={styles.caja1}>Caja 1</Text>

<Text style={styles.caja2}>Caja 2</Text>

<Text style={styles.caja3}>Caja 3</Text>

</View>

);

};

const styles = StyleSheet.create({

container: {

flex: 1,

backgroundColor: '#28C4D9',

},

caja1: {

flex: 4,

borderWidth: 2,

borderColor: 'white',

fontSize: 30,

},

caja2: {

flex: 4,

borderWidth: 2,

borderColor: 'white',

fontSize: 30,

},

caja3: {

flex: 2,

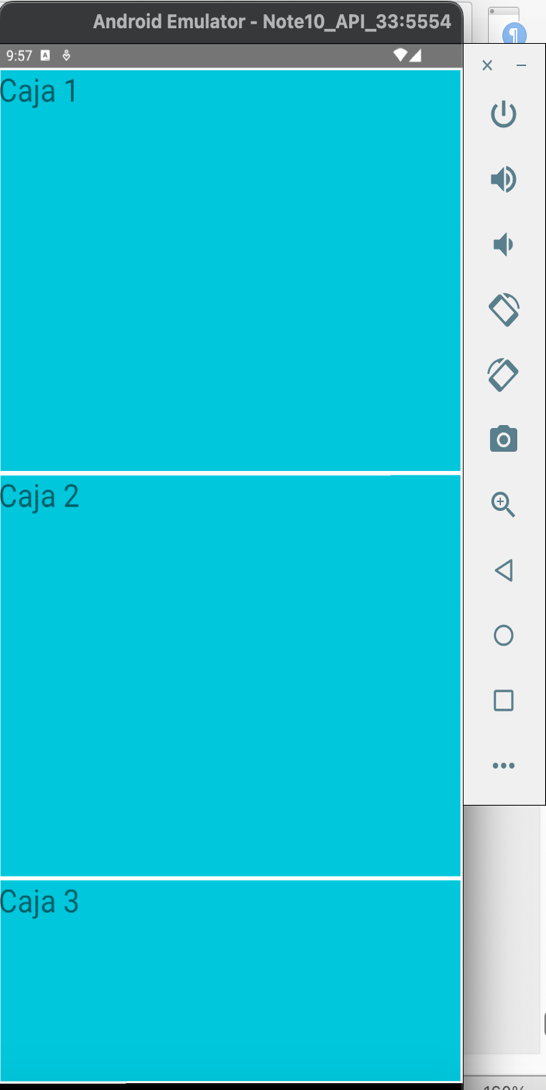
borderWidth: 2,

borderColor: 'white',

fontSize: 30,

},

});



**LAS DEMÁS SECCIONES**

No escribí nada porque es solo CSS que ya conozco, sino ver los videos.

**SEC-6: APLICACIÓN – CALCULADORA DE IOS**

**DISEÑO INICIAL y PANTALLA DE CALCULADORA**

Luego de crear el proyecto con el comando: **npx react-native init MiProyecto --template react-native-template-typescript**

Vamos a crear la carpeta **src**, y dentro de esta, las carpetas **components**, **screens** y **theme**.

En la carpeta **theme** creamos el archivo **appTheme.tsx** el cual va a tener los estilos globales de la aplicación:

import {StyleSheet} from 'react-native';

export const styles = StyleSheet.create({

fondo: {

flex: 1,

backgroundColor: 'black',

},

texto: {

color: 'white',

fontSize: 20,

},

});

Ahora dentro de la carpeta **screens**, creamos el archivo **CalculadoraScreen.tsx** y lo importamos luego en el App.tsx**:**

import React from 'react';

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

import {styles} from '../theme/appTheme';

export const CalculadoraScreen = () => {

return (

<View>

<Text style={styles.texto}>Calculadora Screen</Text>

</View>

);

};

**App.tsx**:

import React from 'react';

import {SafeAreaView, StatusBar} from 'react-native';

import {CalculadoraScreen} from './src/screens/CalculadoraScreen';

import {styles} from './src/theme/appTheme';

const App = () => {

return (

<SafeAreaView style={styles.fondo}>

<StatusBar backgroundColor="black" barStyle="light-content" />

<CalculadoraScreen />

</SafeAreaView>

);

};

export default App;

El componente <**StatusBar />** es donde aparece la hora, la batería, etc.

Podemos ver la documentación del componente aquí:

<https://reactnative.dev/docs/statusbar>

**TEXTOS y MI PRIMER BOTÓN**

Vamos a crear el texto de **resultado** de la calculadora y el primer botón. Para ello, en el **CalculadoraScreen.tsx** vamos a crear unos **View** que serán los contenedores del botón y los estilos en el **appTheme.tsx**

**CalculadoraScreen.tsx:**

import React from 'react';

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

import {styles} from '../theme/appTheme';

export const CalculadoraScreen = () => {

return (

<View style={styles.calculadoraContainer}>

<Text style={styles.resultadoPequeno}>1,500.00</Text>

<Text style={styles.resultado}>1,500.00</Text>

<View>

<View style={styles.boton}>

<Text style={styles.botonTexto}>1</Text>

</View>

</View>

</View>

);

};

**appTheme.tsx:**

import {StyleSheet} from 'react-native';

export const styles = StyleSheet.create({

fondo: {

flex: 1,

backgroundColor: 'black',

},

calculadoraContainer: {

flex: 1,

paddingHorizontal: 20,

justifyContent: 'flex-end',

},

resultado: {

color: 'white',

fontSize: 60,

textAlign: 'right',

marginBottom: 10,

},

resultadoPequeno: {

color: 'rgba(255, 255, 255, 0.5)',

fontSize: 30,

textAlign: 'right',

},

boton: {

height: 80,

width: 80,

backgroundColor: '#9B9B9B',

borderRadius: 100,

justifyContent: 'center',

},

botonTexto: {

textAlign: 'center',

padding: 10,

fontSize: 30,

color: 'black',

fontWeight: '300',

},

});

**BOTÓN DE CALCULADORA y SUS FILAS**

Como vamos a utilizar muchos botones que van a ser iguales pero van a cambiar solo ciertos estilos debemos crear dentro de la carpeta **components**, el componente **BotonCalc.tsx** el cual va a recibir por props, el texto que va a mostrar y el estilo del backgroundColor:

import React from 'react';

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

import {styles} from '../theme/appTheme';

interface Props {

texto: string;

color?: string;

}

export const BotonCalc = ({texto, color = '#2D2D2D'}: Props) => {

return (

<View style={{...styles.boton, backgroundColor: color}}>

<Text style={styles.botonTexto}>{texto}</Text>

</View>

);

};

Ahora en la screen **CalculadoraScreen.tsx** vamos a mostrar este botón y le vamos a mandar sus respectivas props:

import React from 'react';

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

import {styles} from '../theme/appTheme';

import {BotonCalc} from '../components/BotonCalc';

export const CalculadoraScreen = () => {

return (

<View style={styles.calculadoraContainer}>

<Text style={styles.resultadoPequeno}>1,500.00</Text>

<Text style={styles.resultado}>1,500.00</Text>

<View style={styles.fila}>

<BotonCalc texto="C" color="#9B9B9B" />

<BotonCalc texto="+/-" color="#9B9B9B" />

<BotonCalc texto="del" color="#9B9B9B" />

<BotonCalc texto="/" color="#FF9427" />

</View>

</View>

);

};

**appTheme.tsx:**

import {StyleSheet} from 'react-native';

export const styles = StyleSheet.create({

fondo: {

flex: 1,

backgroundColor: 'black',

},

calculadoraContainer: {

flex: 1,

paddingHorizontal: 20,

justifyContent: 'flex-end',

},

resultado: {

color: 'white',

fontSize: 60,

textAlign: 'right',

marginBottom: 10,

},

resultadoPequeno: {

color: 'rgba(255, 255, 255, 0.5)',

fontSize: 30,

textAlign: 'right',

},

fila: {

flexDirection: 'row',

justifyContent: 'center',

marginBottom: 18,

paddingHorizontal: 10,

},

boton: {

height: 80,

width: 80,

backgroundColor: '#2D2D2D',

borderRadius: 100,

justifyContent: 'center',

marginHorizontal: 10,

},

botonTexto: {

textAlign: 'center',

padding: 10,

fontSize: 30,

color: 'white',

fontWeight: '300',

},

});

**TERMINAR TODOS LOS BOTONES Y EL ESTILO DE LA CALCULADORA**

Ahora le pasamos las filas restantes de botones con sus respectivas props en **CalculadoraScreen.tsx:**

import React from 'react';

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

import {styles} from '../theme/appTheme';

import {BotonCalc} from '../components/BotonCalc';

export const CalculadoraScreen = () => {

return (

<View style={styles.calculadoraContainer}>

<Text style={styles.resultadoPequeno}>1,500.00</Text>

<Text style={styles.resultado}>1,500.00</Text>

<View style={styles.fila}>

<BotonCalc texto="C" color="#9B9B9B" />

<BotonCalc texto="+/-" color="#9B9B9B" />

<BotonCalc texto="del" color="#9B9B9B" />

<BotonCalc texto="/" color="#FF9427" />

</View>

<View style={styles.fila}>

<BotonCalc texto="7" />

<BotonCalc texto="8" />

<BotonCalc texto="9" />

<BotonCalc texto="X" color="#FF9427" />

</View>

<View style={styles.fila}>

<BotonCalc texto="4" />

<BotonCalc texto="5" />

<BotonCalc texto="6" />

<BotonCalc texto="-" color="#FF9427" />

</View>

<View style={styles.fila}>

<BotonCalc texto="1" />

<BotonCalc texto="2" />

<BotonCalc texto="3" />

<BotonCalc texto="+" color="#FF9427" />

</View>

<View style={styles.fila}>

<BotonCalc texto="0" ancho />

<BotonCalc texto="." />

<BotonCalc texto="=" color="#FF9427" />

</View>

</View>

);

};

Ahora en el componente **BotonCalc.tsx** debemos pasarle una prop llamada **ancho** que va a ser un valor booleano, si esta propiedad llega, el ancho del botón va a ser más grande:

import React from 'react';

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

import {styles} from '../theme/appTheme';

interface Props {

texto: string;

color?: string;

ancho?: boolean;

}

export const BotonCalc = ({texto, color = '#2D2D2D', ancho = false}: Props) => {

return (

<TouchableOpacity>

<View

style={{

...styles.boton,

backgroundColor: color,

width: ancho ? 180 : 80,

}}>

<Text

style={{

...styles.botonTexto,

color: color === '#9B9B9B' ? 'black' : 'white',

}}>

{texto}

</Text>

</View>

</TouchableOpacity>

);

};

El **TouchableOpacity** es para darle el efecto de interacción con el botón.

**CONSTRUIR EL NÚMERO BASE**

Para hacer esto vamos a usar un **useState** para el número base y el número anterior. Además debemos pasar ahora como props un **action** que será el encargado de ir poniendo los números que digamos, y también aplicamos la action de borrar todo.

**CalculadoraScreen.tsx:**

import React, {useState} from 'react';

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

import {styles} from '../theme/appTheme';

import {BotonCalc} from '../components/BotonCalc';

export const CalculadoraScreen = () => {

const [numberCalc, setNumberCalc] = useState('0');

const [numberBeforeCalc, setNumberBeforeCalc] = useState('0');

const cleanCalc = () => {

setNumberCalc('0');

setNumberBeforeCalc('0');

};

const madeANumber = (textNumber: string) => {

setNumberCalc(numberCalc + textNumber);

};

return (

<View style={styles.calculadoraContainer}>

<Text style={styles.resultadoPequeno}>{numberBeforeCalc}</Text>

<Text style={styles.resultado} numberOfLines={1} adjustsFontSizeToFit>

{numberCalc}

</Text>

<View style={styles.fila}>

<BotonCalc texto="C" color="#9B9B9B" action={cleanCalc} />

<BotonCalc texto="+/-" color="#9B9B9B" action={cleanCalc} />

<BotonCalc texto="del" color="#9B9B9B" action={cleanCalc} />

<BotonCalc texto="/" color="#FF9427" action={cleanCalc} />

</View>

<View style={styles.fila}>

<BotonCalc texto="7" action={madeANumber} />

<BotonCalc texto="8" action={madeANumber} />

<BotonCalc texto="9" action={madeANumber} />

<BotonCalc texto="X" color="#FF9427" action={cleanCalc} />

</View>

<View style={styles.fila}>

<BotonCalc texto="4" action={madeANumber} />

<BotonCalc texto="5" action={madeANumber} />

<BotonCalc texto="6" action={madeANumber} />

<BotonCalc texto="-" color="#FF9427" action={cleanCalc} />

</View>

<View style={styles.fila}>

<BotonCalc texto="1" action={madeANumber} />

<BotonCalc texto="2" action={madeANumber} />

<BotonCalc texto="3" action={madeANumber} />

<BotonCalc texto="+" color="#FF9427" action={cleanCalc} />

</View>

<View style={styles.fila}>

<BotonCalc texto="0" action={madeANumber} ancho />

<BotonCalc texto="." action={madeANumber} />

<BotonCalc texto="=" color="#FF9427" action={cleanCalc} />

</View>

</View>

);

};

En el componente **<Text>** vemos que pasamos las propiedades **numberOfLines** y **adjustsFontSizeToFit**, los cuales son los encargados de que se mantenga el número en una sola línea cuando está creciendo y que el tamaño se ajuste dependiendo de que tan grande sea el número.

**BotonCalc.tsx:**

import React from 'react';

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

import {styles} from '../theme/appTheme';

interface Props {

texto: string;

color?: string;

ancho?: boolean;

action: (textNumber: string) => void;

}

export const BotonCalc = ({

texto,

color = '#2D2D2D',

ancho = false,

action,

}: Props) => {

return (

<TouchableOpacity onPress={() => action(texto)}>

<View

style={{

...styles.boton,

backgroundColor: color,

width: ancho ? 180 : 80,

}}>

<Text

style={{

...styles.botonTexto,

color: color === '#9B9B9B' ? 'black' : 'white',

}}>

{texto}

</Text>

</View>

</TouchableOpacity>

);

};

En el **TouchableOpacity** estamos pasando en el **onPress** la **action**, y poniendo como parámetro el **texto**, que sería el número que apretamos.

**CONSIDERACIONES PARA ARMAR EL NÚMERO**

Vamos a crear en **CalculadoraScreen.tsx** una función para pasar de un valor positivo a negativo y viceversa, y además en la función que va creando el número, debemos aplicar ciertas validaciones:

...

const madeANumber = (textNumber: string) => {

// No aceptar doble punto

if (numberCalc.includes('.') && textNumber === '.') return;

if (numberCalc.startsWith('0') || numberCalc.startsWith('-0')) {

// Punto decimal

if (textNumber === '.') {

setNumberCalc(numberCalc + textNumber);

// Evaluar si es otro cero, y hay un punto

} else if (textNumber === '0' && numberCalc.includes('.')) {

setNumberCalc(numberCalc + textNumber);

// Evaluar si es diferente de 0 y no tiene un punto

} else if (textNumber !== '0' && !numberCalc.includes('.')) {

setNumberCalc(textNumber);

// Evitar 000.0

} else if (textNumber === '0' && !numberCalc.includes('.')) {

setNumberCalc(numberCalc);

} else {

setNumberCalc(numberCalc + textNumber);

}

} else {

setNumberCalc(numberCalc + textNumber);

}

};

const positiveNegative = () => {

if (numberCalc.includes('-')) {

setNumberCalc(numberCalc.replace('-', ''));

} else {

setNumberCalc('-' + numberCalc);

}

};

...

<BotonCalc texto="+/-" color="#9B9B9B" action={positiveNegative} />

...

**TAREA – BOTÓN DE BORRAR ÚLTIMA ENTRADA**

En **CalculadoraScreen.tsx** creamos la función para borrar el último número de nuestro número principal:

...

const btnDeleteLastNumber = () => {

let negative = '';

let numberTemp = numberCalc;

if (numberCalc.includes('-')) {

negative = '-';

numberTemp = numberCalc.substring(1);

}

if (numberTemp.length > 1) {

setNumberCalc(negative + numberTemp.slice(0, -1));

} else {

setNumberCalc('0');

}

};

...

<BotonCalc texto="del" color="#9B9B9B" action={btnDeleteLastNumber} />

...

**BOTONES DE OPERACIONES ARITMÉTICAS**

Para hacer esto vamos a utilizar un **useRef** con el fin de que la aplicación no se vuelva a renderizar. Para ello vamos a usar un **enum** de typescript donde especificamos de que tipos puede ser el useRef.

Luego debemos crear una función que al apretar algún botón de operaciones aritméticas almacene en el state **numberBeforeCalc** el **numberCalc**, y además, las funciones de las operaciones aritméticas donde ejecutamos esa función y le especificamos al ref que operación estamos por ejecutar.

**CalculadoraScreen.tsx:**

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

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

import {styles} from '../theme/appTheme';

import {BotonCalc} from '../components/BotonCalc';

enum Operadores {

sumar,

restar,

multiplicar,

dividir,

}

export const CalculadoraScreen = () => {

const [numberCalc, setNumberCalc] = useState('0');

const [numberBeforeCalc, setNumberBeforeCalc] = useState('0');

const lastOperation = useRef<Operadores>();

...

const changeNumberForBeforeNumber = () => {

if (numberCalc.endsWith('.')) {

setNumberBeforeCalc(numberCalc.slice(0, -1));

} else {

setNumberBeforeCalc(numberCalc);

}

setNumberCalc('0');

};

const btnDivide = () => {

changeNumberForBeforeNumber();

lastOperation.current = Operadores.dividir;

};

const btnMultiply = () => {

changeNumberForBeforeNumber();

lastOperation.current = Operadores.multiplicar;

};

const btnSubtract = () => {

changeNumberForBeforeNumber();

lastOperation.current = Operadores.restar;

};

const btnAdd = () => {

changeNumberForBeforeNumber();

lastOperation.current = Operadores.sumar;

};

return (

...

<BotonCalc texto="/" color="#FF9427" action={btnDivide} />

...

<BotonCalc texto="X" color="#FF9427" action={btnMultiply} />

...

<BotonCalc texto="-" color="#FF9427" action={btnSubtract} />

...

<BotonCalc texto="+" color="#FF9427" action={btnAdd} />

...

**REALIZAR EL CÁLCULO**

Ahora creamos la función **calculate** para realizar las distintas operaciones matemáticas.

**CalculadoraScreen.tsx:**

...

const calculate = () => {

const num1 = Number(numberCalc);

const num2 = Number(numberBeforeCalc);

switch (lastOperation.current) {

case Operadores.sumar:

setNumberCalc(`${num1 + num2}`);

break;

case Operadores.restar:

setNumberCalc(`${num2 - num1}`);

break;

case Operadores.multiplicar:

setNumberCalc(`${num1 \* num2}`);

break;

case Operadores.dividir:

setNumberCalc(`${num2 / num1}`);

break;

}

setNumberBeforeCalc('0');

};

...

<BotonCalc texto="=" color="#FF9427" action={calculate} />

...

**CUSTOM HOOK: useCalculadora**

Ahora toca refactorizar el **CalculadoraScreen.tsx** para que sea más fácil de leer, y el código sea más escalable. Para ello vamos a crear dentro de la carpeta **src**, una carpeta llamada **hooks** con el archivo **useCalculadora.tsx** la cual contendrá toda la lógica que tiene CalculadoraScreen.tsx.

**useCalculadora.tsx:**

import {useRef, useState} from 'react';

enum Operadores {

sumar,

restar,

multiplicar,

dividir,

}

export const useCalculadora = () => {

const [numberCalc, setNumberCalc] = useState('0');

const [numberBeforeCalc, setNumberBeforeCalc] = useState('0');

const lastOperation = useRef<Operadores>();

const cleanCalc = () => {

setNumberCalc('0');

setNumberBeforeCalc('0');

};

const madeANumber = (textNumber: string) => {

// No aceptar doble punto

if (numberCalc.includes('.') && textNumber === '.') return;

if (numberCalc.startsWith('0') || numberCalc.startsWith('-0')) {

// Punto decimal

if (textNumber === '.') {

setNumberCalc(numberCalc + textNumber);

// Evaluar si es otro cero, y hay un punto

} else if (textNumber === '0' && numberCalc.includes('.')) {

setNumberCalc(numberCalc + textNumber);

// Evaluar si es diferente de 0 y no tiene un punto

} else if (textNumber !== '0' && !numberCalc.includes('.')) {

setNumberCalc(textNumber);

// Evitar 000.0

} else if (textNumber === '0' && !numberCalc.includes('.')) {

setNumberCalc(numberCalc);

} else {

setNumberCalc(numberCalc + textNumber);

}

} else {

setNumberCalc(numberCalc + textNumber);

}

};

const positiveNegative = () => {

if (numberCalc.includes('-')) {

setNumberCalc(numberCalc.replace('-', ''));

} else {

setNumberCalc('-' + numberCalc);

}

};

const btnDeleteLastNumber = () => {

let negative = '';

let numberTemp = numberCalc;

if (numberCalc.includes('-')) {

negative = '-';

numberTemp = numberCalc.substring(1);

}

if (numberTemp.length > 1) {

setNumberCalc(negative + numberTemp.slice(0, -1));

} else {

setNumberCalc('0');

}

};

const changeNumberForBeforeNumber = () => {

if (numberCalc.endsWith('.')) {

setNumberBeforeCalc(numberCalc.slice(0, -1));

} else {

setNumberBeforeCalc(numberCalc);

}

setNumberCalc('0');

};

const btnDivide = () => {

changeNumberForBeforeNumber();

lastOperation.current = Operadores.dividir;

};

const btnMultiply = () => {

changeNumberForBeforeNumber();

lastOperation.current = Operadores.multiplicar;

};

const btnSubtract = () => {

changeNumberForBeforeNumber();

lastOperation.current = Operadores.restar;

};

const btnAdd = () => {

changeNumberForBeforeNumber();

lastOperation.current = Operadores.sumar;

};

const calculate = () => {

const num1 = Number(numberCalc);

const num2 = Number(numberBeforeCalc);

switch (lastOperation.current) {

case Operadores.sumar:

setNumberCalc(`${num1 + num2}`);

break;

case Operadores.restar:

setNumberCalc(`${num2 - num1}`);

break;

case Operadores.multiplicar:

setNumberCalc(`${num1 \* num2}`);

break;

case Operadores.dividir:

setNumberCalc(`${num2 / num1}`);

break;

}

setNumberBeforeCalc('0');

};

return {

numberCalc,

numberBeforeCalc,

cleanCalc,

madeANumber,

positiveNegative,

btnDeleteLastNumber,

btnDivide,

btnMultiply,

btnSubtract,

btnAdd,

calculate,

};

};

**CalculadoraScreen.tsx:**

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

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

import {styles} from '../theme/appTheme';

import {BotonCalc} from '../components/BotonCalc';

import {useCalculadora} from '../hooks/useCalculadora';

export const CalculadoraScreen = () => {

const {

numberBeforeCalc,

numberCalc,

cleanCalc,

positiveNegative,

btnDeleteLastNumber,

btnDivide,

madeANumber,

btnMultiply,

btnAdd,

btnSubtract,

calculate,

} = useCalculadora();

return (

<View style={styles.calculadoraContainer}>

{numberBeforeCalc !== '0' && (

<Text style={styles.resultadoPequeno}>{numberBeforeCalc}</Text>

)}

<Text style={styles.resultado} numberOfLines={1} adjustsFontSizeToFit>

{numberCalc}

</Text>

<View style={styles.fila}>

<BotonCalc texto="C" color="#9B9B9B" action={cleanCalc} />

<BotonCalc texto="+/-" color="#9B9B9B" action={positiveNegative} />

<BotonCalc texto="del" color="#9B9B9B" action={btnDeleteLastNumber} />

<BotonCalc texto="/" color="#FF9427" action={btnDivide} />

</View>

<View style={styles.fila}>

<BotonCalc texto="7" action={madeANumber} />

<BotonCalc texto="8" action={madeANumber} />

<BotonCalc texto="9" action={madeANumber} />

<BotonCalc texto="X" color="#FF9427" action={btnMultiply} />

</View>

<View style={styles.fila}>

<BotonCalc texto="4" action={madeANumber} />

<BotonCalc texto="5" action={madeANumber} />

<BotonCalc texto="6" action={madeANumber} />

<BotonCalc texto="-" color="#FF9427" action={btnSubtract} />

</View>

<View style={styles.fila}>

<BotonCalc texto="1" action={madeANumber} />

<BotonCalc texto="2" action={madeANumber} />

<BotonCalc texto="3" action={madeANumber} />

<BotonCalc texto="+" color="#FF9427" action={btnAdd} />

</View>

<View style={styles.fila}>

<BotonCalc texto="0" action={madeANumber} ancho />

<BotonCalc texto="." action={madeANumber} />

<BotonCalc texto="=" color="#FF9427" action={calculate} />

</View>

</View>

);

};

**SEC-7: NAVEGACIÓN APP – TODOS LOS TIPOS DE NAVEGACIÓN**

**EXPLICACIÓN SOBRE EL SISTEMA DE NEVEGACIÓN**

Hay 4:

* Stack Navigation.
* Drawer Navigation.
* BottomTab Navigation.
* MaterialTop Navigation.

**REACT NAVIGATION – PRE REQUISITIOS**

Vamos a la página oficial de **React Navigation** en:

<https://reactnavigation.org/docs/getting-started/>

Y luego en **getting started** seguimos la documentación.

Nos pedirá instalarlo con el comando:

**npm install @react-navigation/native**

Luego instalamos ciertas dependencias:

**npm install react-native-screens react-native-safe-area-context**

Y si estas en Mac debemos instalar el siguiente comando:

**npx pod-install ios**

A continuación, en la siguiente ruta del proyecto **Android/app/src/main/java/<your package name>/MainActivity.java**, y ahí dentro copiamos la siguiente configuración:

import android.os.Bundle;

...

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(null);

}

Y ahora en el **App.tsx** agregamos las siguientes líneas de código para envolver todo dentro de **NavigationContainer**:

import React from 'react';

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

import {Text} from 'react-native';

const App = () => {

return (

<NavigationContainer>

<Text>App Navigation</Text>

</NavigationContainer>

);

};

export default App;

**REACT NAVIGATION – STACK**

Vamos a la documentación en <https://reactnavigation.org/docs/stack-navigator/> y seguimos la instalación.

Corremos los comando:

**npm install @react-navigation/stack**

**npm install react-native-gesture-handler**

Y arriba de todo como primera línea de código en el **App.tsx** escribimos la siguiente línea:

import 'react-native-gesture-handler';

...

Ahora corremos los comando:

**npm install @react-native-masked-view/masked-view**

**npx pod-install ios**

A continuación, en el proyecto creamos la carpeta **src** con la subcarpetas **screens**, **theme**, **components** y **navigation**.

Dentro de **screens** creamos los archivos **Pagina1, Pagina2 y Pagina3Screen.tsx**

Dentro de **navigation** creamos el archivo **StackNavigator.tsx** y copiamos el siguiente código:

import React from 'react';

import {createStackNavigator} from '@react-navigation/stack';

import {Pagina1Screen} from '../screens/Pagina1Screen';

import {Pagina2Screen} from '../screens/Pagina2Screen';

import {Pagina3Screen} from '../screens/Pagina3Screen';

const Stack = createStackNavigator();

export const StackNavigator = () => {

return (

<Stack.Navigator>

<Stack.Screen name="Pagina1Screen" component={Pagina1Screen} />

<Stack.Screen name="Pagina2Screen" component={Pagina2Screen} />

<Stack.Screen name="Pagina3Screen" component={Pagina3Screen} />

</Stack.Navigator>

);

};

Y ahora dentro del **App.tsx** importamos este StackNavigator y lo retornamos:

import 'react-native-gesture-handler';

import React from 'react';

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

import {StackNavigator} from './src/navigation/StackNavigator';

const App = () => {

return (

<NavigationContainer>

<StackNavigator />

</NavigationContainer>

);

};

export default App;

**NAVEGAR A OTRAS PANTALLAS**

Ahora para navegar entre distintas pantallas vamos a poder utilizar dos maneras distinas, mediante **Props**, o mediante un hook llamado **useNavigation()**.

**Pagina1Screen.tsx:**

import {StackScreenProps} from '@react-navigation/stack';

import React from 'react';

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

import {styles} from '../theme/appTheme';

interface Props extends StackScreenProps<any, any> {}

export const Pagina1Screen = ({navigation}: Props) => {

return (

<View style={styles.globalMargin}>

<Text>Pagina1Screen</Text>

<Button

title="ir a la página 2"

onPress={() => navigation.navigate('Pagina2Screen')}

/>

</View>

);

};

Como vemos utilizamos **StackScreenProps** para pasarle el tipado, y luego usamos la desestructuración para agarrar el **navigation** y en el Button pasamos la función para ir a la página con navigation.navigate(‘nombreDeLaPagina’).

**Pagina2Screen.tsx:**

import React from 'react';

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

import {styles} from '../theme/appTheme';

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

export const Pagina2Screen = () => {

const navigation = useNavigation<any>();

return (

<View style={styles.globalMargin}>

<Text>Pagina2Screen</Text>

<Button

title="ir página 3"

onPress={() => navigation.navigate('Pagina3Screen')}

/>

</View>

);

};

En este caso usamos el hook **useNavigation()** el cual se ejecuta de la misma forma mediante el **navigate(‘nombreDeLaPagina’)**.

**Pagina3Screen.tsx:**

import {StackScreenProps} from '@react-navigation/stack';

import React from 'react';

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

import {styles} from '../theme/appTheme';

interface Props extends StackScreenProps<any, any> {}

export const Pagina3Screen = ({navigation}: Props) => {

return (

<View style={styles.globalMargin}>

<Text>Pagina3Screen</Text>

<Button title="Regresar" onPress={() => navigation.pop()} />

<Button title="Ir al Home" onPress={() => navigation.popToTop()} />

</View>

);

};

Y en la carpeta **theme** creamos el **appTheme.tsx** con el fin de tener estilos globales:

import {StyleSheet} from 'react-native';

export const styles = StyleSheet.create({

globalMargin: {

marginHorizontal: 20,

},

});

**ESTILIZANDO EL STACK NAVIGATOR**

En el **navigator/StackNavigator.tsx** si le pasamos la propiedad **initialRouteName** al componente **<Stack.Navigator>** le podemos especificar que arranque por una página en especial:

...

<Stack.Navigator

initialRouteName='Pagina2Screen'

>

...

También tiene la propiedad **screenOptions** en el cual podemos modificar varias cosas de los estilos como el color de fondo, etc.

Y además en cada **<Stack.Screen>** podemos pasarle una propiedad **option** que recibe un objeto y le podemos especificar el **title**:

import React from 'react';

import {createStackNavigator} from '@react-navigation/stack';

import {Pagina1Screen} from '../screens/Pagina1Screen';

import {Pagina2Screen} from '../screens/Pagina2Screen';

import {Pagina3Screen} from '../screens/Pagina3Screen';

const Stack = createStackNavigator();

export const StackNavigator = () => {

return (

<Stack.Navigator

screenOptions={{

headerStyle: {

elevation: 0, //android

shadowColor: 'transparent', //ios

},

cardStyle: {

backgroundColor: 'white',

},

}}>

<Stack.Screen

name="Pagina1Screen"

options={{title: 'Página 1'}}

component={Pagina1Screen}

/>

<Stack.Screen

name="Pagina2Screen"

options={{title: 'Página 2'}}

component={Pagina2Screen}

/>

<Stack.Screen

name="Pagina3Screen"

options={{title: 'Página 3'}}

component={Pagina3Screen}

/>

</Stack.Navigator>

);

};

Para termina, en **Pagina2Screen.tsx** vamos a cambiarle para ios el nombre que aparece al lado de la flechita para volver a la página anterior:

...

export const Pagina2Screen = () => {

const navigation = useNavigation<any>();

useEffect(() => {

navigation.setOptions({

headerBackTitle: 'Back',

});

}, []);

...

**ENVIAR ARGUMENTOS ENTRE PANTALLAS**

Nos vamos a crear un **PersonaScreen.tsx** el cual vamos a tener que poner en el router de **StackNavigator.tsx**:

...

<Stack.Screen

name="PersonaScreen"

options={{title: 'Página Persona'}}

component={PersonaScreen}

/>

...

Luego en **Pagina1Screen.tsx** creamos un touchableOpacity para navegar a esta página en la cual en la propiedad **onPress** le podemos pasar un objeto que contengan argumentos por props:

...

<Text>Navegar con argumentos</Text>

<TouchableOpacity

onPress={() => navigation.navigate('PersonaScreen', {

id: 1,

nombre: 'Joaquin'

})}

>

<Text>Joaquín</Text>

</TouchableOpacity>

...

**PersonaScreen.tsx:**

import React, {useEffect} from 'react';

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

import {styles} from '../theme/appTheme';

import {StackScreenProps} from '@react-navigation/stack';

interface Props extends StackScreenProps<any, any> {}

export const PersonaScreen = ({navigation, route}: Props) => {

const params = route.params;

useEffect(() => {

navigation.setOptions({

title: params!.nombre,

});

}, []);

return (

<View style={styles.globalMargin}>

<Text style={styles.title}>{JSON.stringify(params, null, 3)}</Text>

</View>

);

};

En **route.params** vamos a recibir los argumentos que le pasamos desde **Pagina1Screen.tsx**, pero de esta forma no estamos especificando bien el tipado. En el próximo video lo explica cómo hay que hacer.

**DIFERENTES FORMAS DE COLOCAR EL TIPO DE DATO DE LOS ARGUMENTOS**

Primera forma:

Hacer una interface que defina los argumentos que estamos pasándole:

...

interface RouterParams {

id: number;

name: string;

}

export const PersonaScreen = ({navigation, route}: Props) => {

const params = route.params as RouterParams;

...

Segunda forma (recomendada):

En el **StackNavigator.tsx** es donde creamos un **type** para especificar las pantallas que vamos a tener y cuales reciben parámetros. Si no reciben parámetros se le pasa **undefined**.

...

export type RootStackParams = {

Pagina1Screen: undefined;

Pagina2Screen: undefined;

Pagina3Screen: undefined;

PersonaScreen: {id: number; name: string};

};

const Stack = createStackNavigator<RootStackParams>();

...

Luego en **PersonaScreen.tsx** en la interface de **Props** es donde le pasamos este **RootStackParams** y el nombre de la página:

...

interface Props extends StackScreenProps<RootStackParams, 'PersonaScreen'> {}

export const PersonaScreen = ({navigation, route}: Props) => {

const params = route.params;

useEffect(() => {

navigation.setOptions({

title: params.name,

});

}, []);

...

**CONFIGURAR DRAWER BÁSICO**

Para ello sigamos los pasos de la documentación oficial:

<https://reactnavigation.org/docs/drawer-navigator/>

Debemos correr los comandos:

**npm install @react-navigation/drawer**

**npm install react-native-gesture-handler react-native-reanimated**

**npx pod-install**

No olvidarse de tener la siguiente línea en el comienzo del **App.tsx**:

import 'react-native-gesture-handler';

Luego creamos la carpeta **navigation** donde van a ir **MenuLateralBasico.tsx** y **StackNavigator.tsx**.

**MenuLateralBasico.tsx:**

import React from 'react';

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

import {StackNavigator} from './StackNavigator';

import {SettingsScreen} from '../screens/SettingsScreen';

const Drawer = createDrawerNavigator();

export const MenuLateralBasico = () => {

return (

<Drawer.Navigator>

<Drawer.Screen name="StackNavigator" component={StackNavigator} />

<Drawer.Screen name="SettingsScreen" component={SettingsScreen} />

</Drawer.Navigator>

);

};

Utilizamos **<Drawer.Navigator>** y **<Drawer.Screen>** para elegir los componentes que vamos a mostrar en el drawer.

**TOGGLE DRAWER – MOSTRAR/OCULTAR**

Ahora ya tenemos nuestro menú hamburguesa en la esquina superior izquierda.

Para modificar la posición desde donde se abre el drawer, debemos utilizar la prop **screenOptions**, la cual recibe un objeto con la propiedad **drawerPosition**, y el valor puede ser **left** o **right**.

También podemos hacer el drawer cuando el celular esté en posición horizontal permanezca abierto, colocando la propiedad **drawerType**, con el valor en **permanent**. Para eso, utilizamos un ternario en donde si la pantalla es >= 768 sea permanent y sino **front**. El **width** para el ternario lo sacamos del hook **useWindowDimensions**.

Para cambiar el nombre de como aparecen los componentes en el drawer, dentro de **<Drawer.Screen>** colocamos la propiedad **options={{title: ‘nombreQueQueramos’}}**.

import React from 'react';

import {useWindowDimensions} from 'react-native';

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

import {StackNavigator} from './StackNavigator';

import {SettingsScreen} from '../screens/SettingsScreen';

const Drawer = createDrawerNavigator();

export const MenuLateralBasico = () => {

const {width} = useWindowDimensions();

return (

<Drawer.Navigator

screenOptions={{

drawerPosition: 'right',

drawerType: width >= 768 ? 'permanent' : 'front',

}}>

<Drawer.Screen

name="StackNavigator"

options={{title: 'Home'}}

component={StackNavigator}

/>

<Drawer.Screen

name="SettingsScreen"

options={{title: 'Settings'}}

component={SettingsScreen}

/>

</Drawer.Navigator>

);

};

Además, el profe crea de manera manual un menú en Pagina1Screen, pero esto en la nueva versión ya no hace falta porque el menú viene automáticamente.

Utiliza de **props** el **DrawerScreenProps** la vual tiene la propiedad de **toggleDrawer** para abrir y cerrar el drawer.

Utiliza un **useEffect** donde devuelve un Button que será el menú hamburguesa.

import React, {useEffect} from 'react';

import {Button, Text, View, TouchableOpacity} from 'react-native';

import {DrawerScreenProps} from '@react-navigation/drawer';

// import {StackScreenProps} from '@react-navigation/stack';

import {styles} from '../theme/appTheme';

// interface Props extends StackScreenProps<any, any> {}

interface Props extends DrawerScreenProps<any, any> {}

export const Pagina1Screen = ({navigation}: Props) => {

useEffect(() => {

navigation.setOptions({

headerLeft: () => (

<Button title="menu" onPress={() => navigation.toggleDrawer()} />

),

});

}, []);

return (

...

**DRAWER PERSONALIZADO**

En la carpeta **navigation** creamos el componente **MenuLateral.tsx**, en el cual dentro del **<Drawer.Navigator>** vamos a usar la propiedad **drawerContent**, la cual recibe una función que devuelve un functional component. Por lo que, creamos este componente por debajo del principal y retorna un **<DrawerContentScrollView>**, el cual tendrá todo lo que queramos mostrar en el drawer:

import React from 'react';

import {Image, Text, View, useWindowDimensions} from 'react-native';

import {

DrawerContentComponentProps,

DrawerContentScrollView,

createDrawerNavigator,

} from '@react-navigation/drawer';

import {StackNavigator} from './StackNavigator';

import {SettingsScreen} from '../screens/SettingsScreen';

import {styles} from '../theme/appTheme';

const Drawer = createDrawerNavigator();

export const MenuLateral = () => {

const {width} = useWindowDimensions();

return (

<Drawer.Navigator

screenOptions={{

drawerType: width >= 768 ? 'permanent' : 'front',

}}

drawerContent={props => <MenuInterno {...props} />}>

<Drawer.Screen name="StackNavigator" component={StackNavigator} />

<Drawer.Screen name="SettingsScreen" component={SettingsScreen} />

</Drawer.Navigator>

);

};

const MenuInterno = (props: DrawerContentComponentProps) => {

return (

<DrawerContentScrollView>

<View style={styles.avatarContainer}>

<Image

source={{

uri: 'https://medgoldresources.com/wp-content/uploads/2018/02/avatar-placeholder.gif',

}}

style={styles.avatar}

/>

</View>

</DrawerContentScrollView>

);

};

**appTheme.tsx:**

...

avatar: {

width: 100,

height: 100,

borderRadius: 100,

},

avatarContainer: {

alignItems: 'center',

marginTop: 20,

},

...

No olvidarse de colocar el **MenuLateral.tsx** dentro del **App.tsx:**

import 'react-native-gesture-handler';

import React from 'react';

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

import {MenuLateral} from './src/navigation/MenuLateral';

// import { MenuLateralBasico } from './src/navigation/MenuLateralBasico';

// import {StackNavigator} from './src/navigation/StackNavigator';

const App = () => {

return (

<NavigationContainer>

{/\* <StackNavigator /> \*/}

{/\* <MenuLateralBasico /> \*/}

<MenuLateral />

</NavigationContainer>

);

};

export default App;

**NAVEGAR DESDE EL MENU LATERAL PERSONALIZADO**

Ahora en el **MenuLateral.tsx** dentro del componente **MenuInterno** vamos a escribir las opciones de navegación que tendrá el drawer:

...

const MenuInterno = ({navigation}: DrawerContentComponentProps) => {

return (

<DrawerContentScrollView>

{/\* Avatar \*/}

<View style={styles.avatarContainer}>

<Image

source={{

uri: 'https://medgoldresources.com/wp-content/uploads/2018/02/avatar-placeholder.gif',

}}

style={styles.avatar}

/>

</View>

{/\* Opciones de menú \*/}

<View style={styles.menuContainer}>

<TouchableOpacity

onPress={() => navigation.navigate('StackNavigator')}

style={styles.menuButton}>

<Text style={styles.menuText}>Navegación</Text>

</TouchableOpacity>

<TouchableOpacity

onPress={() => navigation.navigate('SettingsScreen')}

style={styles.menuButton}>

<Text style={styles.menuText}>Ajustes</Text>

</TouchableOpacity>

</View>

</DrawerContentScrollView>

);

};

**appTheme.tsx:**

...

menuContainer: {

marginVertical: 30,

marginHorizontal: 50,

},

menuText: {

fontSize: 20

},

menuButton: {

marginVertical: 5,

},

...

**useSafeAreaInsets**

En el video del curso como está desactualizado hace falta implementar un margin en **SettingsScreen.tsx**, pero en la nueva versión no hace falta porque el drawer genera un stack automático.

Para implementar este margin, utilizamos un hook llamado **useSafeAreaInsets()**, el cual te da las medidas necesarias para que quede bien.

**SettingsScreen.tsx**:

import React from 'react';

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

import {useSafeAreaInsets} from 'react-native-safe-area-context';

import {styles} from '../theme/appTheme';

export const SettingsScreen = () => {

const insets = useSafeAreaInsets();

return (

<View style={{...styles.globalMargin, marginTop: insets.top}}>

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

</View>

);

};

**SEC-8: TABS, MATERIAL TABS y MATERIAL TOP SCROLLABLE TABS**

**CREAR EL BOTTOM TAB NAVIGATOR**

Primero debemos instalarlo con el comando:

**npm install @react-navigation/bottom-tabs**

Ver la documentación aquí: <https://reactnavigation.org/docs/bottom-tab-navigator/>

Luego en la carpeta **Navigation** creamos el componente **Tabs.tsx** el cual importa **createBottomTabNavigator** para su configuración. Además importamos 2 screen más que sirvan como página de ejemplo.

**Tabs.tsx:**

import React from 'react';

import {createBottomTabNavigator} from '@react-navigation/bottom-tabs';

import Tab1Screen from '../screens/Tab1Screen';

import Tab2Screen from '../screens/Tab2Screen';

import { StackNavigator } from './StackNavigator';

const Tab = createBottomTabNavigator();

export const Tabs = () => {

return (

<Tab.Navigator>

<Tab.Screen name="Tab1Screen" component={Tab1Screen} />

<Tab.Screen name="Tab2Screen" component={Tab2Screen} />

<Tab.Screen name="StackNavigator" component={StackNavigator} />

</Tab.Navigator>

);

};

**Tab1Screen.tsx:**

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

import React from 'react';

const Tab1Screen = () => {

return (

<View>

<Text>Tab1Screen</Text>

</View>

);

};

export default Tab1Screen;

Ahora solo queda utilizarlo en el **drawer** para no perder toda esa funcionalidad que sería la principal. Para ello, vamos al **MenuLateral.tsx** y en vez de utilizar el **StackNavigator** implementamos el **Tabs**:

...

import {Tabs} from './Tabs';

...

return (

<Drawer.Navigator

screenOptions={{

drawerType: width >= 768 ? 'permanent' : 'front',

}}

drawerContent={props => <MenuInterno {...props} />}>

<Drawer.Screen name="Tabs" component={Tabs} />

<Drawer.Screen

name="SettingsScreen"

options={{title: 'ajustes'}}

component={SettingsScreen}

/>

</Drawer.Navigator>

);

...

{/\* Opciones de menú \*/}

<View style={styles.menuContainer}>

<TouchableOpacity

onPress={() => navigation.navigate('Tabs')}

style={styles.menuButton}>

<Text style={styles.menuText}>Navegación</Text>

</TouchableOpacity>

...

**PERSONALIZANDO EL BOTTOM TAB NAVIGATOR**

En **Tab.tsx** vamos a utilizar dentro de **<Tab.Navigator>** la propiedad **screenOptions** para todo lo relacionado a los botones y bordes del tab y **sceneContainerStyle** para la página en concreto:

import React from 'react';

import {createBottomTabNavigator} from '@react-navigation/bottom-tabs';

import Tab1Screen from '../screens/Tab1Screen';

import Tab2Screen from '../screens/Tab2Screen';

import {StackNavigator} from './StackNavigator';

import {colors} from '../theme/appTheme';

const Tab = createBottomTabNavigator();

export const Tabs = () => {

return (

<Tab.Navigator

sceneContainerStyle={{

backgroundColor: 'white',

}}

screenOptions={{

tabBarActiveTintColor: colors.primary,

tabBarStyle: {

borderTopColor: colors.primary,

borderTopWidth: 0,

elevation: 0,

},

tabBarLabelStyle:{

fontSize: 15,

}

}}>

<Tab.Screen

name="Tab1Screen"

options={{title: 'Tab1'}}

component={Tab1Screen}

/>

<Tab.Screen

name="Tab2Screen"

options={{title: 'Tab2'}}

component={Tab2Screen}

/>

<Tab.Screen

name="StackNavigator"

options={{title: 'Stack'}}

component={StackNavigator}

/>

</Tab.Navigator>

);

};

En el **appTheme.tsx** creamos la paleta de colores:

...

export const colors = {

primary: '#5856D6',

}

...

**PREPARAR EL ESPACIO PARA EL ÍCONO**

Para colocar de forma global los íconos debemos utilizar la propiedad **TabBarIcon** dentro de **screenOptions**, utilizando el argumento **route** de la misma.

...

const Tab = createBottomTabNavigator();

export const Tabs = () => {

return (

<Tab.Navigator

sceneContainerStyle={{

backgroundColor: 'white',

}}

screenOptions={({route}) => ({

...

// Forma global de importar íconos

tabBarIcon: ({color, focused, size}) => {

let iconName: string = '';

switch (route.name) {

case 'Tab1Screen':

iconName = 'T1';

break;

case 'Tab2Screen':

iconName = 'T2';

break;

case 'StackNavigator':

iconName = 'St';

break;

default:

break;

}

return <Text style={{color}}>{iconName}</Text>;

},

})}>

{/\* UNA FORMA DE IMPORTAR UN ÍCONO \*/}

{/\* <Tab.Screen

name="Tab1Screen"

options={{

title: 'Tab1',

tabBarIcon: props => <Text style={{color: props.color}}>T1</Text>,

}}

component={Tab1Screen}

/> \*/}

...

**MATERIAL BOTTOM TAB NAVIGATOR**

Debemos instalarlo con el commando:

**npm install @react-navigation/material-bottom-tabs react-native-paper react-native-vector-icons**

Ver la documentación aquí: <https://reactnavigation.org/docs/material-bottom-tab-navigator/>

Ahora en **navigaton/Tabs.tsx** vamos a separar un tab para ios y otro para Android. El de IOS va a ser el que ya teníamos. El de Android vamos hacerlo con el paquete recién instalado. Sería el mismo código casi, utilizamos **Platform** para saber si estamos en ios o android y retornar cierto Tab.

...

import {createMaterialBottomTabNavigator} from '@react-navigation/material-bottom-tabs';

import {Platform, Text} from 'react-native';

...

export const Tabsp = () => {

return Platform.OS === 'ios' ? <TabsIOS /> : <TabsAndroid />;

};

const BottomTabAndroid = createMaterialBottomTabNavigator();

const TabsAndroid = () => {

return (

<BottomTabAndroid.Navigator

sceneAnimationEnabled={true}

barStyle={{

backgroundColor: colors.primary,

}}

screenOptions={({route}) => ({

// Forma global de importar íconos

tabBarIcon: ({color, focused}) => {

let iconName: string = '';

switch (route.name) {

case 'Tab1Screen':

iconName = 'T1';

break;

case 'Tab2Screen':

iconName = 'T2';

break;

case 'StackNavigator':

iconName = 'St';

break;

default:

break;

}

return <Text style={{color}}>{iconName}</Text>;

},

})}>

<BottomTabAndroid.Screen

name="Tab1Screen"

options={{

title: 'Tab1',

}}

component={Tab1Screen}

/>

<BottomTabAndroid.Screen

name="Tab2Screen"

options={{title: 'Tab2'}}

component={Tab2Screen}

/>

<BottomTabAndroid.Screen

name="StackNavigator"

options={{title: 'Stack'}}

component={StackNavigator}

/>

</BottomTabAndroid.Navigator>

);

};

const BottomTabIOS = createBottomTabNavigator();

const TabsIOS = () => {

return (

...

**MATERIAL TOP TAB NAVIGATOR**

Ver documentación aquí: <https://reactnavigation.org/docs/material-top-tab-navigator/>

Debemos instalar los comandos:

**npm install @react-navigation/material-top-tabs react-native-tab-view**

**npm install react-native-pager-view**

**npx pod-install ios** (solamente si estamos en mac)

Luego creamos en la carpeta **navigation** el archivo **TopTabNavigator.tsx** y 3 screen que sirvan como ejemplo para la navegación:

import React from 'react';

import {createMaterialTopTabNavigator} from '@react-navigation/material-top-tabs';

import ChatScreen from '../screens/ChatScreen';

import ContacScreen from '../screens/ContacScreen';

import AlbumScreen from '../screens/AlbumScreen';

const Tab = createMaterialTopTabNavigator();

export const TopTabNavigator = () => {

return (

<Tab.Navigator>

<Tab.Screen name="ChatScreen" component={ChatScreen} />

<Tab.Screen name="ContacScreen" component={ContacScreen} />

<Tab.Screen name="AlbumScreen" component={AlbumScreen} />

</Tab.Navigator>

);

};

A continuación, utilizamos este TopTab en **Tabs.tsx** en lugar de Tab2Screen.

**PERSONALIZANDO EL MATERIAL TOP TAB NAVIGATOR**

En **TopTabNavigator.tsx** utilizamos distintas propiedades para personalizarlo:

...

<Tab.Navigator

sceneContainerStyle={{

backgroundColor: 'white',

}}

screenOptions={({route}) => ({

tabBarPressColor: colors.primary,

tabBarShowIcon: true,

tabBarIndicatorStyle: {

backgroundColor: colors.primary,

},

tabBarStyle: {

shadowColor: 'transparent',

elevation: 0,

},

tabBarIcon: ({color, focused}) => {

let iconName: string = '';

switch (route.name) {

case 'ChatScreen':

iconName = 'Ch';

break;

case 'ContacScreen':

iconName = 'Co';

break;

case 'AlbumScreen':

iconName = 'Al';

break;

default:

break;

}

return <Text style={{color}}>{iconName}</Text>;

},

})}>

...

**ÍCONOS EN NUESTRA APLICACIÓN - ANDROID**

Ver la documentación aquí: <https://github.com/oblador/react-native-vector-icons>

Documentación de IonIcons: <https://ionic.io/ionicons/>

Lo instalamos con el comando:

**npm install react-native-vector-icons**

Vamos a la ruta **Android/app/build.gradle** y pegamos la siguiente línea si queremos descargar todos los paquetes de íconos (no recomendado):

apply from: "../../node\_modules/react-native-vector-icons/fonts.gradle"

Si queremos usar cierto paquete, antes de agregar esa línea escribimos:

project.ext.vectoricons = [

iconFontNames: [ 'IonIcons.ttf' ] // Name of the font files you want to copy

]

(Puede que esta línea de código no funcione, en ese caso borrarla y dejar solo lo de **apply…**)

Ahora en la screen **Tab1Screen.tsx** importamos el paquete de íconos y lo utilizamos para mostrarlo:

import React from 'react';

import Icon from 'react-native-vector-icons/Ionicons';

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

import {styles} from '../theme/appTheme';

const Tab1Screen = () => {

return (

<View style={styles.globalMargin}>

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

<Text>

<Icon name="airplane-outline" size={80} color="#900" />

</Text>

</View>

);

};

export default Tab1Screen;

Luego instalar las dependencias en caso de que todavía no esté sujeto a Typescript con el comando:

**npm i -D @types/react-native-vector-icons**

**ÍCONOS EN NUESTRA APLICACIÓN – IOS**

Vamos a la ruta **carpetaDelProyecto/node\_modules/react-native-vector-icons/fonts**

Luego abrimos el archivo **carpetaDelProyecto/ios/nombrDelProyecto.xcworkspace** esto nos va abrir el proyecto en XCode.

Ahora en el nombre del proyecto que aparece a la izquierda arriba en xcode, hacemos click derecho y seleccionamos **new group**, y le damos el nombre de **Fonts**.

A continuación, de la carpeta Fonts que habíamos abierto en el primer paso, seleccionamos el paquete de íconos que queramos y lo dejamos caer en este group creado en el paso anterior. Seleccionar **copy ítems if needed** en caso que no esté tildado.

Luego en la documentación de github que deje en el video anterior, buscamos donde esta IOS y copiamos las líneas que están en el título **list of all available fonts**, y las pegamos dentro de xcode, abrimos la carpeta con el nombre del proyecto y buscamos el archivo **Info.plist**, click derecho y elegimos **open as>source code**.

Una vez ahí antes de que se cierre la última etiqueta **</dict>**, pegamos las líneas:

<key>UIAppFonts</key>

<array>

<string>AntDesign.ttf</string>

<string>Entypo.ttf</string>

<string>EvilIcons.ttf</string>

<string>Feather.ttf</string>

<string>FontAwesome.ttf</string>

<string>FontAwesome5\_Brands.ttf</string>

<string>FontAwesome5\_Regular.ttf</string>

<string>FontAwesome5\_Solid.ttf</string>

<string>Foundation.ttf</string>

<string>Ionicons.ttf</string>

<string>MaterialIcons.ttf</string>

<string>MaterialCommunityIcons.ttf</string>

<string>SimpleLineIcons.ttf</string>

<string>Octicons.ttf</string>

<string>Zocial.ttf</string>

<string>Fontisto.ttf</string>

</array>

Dejamos en verdad las fuentes que usemos, en este caso solo Ionicons.ttf, las otras las podemos borrar.

Y por último ejecutamos el comando **npx pod-install**

**SEC-9: CONTEXT y ESTADO GLOBAL DE LA APLICACIÓN**

**CREAR NUESTRO PRIMER CONTEXT EN TYPESCRIPT**

Creamos la carpeta **context** con el archivo **AuthContext.tsx**:

import React, {createContext} from 'react';

export interface AuthState {

isLoggedIn: boolean;

userName?: string;

favoriteIcon?: string;

}

export const authInitialState: AuthState = {

isLoggedIn: false,

userName: undefined,

favoriteIcon: undefined,

};

export interface AuthContextProps {

authState: AuthState;

signIn: () => void;

}

export const AuthContext = createContext({} as AuthContextProps);

export const AuthProvider = ({children}: any) => {

return (

<AuthContext.Provider

value={{

authState: authInitialState,

signIn: () => {},

}}>

{children}

</AuthContext.Provider>

);

};

Luego utilizamos el provider en el **App.tsx**:

import 'react-native-gesture-handler';

import React from 'react';

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

import {MenuLateral} from './src/navigation/MenuLateral';

import {AuthProvider} from './src/context/AuthContext';

const App = () => {

return (

<NavigationContainer>

<AppState>

<MenuLateral />

</AppState>

</NavigationContainer>

);

};

const AppState = ({children}: any) => {

return <AuthProvider>{children}</AuthProvider>;

};

export default App;

**CONSUMIR EL CONTEXT – AUTHSTATE**

En la screen **SettingsScreen.tsx** consumimos el context:

import React, {useContext} from 'react';

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

// import {useSafeAreaInsets} from 'react-native-safe-area-context';

import {styles} from '../theme/appTheme';

import {AuthContext} from '../context/AuthContext';

export const SettingsScreen = () => {

// const insets = useSafeAreaInsets();

const {authState} = useContext(AuthContext);

return (

<View style={{...styles.globalMargin /\* marginTop: insets.top\*/}}>

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

<Text>{JSON.stringify(authState, null, 4)}</Text>

</View>

);

};

**REDUCER -useReducer**

Vamos a crear en la carpeta **context** el archivo **AuthReducer.tsx**:

import {AuthState} from './AuthContext';

export const authReducer = (state: AuthState, action: any): AuthState => {

return state;

};

Y ahora en el **AuthContext.tsx** vamos a utilizar este reducer:

...

export const AuthProvider = ({children}: any) => {

const [authState, dispatch] = useReducer(authReducer, authInitialState);

return (

<AuthContext.Provider

value={{

authState,

signIn: () => {},

}}>

{children}

</AuthContext.Provider>

);

};

**DISPARAR ACCIONES**

Ahora debemos crear en el **AuthReducer.tsx** las distintas acciones que modificaran al state:

import {AuthState} from './AuthContext';

type AuthAction = {type: 'signIn'};

export const authReducer = (

state: AuthState,

action: AuthAction,

): AuthState => {

switch (action.type) {

case 'signIn':

return {

...state,

isLoggedIn: true,

userName: 'no-userName-yet',

};

default:

return state;

}

};

Y con esto armado podemos ejecutar el **dispatch** en el **AuthProvider**:

...

export const AuthProvider = ({children}: any) => {

const [authState, dispatch] = useReducer(authReducer, authInitialState);

const signIn = () => {

dispatch({type: 'signIn'});

};

return (

<AuthContext.Provider

value={{

authState,

signIn,

}}>

{children}

</AuthContext.Provider>

);

};

**DISPARAR EL SIGNIN DESDE OTRA PANTALLA**

En la screen **ContactScreen.tsx** vamos a crear un Button que dispare la acción del signIn:

import React, {useContext} from 'react';

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

import {styles} from '../theme/appTheme';

import {AuthContext} from '../context/AuthContext';

const ContacScreen = () => {

const {signIn, authState} = useContext(AuthContext);

return (

<View style={styles.globalMargin}>

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

{!authState.isLoggedIn && <Button title="Sign In" onPress={signIn} />}

</View>

);

};

export default ContacScreen;

Y ahora si vamos a la pantalla de **SettingsScreen.tsx** en el emulador, vamos a ver que el state cambió.

**TOUCHABLEICON – CUSTOM COMPONENT**

En la carpeta **components** creamos el componente **TouchableIcon.tsx**, el cual será un ícono al que lo podes clickear y saber que nombre lleva para luego usarlo en la propiedad de icono favorito en el state:

import React from 'react';

import {TouchableOpacity} from 'react-native';

import Icon from 'react-native-vector-icons/Ionicons';

import {colors} from '../theme/appTheme';

interface Props {

iconName: string;

}

const TouchableIcon = ({iconName}: Props) => {

return (

<TouchableOpacity onPress={() => console.log(iconName)}>

<Icon name={iconName} size={80} color={colors.primary} />

</TouchableOpacity>

);

};

export default TouchableIcon;

Ahora en el **TabScreen1.tsx** importamos este componente y lo reutilizamos con todos los íconos que queramos poner:

import React from 'react';

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

import {styles} from '../theme/appTheme';

import TouchableIcon from '../components/TouchableIcon';

const Tab1Screen = () => {

return (

<View style={styles.globalMargin}>

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

<Text>

<TouchableIcon iconName="airplane-outline" />

<TouchableIcon iconName="attach-outline" />

<TouchableIcon iconName="bonfire-outline" />

<TouchableIcon iconName="calculator-outline" />

<TouchableIcon iconName="chatbubble-ellipses-outline" />

<TouchableIcon iconName="images-outline" />

<TouchableIcon iconName="leaf-outline" />

</Text>

</View>

);

};

export default Tab1Screen;

**CAMBIAR EL ÍCONO FAVORITO – AUTHSTATE**

Ahora en el **AuthReducer.tsx** debemos crear otra action que ejecute el cambio del state para el ícono:

import {AuthState} from './AuthContext';

type AuthAction = {type: 'signIn'} | {type: 'changeFavIcon'; payload: string};

export const authReducer = (

state: AuthState,

action: AuthAction,

): AuthState => {

switch (action.type) {

case 'signIn':

return {

...state,

isLoggedIn: true,

userName: 'no-userName-yet',

};

case 'changeFavIcon':

return {

...state,

favoriteIcon: action.payload,

};

default:

return state;

}

};

En el **AuthContext.tsx** ejecutamos el **dispatch** de esta action:

...

export interface AuthContextProps {

authState: AuthState;

signIn: () => void;

changeFavoriteIcon: (iconName: string) => void

}

export const AuthContext = createContext({} as AuthContextProps);

export const AuthProvider = ({children}: any) => {

const [authState, dispatch] = useReducer(authReducer, authInitialState);

const signIn = () => {

dispatch({type: 'signIn'});

};

const changeFavoriteIcon = (iconName:string) => {

dispatch({type: 'changeFavIcon', payload: iconName})

}

return (

<AuthContext.Provider

value={{

authState,

signIn,

changeFavoriteIcon

}}>

{children}

</AuthContext.Provider>

);

};

Con esto ya podemos llamar a la función **changeFavoriteIcon** en el componente **TouchableIcon.tsx** y hacer cambiar el ícono favorito haciendo click en cada ícono:

import React, { useContext } from 'react';

import {TouchableOpacity} from 'react-native';

import Icon from 'react-native-vector-icons/Ionicons';

import {colors} from '../theme/appTheme';

import { AuthContext } from '../context/AuthContext';

interface Props {

iconName: string;

}

const TouchableIcon = ({iconName}: Props) => {

const {changeFavoriteIcon} = useContext(AuthContext)

return (

<TouchableOpacity onPress={() => changeFavoriteIcon(iconName)}>

<Icon name={iconName} size={80} color={colors.primary} />

</TouchableOpacity>

);

};

export default TouchableIcon;

Y en **SettingsScreen.tsx** podemos ver el ícono favorito:

...

const {authState} = useContext(AuthContext);

return (

<View style={{...styles.globalMargin /\* marginTop: insets.top\*/}}>

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

<Text>{JSON.stringify(authState, null, 4)}</Text>

{authState.favoriteIcon && (

<Icon name={authState.favoriteIcon} size={150} color={colors.primary} />

)}

</View>

);

...

**BORRAR INFORMACIÓN DEL AUTHSTATE**

En el **AuthReducer.tsx** creamos la nueva action para eliminar el state a como estaba:

...

type AuthAction =

...

| {type: 'signOut'};

...

case 'signOut':

return {

...state,

isLoggedIn: false,

userName: undefined,

favoriteIcon: undefined,

}

...

En el **AuthContext.tsx** creamos la función que ejecuta el dispatch de esta action:

...

export interface AuthContextProps {

...

signOut: () => void;

}

...

const signOut = () => {

dispatch({type: 'signOut'});

};

...

Y ahora en **ContactScreen.tsx** renderizamos o el botón de signIn o el de signOut:

...

{!authState.isLoggedIn ? (

<Button title="Sign In" onPress={signIn} />

) : (

<Button title="Sign Out" onPress={signOut} />

)}

...

**CAMBIAR EL USERNAME**

En el **AuthReducer.tsx** creamos la nueva action para cambiar el userName:

...

type AuthAction =

...

| {type: 'changeUserName'; payload: string};

...

case 'changeUserName':

return {

...state,

userName: action.payload,

};

...

En el **AuthContext.tsx** creamos la función que ejecuta el dispatch de esta action:

...

export interface AuthContextProps {

...

changeUserName: (name: string) => void;

}

...

const changeUserName = (name: string) => {

dispatch({type: 'changeUserName', payload: name})

}

...

Y ahora en **PersonaScreen.tsx** creamos un **useEffect** que ejecute esta función y cambie el username mediante los **params** que ya estábamos utilizando:

import React, {useContext, useEffect} from 'react';

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

import {styles} from '../theme/appTheme';

import {StackScreenProps} from '@react-navigation/stack';

import {RootStackParams} from '../navigation/StackNavigator';

import {AuthContext} from '../context/AuthContext';

// interface RouterParams {

// id: number;

// name: string;

// }

interface Props extends StackScreenProps<RootStackParams, 'PersonaScreen'> {}

export const PersonaScreen = ({navigation, route}: Props) => {

// const params = route.params as RouterParams;

const params = route.params;

const {changeUserName} = useContext(AuthContext);

useEffect(() => {

navigation.setOptions({

title: params.name,

});

}, []);

useEffect(() => {

changeUserName(params.name);

}, []);

return (

<View style={styles.globalMargin}>

<Text style={styles.title}>{JSON.stringify(params, null, 3)}</Text>

</View>

);

};

**SEC-10: APLICACIÓN DE PELÍCULAS**

**INICIO DE PROYECTO**

Creamos el proyecto, y luego nos dirigimos a la página <https://www.themoviedb.org/> donde nos creamos un usuario y utilizamos su API.

La mía es: 29ae7ede5815710001d04a89c2736fa2

Luego vamos a la documentación: <https://developer.themoviedb.org/docs>

Y vamos a utilizar la API:

<https://api.themoviedb.org/3/movie/now_playing?api_key=29ae7ede5815710001d04a89c2736fa2>

**NAVEGACIÓN ENTRE PANTALLAS**

Hacemos toda la instalación de React Navigation:

<https://reactnavigation.org/docs/getting-started/>

Y luego del **Stack Navigator**: <https://reactnavigation.org/docs/stack-navigator/>

En la sección 8 está explicado sino.

Luego creamos la carpeta **src** con las carpetas **navigation, components, api, hooks, screen**.

Ahora dentro de **navigation** creamos el archivo **Navigation.tsx**:

import React from 'react';

import {createStackNavigator} from '@react-navigation/stack';

import HomeScreen from '../screens/HomeScreen';

import DetailScreen from '../screens/DetailScreen';

const Stack = createStackNavigator();

export const Navigation = () => {

return (

<Stack.Navigator

screenOptions={{

headerShown: false,

}}>

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

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

</Stack.Navigator>

);

};

Y dentro de la carpeta **screens**, creamos **HomeScreen.tsx** y **DetailScreen.tsx**:

import React from 'react';

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

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

const HomeScreen = () => {

const navigation = useNavigation<any>();

return (

<View>

<Text>HomeScreen</Text>

<Button

title="ir a Detail"

onPress={() => navigation.navigate('DetailScreen')}

/>

</View>

);

};

export default HomeScreen;

Utilizamos el **useNavigation()** para verificar que esté funcionando la navegación del stack.

Por último en el **App.tsx** importamos este **Navigation.tsx**:

import 'react-native-gesture-handler';

import React from 'react';

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

import {Navigation} from './src/navigation/Navigation';

const App = () => {

return (

<NavigationContainer>

<Navigation />

</NavigationContainer>

);

};

export default App;

**OBTENER PELÍCULAS EN CARTELERA**

En la carpeta **api** nos creamos el archivo **moviesDB.tsx** en la cual crearemos nuestro URL para todas las peticiones a la api de películas:

import axios from 'axios';

const moviesDB = axios.create({

baseURL: 'https://api.themoviedb.org/3/movie',

params: {

api\_key: '29ae7ede5815710001d04a89c2736fa2',

language: 'es-ES',

},

});

export default moviesDB;

Luego en **HomeScreen.tsx** ejecutamos un useEffect el cual haga la petición a todas las películas:

import React, {useEffect} from 'react';

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

import moviesDB from '../api/moviesDB';

const HomeScreen = () => {

useEffect(() => {

moviesDB.get('/now\_playing').then(resp => console.log(resp.data));

}, []);

return (

<View>

<Text>HomeScreen</Text>

</View>

);

};

export default HomeScreen;

**TIPADO DE TheMovieDB**

Debemos tipar la respuesta de la api de movies, por lo que creamos la carpeta **interfaces** con el archivo **movieInerface.tsx** y utilizamos la página <https://app.quicktype.io/> :

export interface MovieDbNowPlaying {

dates: Dates;

page: number;

results: Movie[];

total\_pages: number;

total\_results: number;

}

export interface Dates {

maximum: Date;

minimum: Date;

}

export interface Movie {

adult: boolean;

backdrop\_path: string;

genre\_ids: number[];

id: number;

original\_language: OriginalLanguage;

original\_title: string;

overview: string;

popularity: number;

poster\_path: string;

release\_date: Date;

title: string;

video: boolean;

vote\_average: number;

vote\_count: number;

}

export enum OriginalLanguage {

En = 'en',

Fi = 'fi',

Ja = 'ja',

}

Ahora en el **HomeScreen.tsx** podemos marcarle el tipado a la response:

...

useEffect(() => {

moviesDB

.get<MovieDbNowPlaying>('/now\_playing')

.then(resp => console.log(resp.data));

}, []);

...

**useMovies – HOOK**

En la carpeta **hooks** creamos el archvo **useMovies.tsx** el cual va a contener la lógica para hacer el pedido a la api y conseguir las películas:

import {useEffect, useState} from 'react';

import moviesDB from '../api/moviesDB';

import {Movie, MovieDbNowPlaying} from '../interfaces/movieInterface';

export const useMovies = () => {

const [isLoading, setIsLoading] = useState(true);

const [moviesInCine, setMoviesInCine] = useState<Movie[]>([]);

useEffect(() => {

getMovies();

}, []);

const getMovies = async () => {

const resp = await moviesDB.get<MovieDbNowPlaying>('/now\_playing');

setMoviesInCine(resp.data.results);

setIsLoading(false);

};

return {

moviesInCine,

isLoading,

};

};

Ahora con ese hook creado podemos borrar la lógica del **HomeScreen.tsx** y solo usar este hook:

import React from 'react';

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

import {useMovies} from '../hooks/useMovies';

const HomeScreen = () => {

const {moviesInCine, isLoading} = useMovies();

if (isLoading) {

<View style={{flex: 1, justifyContent: 'center', alignItems: 'center'}}>

<ActivityIndicator color="red" size={100} />

</View>;

}

return (

<View>

<Text>HomeScreen</Text>

</View>

);

};

export default HomeScreen;

**MOSTRAR EL PÓSTER DE LA PELÍCULA**

En la carpeta **components** creamos el archivo **MoviePoster.tsx** el cual será un componente reutilizable que muestre todos los posters de las películas:

import React from 'react';

import {View, Image, StyleSheet} from 'react-native';

import {Movie} from '../interfaces/movieInterface';

interface Props {

movie: Movie;

}

const MoviePoster = ({movie}: Props) => {

const uri = `https://image.tmdb.org/t/p/w500${movie?.poster\_path}`;

return (

<View style={{width: 300, height: 420}}>

<View style={styles.imageContainer}>

<Image source={{uri}} style={styles.image} />

</View>

</View>

);

};

const styles = StyleSheet.create({

imageContainer: {

backgroundColor: 'white',

flex: 1,

borderRadius: 18,

shadowColor: '#000000',

shadowOffset: {

width: 0,

height: 10,

},

shadowOpacity: 0.25,

shadowRadius: 7,

elevation: 10,

},

image: {

flex: 1,

borderRadius: 18,

},

});

export default MoviePoster;

Ahora desde **HomeScreen.tsx** debemos importar este componente y pasarle por props la movie:

import React from 'react';

import {View, ActivityIndicator} from 'react-native';

import {useMovies} from '../hooks/useMovies';

import MoviePoster from '../components/MoviePoster';

import { useSafeAreaInsets } from 'react-native-safe-area-context';

const HomeScreen = () => {

const {moviesInCine, isLoading} = useMovies();

const {top} = useSafeAreaInsets();

if (isLoading) {

<View style={{flex: 1, justifyContent: 'center', alignItems: 'center'}}>

<ActivityIndicator color="red" size={100} />

</View>;

}

return (

<View style={{marginTop: top + 20}}>

<MoviePoster movie={moviesInCine[0]}/>

</View>

);

};

export default HomeScreen;

**CAROUSEL DE TARJETAS**

Vamos a la documentación en: <https://github.com/dohooo/react-native-reanimated-carousel>

Y corremos los comandos:

**npm install react-native-reanimated-carousel**

(Hay que tener instalado react-native-reanimated: leer la documentación)

Y ahora en el **HomeScreen.tsx** importamos este **Carousel** y lo utilizamos:

import React from 'react';

import {View, ActivityIndicator, Dimensions} from 'react-native';

import {useMovies} from '../hooks/useMovies';

import Carousel from 'react-native-reanimated-carousel';

import MoviePoster from '../components/MoviePoster';

import {useSafeAreaInsets} from 'react-native-safe-area-context';

const {width: windowWidth} = Dimensions.get('window');

const HomeScreen = () => {

const {moviesInCine, isLoading} = useMovies();

const {top} = useSafeAreaInsets();

if (isLoading) {

<View style={{flex: 1, justifyContent: 'center', alignItems: 'center'}}>

<ActivityIndicator color="red" size={100} />

</View>;

}

return (

<View style={{marginTop: top + 20}}>

<View style={{height: 440}}>

<Carousel

// Just one of the many styles from the Carousel module

mode="parallax"

// This style prop regards to the carousel container not to the item itself

style={{width: windowWidth, justifyContent: 'center'}}

// Paging in false allows to do superfast scroll

pagingEnabled={false}

// and that superfast scroll stops on multiples of windowSize

windowSize={2}

// the snap helps to stop exactly in 1 item, no in the middle of two or so

snapEnabled

// This props are for the item in the middle

width={300}

height={420}

modeConfig={{

// How the "main" item will look

parallaxScrollingScale: 0.9,

// How separate the adjacent items will be

parallaxScrollingOffset: 40,

// How big the adjacent items will look compared to the "main" item

parallaxAdjacentItemScale: 0.75,

}}

data={moviesInCine}

renderItem={({item}) => <MoviePoster movie={item} />}

/>

</View>

</View>

);

};

export default HomeScreen;

**FLATLIST DE PELÍCULAS**

En el componente **HomeScreen.tsx** por debajo del carousel vamos a agregar un **<FlatList/>** que sería un carousel también, y además envolvemos todo en un **<ScrollView/>** para poder hacer un scroll vertical de la pantalla:

return (

<ScrollView>

<View style={{marginTop: top + 20}}>

{/\* Carousel Principal \*/}

<View style={{height: 440}}>

...

</View>

{/\* Películas populares \*/}

<View style={{height: 260}}>

<Text style={{fontSize: 30, fontWeight: 'bold'}}>Populares</Text>

<FlatList

data={moviesInCine}

renderItem={({item}: any) => (

<MoviePoster movie={item} width={140} height={200} />

)}

keyExtractor={item => item.id.toString()}

// Hace que el scroll sea horizontal

horizontal={true}

// Quita la barra de scroll

showsHorizontalScrollIndicator={false}

/>

</View>

</View>

</ScrollView>

);

Además en el componente **MoviePoster.tsx** agregamos algunas props para modificar el width y el height del póster:

...

interface Props {

movie: Movie;

height?: number;

width?: number;

}

const MoviePoster = ({movie, height = 420, width = 300}: Props) => {

const uri = `https://image.tmdb.org/t/p/w500${movie?.poster\_path}`;

return (

<View style={{width, height, marginHorizontal: 8}}>

<View style={styles.imageContainer}>

<Image source={{uri}} style={styles.image} />

</View>

</View>

);

};

...

**COMPONENTE HORIZONTALSLIDER**

En la carpeta **components** creamos el archivo **HorizontalSlider.tsx**, el cual contendrá la lógica del **<FlatList />** del video anterior para ser reutilizable:

import React from 'react';

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

import MoviePoster from './MoviePoster';

import {Movie} from '../interfaces/movieInterface';

interface Props {

title?: string;

movies: Movie[];

}

const HorizontalSlider = ({title, movies}: Props) => {

return (

<View style={{height: title ? 260 : 220}}>

{title && (

<Text style={{fontSize: 30, fontWeight: 'bold', marginLeft: 10}}>

{title}

</Text>

)}

<FlatList

data={movies}

renderItem={({item}: any) => (

<MoviePoster movie={item} width={140} height={200} />

)}

keyExtractor={item => item.id.toString()}

// Hace que el scroll sea horizontal

horizontal={true}

// Quita la barra de scroll

showsHorizontalScrollIndicator={false}

/>

</View>

);

};

export default HorizontalSlider;

Notemos que si no viene un **title** por props el height del View cambia para mantener una distancia correcta.

**PELÍCULAS POPULARES, MEJOR CALIFICADAS y PRÓXIMAS A SALIR**

Ahora vamos a usar otro endpoint para traer diferentes películas, lo que va a hacer que cambiemos un poco la interface que habíamos creado **movieInterface.tsx**. Le modificamos el nombre a **MovieDBNowPlaying**  por **MovieDBMoviesResponse** y además las **dates** van a ser opcionales:

export interface MovieDBMoviesResponse {

dates?: Dates;

...

}

En **useMovies.tsx** ahora cambiamos el nombre de la importación de la interface, y hacemos los distintos llamados a los endpoint:

import {useEffect, useState} from 'react';

import moviesDB from '../api/moviesDB';

import {Movie, MovieDBMoviesResponse} from '../interfaces/movieInterface';

export const useMovies = () => {

const [isLoading, setIsLoading] = useState(true);

const [moviesInCine, setMoviesInCine] = useState<Movie[]>([]);

const [moviesPopulars, setMoviesPopulars] = useState<Movie[]>([]);

useEffect(() => {

getMovies();

}, []);

const getMovies = async () => {

const respNowPlaying = await moviesDB.get<MovieDBMoviesResponse>('/now\_playing');

const respPopular = await moviesDB.get<MovieDBMoviesResponse>('/popular');

const respTopRated = await moviesDB.get<MovieDBMoviesResponse>('/top\_rated');

const respUpcoming = await moviesDB.get<MovieDBMoviesResponse>('/upcoming');

setMoviesInCine(respNowPlaying.data.results);

setMoviesPopulars(respPopular.data.results);

setIsLoading(false);

};

return {

moviesInCine,

isLoading,

moviesPopulars,

};

};

(No agregue más useState porque en el próximo video vamos a hacer todas las peticiones de una sola vez para optimizar el tiempo de peticiones)

En el **HomeScreen.tsx** usamos este array de **moviesPopulars**:

...

{/\* Películas populares \*/}

<HorizontalSlider title="Populares" movies={moviesPopulars} />

...

**MÚLTIPLES PETICIONES DE FORMA SIMULTÁNEA**

Ahora en el **useMovies.tsx** vamos a crear una interface la cual va a tener el nombre de las distintas peticiones que haremos.

Luego usaremos un solo **useState** el cual tendrá el tipado de esta interface.

Y usaremos **Promises.all()** para hacer el llamado de todas las peticiones de una sola vez:

import {useEffect, useState} from 'react';

import moviesDB from '../api/moviesDB';

import {Movie, MovieDBMoviesResponse} from '../interfaces/movieInterface';

interface MoviesState {

nowPlaying: Movie[];

popular: Movie[];

topRated: Movie[];

upcoming: Movie[];

}

export const useMovies = () => {

const [isLoading, setIsLoading] = useState(true);

const [moviesState, setMoviesState] = useState<MoviesState>({

nowPlaying: [],

popular: [],

topRated: [],

upcoming: [],

});

useEffect(() => {

getMovies();

}, []);

const getMovies = async () => {

const nowPlayingPromise = moviesDB.get<MovieDBMoviesResponse>('/now\_playing');

const PopularPromise = moviesDB.get<MovieDBMoviesResponse>('/popular');

const topRatedPromise = moviesDB.get<MovieDBMoviesResponse>('/top\_rated');

const upcomingPromise = moviesDB.get<MovieDBMoviesResponse>('/upcoming');

const resps = await Promise.all([

nowPlayingPromise,

PopularPromise,

topRatedPromise,

upcomingPromise,

]);

setMoviesState({

nowPlaying: resps[0].data.results,

popular: resps[1].data.results,

topRated: resps[2].data.results,

upcoming: resps[3].data.results,

});

setIsLoading(false);

};

return {

...moviesState,

isLoading,

};

};

Notese que en el return usamos el spread operation del state para que ahora en el **HomeScreen.tsx** pueda destructurarlo y utilizar el **<HorizontalSlider />**:

...

const HomeScreen = () => {

const {nowPlaying, popular, topRated, upcoming, isLoading} = useMovies();

...

<HorizontalSlider title="Populares" movies={popular} />

<HorizontalSlider title="Mejor Calificadas" movies={topRated} />

<HorizontalSlider title="Próximas a estrenarse" movies={upcoming} />

...

**NAVEGAR A LA PANTALLA DE DETALLES**

En el componente **MoviePoster.tsx** vamos a utilizar el hook **useNavigation()** para navegar a la screen **DetailScreen.tsx**, además vamos a envolver todo en un **<TouchableOpacity/>** y utilizar ahí el navigation:

...

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

...

const MoviePoster = ({movie, height = 420, width = 300}: Props) => {

const uri = `https://image.tmdb.org/t/p/w500${movie?.poster\_path}`;

const navigation = useNavigation<any>();

return (

<TouchableOpacity

onPress={() => navigation.navigate('DetailScreen', movie)}

activeOpacity={0.8}

style={{

width,

height,

marginHorizontal: 2,

paddingBottom: 20,

paddingHorizontal: 7,

}}>

<View style={styles.imageContainer}>

<Image source={{uri}} style={styles.image} />

</View>

</TouchableOpacity>

);

};

...

En el **onPress** estamos diciendo a donde navegar y pasamos por **params** la movie.

Luego en **Navigation.tsx** vamos a especificar un **type** para el **Stack** con el objetivo de tener el tipado de las movies en DetailScreen:

...

export type RootStackParams = {

HomeScreen: undefined;

DetailScreen: Movie;

}

const Stack = createStackNavigator<RootStackParams>();

...

Por último, en **DetailScreen.tsx** creamos la **interface Props** e importamos este **RootStackParams** y con eso vamos a tener los parámetros en las **props.route**, es decir, la movie:

import React from 'react';

import {StackScreenProps} from '@react-navigation/stack';

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

import {RootStackParams} from '../navigation/Navigation';

interface Props extends StackScreenProps<RootStackParams, 'DetailScreen'> {}

const DetailScreen = ({route}: Props) => {

const movie = route.params;

return (

<View>

<Text>DetailScreen</Text>

</View>

);

};

export default DetailScreen;

**DISEÑO INICIAL DE LA PANTALLA DE DETALLES**

Diseñamos la screen **DetailScreen.tsx**:

import React from 'react';

import {StackScreenProps} from '@react-navigation/stack';

import {

View,

Text,

Image,

StyleSheet,

Dimensions,

ScrollView,

} from 'react-native';

import {RootStackParams} from '../navigation/Navigation';

const screenHeight = Dimensions.get('screen').height;

interface Props extends StackScreenProps<RootStackParams, 'DetailScreen'> {}

const DetailScreen = ({route}: Props) => {

const movie = route.params;

const uri = `https://image.tmdb.org/t/p/w500${movie?.poster\_path}`;

return (

<ScrollView>

<View style={styles.imageContainer}>

<View style={styles.imageBorder}>

<Image source={{uri}} style={styles.posterImage} />

</View>

</View>

<View style={styles.marginContainer}>

<Text style={styles.subtitle}>

{movie.original\_title}

</Text>

<Text style={styles.title}>

{movie.title}

</Text>

</View>

</ScrollView>

);

};

const styles = StyleSheet.create({

imageContainer: {

width: '100%',

height: screenHeight \* 0.7,

backgroundColor: 'white',

shadowColor: '#000000',

shadowOffset: {

width: 0,

height: 10,

},

shadowOpacity: 0.25,

shadowRadius: 7,

elevation: 10,

borderBottomEndRadius: 25,

borderBottomStartRadius: 25,

},

imageBorder: {

flex: 1,

overflow: 'hidden',

borderBottomEndRadius: 25,

borderBottomStartRadius: 25,

},

posterImage: {

flex: 1,

},

marginContainer: {

marginHorizontal: 20,

marginTop: 20,

},

subtitle: {

fontSize: 16,

opacity: 0.8,

},

title: {

fontSize: 20,

fontWeight: 'bold',

},

});

export default DetailScreen;

**SOMBRAS E ÍCONOS DE NUESTRA APLICACIÓN**

Debemos descargarlos desde aquí: <https://github.com/oblador/react-native-vector-icons>

Buscar **íconos en nuestra aplicación** en este Word y vamos a encontrar bien el tutorial.

**useMovieDetails – CUSTOM HOOKS**

Vamos a crear en la carpeta **hook** un **usemovieDetail.tsx** para hacer las peticiones a los endpoints del detalle y al casting de una sola película, para ello debemos hacer nuestra **interface** para tener tipada la respuesta. Usemos la página **quicktype**.

import {useEffect, useState} from 'react';

import moviesDB from '../api/moviesDB';

import {MovieFull} from '../interfaces/movieInterface';

import {Cast, CreditsResponse} from '../interfaces/creditsInterface';

interface MovieDetails {

isLoading: boolean;

movieFull?: MovieFull;

cast: Cast[];

}

export const useMovieDetails = (movieId: number) => {

const [state, setState] = useState<MovieDetails>({

isLoading: true,

movieFull: undefined,

cast: [],

});

useEffect(() => {

getMovieDetails();

}, []);

const getMovieDetails = async () => {

const movieDetailsPromise = await moviesDB.get<MovieFull>(`/${movieId}`);

const castPromise = await moviesDB.get<CreditsResponse>(

`/${movieId}/credits`,

);

const [movieDetailsResp, castPromiseResp] = await Promise.all([

movieDetailsPromise,

castPromise,

]);

setState({

isLoading: false,

movieFull: movieDetailsResp.data,

cast: castPromiseResp.data.cast,

});

};

return {

...state,

};

};

**movieInterface.tsx:**

export interface MovieFull {

adult: boolean;

backdrop\_path: string;

belongs\_to\_collection: BelongsToCollection;

budget: number;

genres: Genre[];

homepage: string;

id: number;

imdb\_id: string;

original\_language: string;

original\_title: string;

overview: string;

popularity: number;

poster\_path: string;

production\_companies: ProductionCompany[];

production\_countries: ProductionCountry[];

release\_date: Date;

revenue: number;

runtime: number;

spoken\_languages: SpokenLanguage[];

status: string;

tagline: string;

title: string;

video: boolean;

vote\_average: number;

vote\_count: number;

}

export interface BelongsToCollection {

id: number;

name: string;

poster\_path: string;

backdrop\_path: string;

}

export interface Genre {

id: number;

name: string;

}

export interface ProductionCompany {

id: number;

logo\_path: null | string;

name: string;

origin\_country: string;

}

export interface ProductionCountry {

iso\_3166\_1: string;

name: string;

}

export interface SpokenLanguage {

english\_name: string;

iso\_639\_1: string;

name: string;

}

Creamos la interface **creditsInterface.tsx**:

export interface CreditsResponse {

id: number;

cast: Cast[];

crew: Cast[];

}

export interface Cast {

adult: boolean;

gender: number;

id: number;

known\_for\_department: Department;

name: string;

original\_name: string;

popularity: number;

profile\_path: null | string;

cast\_id?: number;

character?: string;

credit\_id: string;

order?: number;

department?: Department;

job?: string;

}

export enum Department {

Acting = 'Acting',

Art = 'Art',

Camera = 'Camera',

CostumeMakeUp = 'Costume & Make-Up',

Crew = 'Crew',

Directing = 'Directing',

Editing = 'Editing',

Lighting = 'Lighting',

Production = 'Production',

Sound = 'Sound',

VisualEffects = 'Visual Effects',

Writing = 'Writing',

}

Ahora en **DetailScreen.tsx** llamamos a este hook y le pasamos como parámetro el **id**, y podemos desestructurar todas las propiedades del state:

...

const DetailScreen = ({route}: Props) => {

const movie = route.params;

const uri = `https://image.tmdb.org/t/p/w500${movie?.poster\_path}`;

const {isLoading, cast, movieFull} = useMovieDetails(movie.id);

...

**DETALLES DE LA PELÍCULA**

Vamos a crear un componente que contenga los detalles de la película, para eso en la carpeta **components** creamos el archivo **MovieDetails.tsx**:

import React from 'react';

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

import currencyFormatter from 'currency-formatter';

import {MovieFull} from '../interfaces/movieInterface';

import {Cast} from '../interfaces/creditsInterface';

import Icon from 'react-native-vector-icons/Ionicons';

interface Props {

movieFull: MovieFull;

cast: Cast[];

}

const MovieDetails = ({movieFull, cast}: Props) => {

return (

<>

{/\* Detalles \*/}

<View style={{marginHorizontal: 20}}>

<View style={{flexDirection: 'row'}}>

<Icon name="star-outline" size={16} color="grey" />

<Text> {movieFull.vote\_average}</Text>

<Text style={{marginLeft: 5}}>

- {movieFull.genres.map(genre => genre.name).join(', ')}

</Text>

</View>

<Text style={{marginTop: 10, fontSize: 23, fontWeight: 'bold'}}>

Historia

</Text>

<Text style={{fontSize: 16}}>{movieFull.overview}</Text>

<Text style={{marginTop: 10, fontSize: 23, fontWeight: 'bold'}}>

Presupuesto

</Text>

<Text style={{fontSize: 18}}>

{currencyFormatter.format(movieFull.budget, {code: 'USD'})}

</Text>

</View>

</>

);

};

export default MovieDetails;

Veamos que recibe por props los datos de una película, y el casting de una película.

Además estamos usando el paquete **currency-formatter** para mostrar bien el presupuesto. Lo instalamos con el comando:

npm install currency-formatter

Y debemos instalar el tipado también:

**npm i -D @types/currency-formatter**

Ahora con todo esto armado, en **DetailScreen.tsx** ponemos un condicional utilizando el **isLoading** del **useMovieDetails()** que muestre este componente creado o un **<ActivityIndicator/>**:

...

const {isLoading, cast, movieFull} = useMovieDetails(movie.id);

return (

...

{isLoading ? (

<ActivityIndicator size={35} color="grey" style={{marginTop: 20}} />

) : (

<MovieDetails movieFull={movieFull!} cast={cast} />

)}

...

**COMPONENTE PARA MOSTRAR ACTORES**

Creamos en la carpeta **components** el archivo **CastItem.tsx** para mostrar todos los actores de las película:

import React from 'react';

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

import {Cast} from '../interfaces/creditsInterface';

interface Props {

actor: Cast;

}

const CastItem = ({actor}: Props) => {

const uri = `https://image.tmdb.org/t/p/w500${actor.profile\_path}`;

return (

<View style={styles.container}>

{

actor.profile\_path && <Image source={{uri}} style={styles.imageStyle} />

}

<View style={styles.actorInfo}>

<Text style={styles.actorName}>{actor.name}</Text>

<Text style={styles.actorCharacter}>{actor.character}</Text>

</View>

</View>

);

};

const styles = StyleSheet.create({

container: {

flexDirection: 'row',

borderRadius: 10,

backgroundColor: 'white',

shadowColor: '#000000',

shadowOffset: {

width: 0,

height: 10,

},

shadowOpacity: 0.25,

shadowRadius: 7,

elevation: 10,

},

imageStyle: {

width: 50,

height: 50,

borderRadius: 10,

},

actorInfo: {

marginLeft: 10,

},

actorName: {

fontSize: 18,

fontWeight: 'bold',

color: 'black',

},

actorCharacter: {

fontSize: 16,

color: 'grey',

},

});

export default CastItem;

Luego lo importamos en el componente **MovieDetail.tsx** y le pasamos por props por ahora solamente un actor, en el próximo video haremos el mapeo:

...

{/\* Casting \*/}

<View style={{marginTop: 10, marginBottom: 100}}>

<Text

style={{

marginTop: 10,

fontSize: 23,

fontWeight: 'bold',

color: 'black',

marginHorizontal: 20,

}}>

Actores

</Text>

<CastItem actor={cast[0]} />

</View>

...

**LISTA DE ACTORES**

Ahora en **MovieDetails.tsx** en vez de regresar un solo **<CastItem />** vamos a regresar un **<FlatList />** para mostrar todos los actores:

<FlatList

data={cast}

keyExtractor={item => item.id.toString()}

renderItem={({item}) => <CastItem actor={item} />}

horizontal={true}

showsHorizontalScrollIndicator={false}

style={{marginTop: 10, height: 70}}

/>

Y los **styles** en **CastItem.tsx** se modificaron un poco:

const styles = StyleSheet.create({

container: {

flexDirection: 'row',

height: 50,

borderRadius: 10,

backgroundColor: 'white',

shadowColor: '#000000',

shadowOffset: {

width: 0,

height: 10,

},

shadowOpacity: 0.25,

shadowRadius: 7,

elevation: 10,

marginLeft: 20,

paddingRight: 15,

},

imageStyle: {

width: 50,

height: 50,

borderRadius: 10,

},

actorInfo: {

marginLeft: 10,

marginTop: 4,

},

actorName: {

fontSize: 18,

fontWeight: 'bold',

color: 'black',

},

actorCharacter: {

fontSize: 16,

color: 'grey',

},

});

**BOTÓN PARA REGRESAR**

En **DetailScreen.tsx** vamos a implementar un **<TouchableOpacity>** para navegar hacia atrás, además vamos a utilizar el **navigation** de las **props**:

interface Props extends StackScreenProps<RootStackParams, 'DetailScreen'> {}

const DetailScreen = ({route, navigation}: Props) => {

...

{/\* Botón para regresar \*/}

<TouchableOpacity

style={styles.backButton}

onPress={() => navigation.pop()}>

<Icon name="arrow-back-outline" size={60} color="white" />

</TouchableOpacity>

...

backButton: {

position: 'absolute',

zIndex: 999,

elevation: 9,

top: 30,

left: 5,

},

...

**SEC-11: GRADIENTE ANIMADO – CONTEXT API**

**EJERCICIO PARA COMPRENDER EL FADE IN**

Para este ejercicio vamos a crear una nueva screen llamada **FadeScreen.tsx**, en la cual vamos a utilizar un **useRef** para crear un **Animated** con el valor en 0, en donde después en la función **fadeIn** especificamos las opciones para que pase a 1, y pasarle esta opacidad a los styles:

import React, {useRef} from 'react';

import {View, Animated, Button} from 'react-native';

const FadeScreen = () => {

const opacity = useRef(new Animated.Value(0)).current;

const fadeIn = () => {

Animated.timing(opacity, {

toValue: 1,

duration: 1000, // 1 segundo

useNativeDriver: true,

}).start();

};

return (

<View

style={{

flex: 1,

backgroundColor: 'grey',

justifyContent: 'center',

alignItems: 'center',

}}>

<Animated.View

style={{

backgroundColor: '#084F6A',

width: 150,

height: 150,

borderColor: 'white',

borderWidth: 10,

marginBottom: 20,

opacity: opacity,

}}

/>

<Button title="FadeIn" onPress={fadeIn} />

</View>

);

};

export default FadeScreen;

**FADE OUT**

Es lo mismo que hacer la función **fadeIn** del video anterior, pero ahora pasando la opacidad a 0. Además, para reutilizar este efecto fade vamos a crear un hook llamado **useFade.tsx**:

import {useRef} from 'react';

import {Animated} from 'react-native';

export const useFade = () => {

const opacity = useRef(new Animated.Value(0)).current;

const fadeIn = () => {

Animated.timing(opacity, {

toValue: 1,

duration: 1000, // 1 segundo

useNativeDriver: true,

}).start();

};

const fadeOut = () => {

Animated.timing(opacity, {

toValue: 0,

duration: 300, // 1 segundo

useNativeDriver: true,

}).start();

};

return {

opacity,

fadeIn,

fadeOut

}

};

**FONDO CON GRADIENTE**

Vamos a instalar **react-native-linear-gradient** con el comando:

**npm install react-native-linear-gradient**

Y luego si estamos en IOS el comando: **npx pod-install**

Ver la documentación aquí: <https://github.com/react-native-linear-gradient/react-native-linear-gradient>

Ahora nos creamos en la carpeta **components** el archivo **GradientBackground.tsx** el cual va a recibir por props las children, y utiliza el **<LinearGradient />** para darle color:

import React from 'react';

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

import LinearGradient from 'react-native-linear-gradient';

interface Props {

children: JSX.Element | JSX.Element[];

}

const GradientBackground = ({children}: Props) => {

return (

<View style={{flex: 1}}>

<LinearGradient

colors={['#084F6A', '#75CEDB', 'white']}

style={{...StyleSheet.absoluteFillObject}} //Toma toda la pantalla

start={{x: 0.1, y: 0.1}}

end={{x: 0.5, y: 0.7}}

/>

{children}

</View>

);

};

export default GradientBackground;

Por último, en la screen **HomeScreen.tsx** envolvemos todo el **return** en este componente:

return (

<GradientBackground>

<ScrollView>

...

</ScrollView>

</GradientBackground>

);

**OBTENER LOS COLORES DE LAS IMÁGENES**

Vamos a instalar un paquete para analizar los colores con el comando:

**npm install react-native-image-colors**

SI NO FUNCIONA INSTALAR LA VERSIÓN 1.5.2

Y luego si estamos en IOS ejecutar **npx pod-install**

Ver la documentación aquí: <https://github.com/osamaqarem/react-native-image-colors>

Ahora en **HomeScreen.tsx** en el **<Carousel />** necesitamos usar la propiedad **onSnapToItem** la cual al pasar la película nos da el **index** de la misma, y así ejecutar la función **getPosterColors()** que será la encargada de conseguirnos los colores mediante la importación del paquete que instalamos:

import ImageColors from 'react-native-image-colors';

...

const getPosterColors = async (index: number) => {

const movie = nowPlaying[index];

const uri = `https://image.tmdb.org/t/p/w500${movie.poster\_path}`;

const colors = await ImageColors.getColors(uri, {});

console.log(colors);

};

...

<Carousel

...

onSnapToItem={index => getPosterColors(index)}

/>

...

**HELPER: OBTENER COLORES**

Vamos a crear la carpeta **helpers** con el archivo **getColores.tsx**, en donde dependiendo si estamos en Android o IOS vamos a tener distintas propiedades de colores:

import ImageColors from 'react-native-image-colors';

export const getImageColors = async (uri: string) => {

const colors = await ImageColors.getColors(uri, {});

let primary;

let secondary;

if (colors.platform === 'android') {

primary = colors.dominant;

secondary = colors.average;

}

if (colors.platform === 'ios') {

primary = colors.primary;

secondary = colors.secondary;

}

return [primary, secondary];

};

Ahora en la función **getPosterColors** en **HomeScreen.tsx** vamos a utilizar este **helper**:

const getPosterColors = async (index: number) => {

const movie = nowPlaying[index];

const uri = `https://image.tmdb.org/t/p/w500${movie.poster\_path}`;

const [primary, secondary] = await getImageColors(uri);

console.log({primary, secondary})

};

**CREAR UN CONTEXTO PARA COMUNICAR EL CAMBIO DE COLOR**

Creamos la carpeta **context** con el archivo **GradientContext.tsx**

Este va a tener 2 states para los colores de la imagen previa y los actuales, y vamos a setear los nuevos colores con diferentes funciones:

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

interface ImageColors {

primary: string;

secondary: string;

}

interface ContextProps {

colors: ImageColors;

prevColors: ImageColors;

setMainColors: (colors: ImageColors) => void;

setPrevMainColors: (colors: ImageColors) => void;

}

export const GradientContext = createContext({} as ContextProps);

export const GradientProvider = ({children}: any) => {

const [colors, setColors] = useState<ImageColors>({

primary: 'transparent',

secondary: 'transparent',

});

const [prevColors, setPrevColors] = useState<ImageColors>({

primary: 'transparent',

secondary: 'transparent',

});

const setMainColors = (colors: ImageColors) => {

setColors(colors);

};

const setPrevMainColors = (colors: ImageColors) => {

setPrevColors(colors);

};

return (

<GradientContext.Provider

value={{

colors,

prevColors,

setMainColors,

setPrevMainColors,

}}>

{children}

</GradientContext.Provider>

);

};

Una vez creado debemos englobar toda la aplicación con este context en el **App.tsx**:

import 'react-native-gesture-handler';

import React from 'react';

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

import {Navigation} from './src/navigation/Navigation';

import {GradientProvider} from './src/context/GradientContext';

// import FadeScreen from './src/screens/FadeScreen';

const AppState = ({children}: any) => {

return <GradientProvider>{children}</GradientProvider>;

};

const App = () => {

return (

<NavigationContainer>

<AppState>

<Navigation />

</AppState>

</NavigationContainer>

);

};

export default App;

**USAR LOS COLORES DEL CONTEXT**

Ahora debemos usar este context. Primero vamos al componente **GradientBackground.tsx** y en **<LinearGradient>** en la propiedad **colors**, le pasamos los colors del **state del context**:

import React, { useContext } from 'react';

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

import LinearGradient from 'react-native-linear-gradient';

import { GradientContext } from '../context/GradientContext';

interface Props {

children: JSX.Element | JSX.Element[];

}

const GradientBackground = ({children}: Props) => {

const {colors} = useContext(GradientContext);

return (

<View style={{flex: 1}}>

<LinearGradient

colors={[colors.primary, colors.secondary, 'white']}

style={{...StyleSheet.absoluteFillObject}} //Toma toda la pantalla

start={{x: 0.1, y: 0.1}}

end={{x: 0.5, y: 0.7}}

/>

{children}

</View>

);

};

export default GradientBackground;

Y ahora en **HomeScreen.tsx** en la función **getPosterColors** debemos setear los colores:

...

const {setMainColors} = useContext(GradientContext);

const getPosterColors = async (index: number) => {

const movie = nowPlaying[index];

const uri = `https://image.tmdb.org/t/p/w500${movie.poster\_path}`;

const [primary = 'transparent', secondary = 'transparent'] = await getImageColors(uri);

setMainColors({primary, secondary})

};

...

**LÓGICA PARA CAMBIAR Y ANIMAR EL GRADIENTE**

En el **HomeScreen.tsx** vamos a implementar un **useEffect** para que cuando apenas se monte la aplicación tengamos los colores de la primera película:

...

useEffect(() => {

if (nowPlaying.length > 0) {

getPosterColors(0);

}

}, [nowPlaying]);

...

(**nowPlaying** son las películas que mostramos en el **<Carousel>**)

En el **hook useFade.tsx** debemos pasarle un callback a nuestra función de **fadeIn()** para que luego pueda ejecutar el setState de los colores previos:

import {useRef} from 'react';

import {Animated} from 'react-native';

export const useFade = () => {

const opacity = useRef(new Animated.Value(0)).current;

const fadeIn = (callback?: Function) => {

Animated.timing(opacity, {

toValue: 1,

duration: 300, // 1 segundo

useNativeDriver: true,

}).start(() => callback ? callback() : null);

};

const fadeOut = () => {

Animated.timing(opacity, {

toValue: 0,

duration: 300, // 1 segundo

useNativeDriver: true,

}).start();

};

return {

opacity,

fadeIn,

fadeOut

}

};

Ahora en el componente **GradientBackground.tsx** utilizamos el hook **useFade()**, creamos un **useEffect** que ejecute la función **fadeIn()** del hook, en donde el callback setea los colores previos con los colores actuales y luego ejecuta el **fadeOut()**.

Además para que esto funcione tenemos dos **<LinearGradient>**, uno va a tener los colores previos y el otro va a ser el que tenga los colores actuales. A uno de estos elementos le pasamos la **opacity** del hook para que se vuelva transparente al cambiar:

import React, {useContext, useEffect} from 'react';

import {Animated, StyleSheet, View} from 'react-native';

import LinearGradient from 'react-native-linear-gradient';

import {GradientContext} from '../context/GradientContext';

import {useFade} from '../hooks/useFade';

interface Props {

children: JSX.Element | JSX.Element[];

}

const GradientBackground = ({children}: Props) => {

const {colors, prevColors, setPrevMainColors} = useContext(GradientContext);

const {opacity, fadeIn, fadeOut} = useFade();

useEffect(() => {

fadeIn(() => {

setPrevMainColors(colors);

fadeOut();

});

}, [colors]);

return (

<View style={{flex: 1}}>

<LinearGradient

colors={[prevColors.primary, prevColors.secondary, 'white']}

style={{...StyleSheet.absoluteFillObject}} //Toma toda la pantalla

start={{x: 0.1, y: 0.1}}

end={{x: 0.5, y: 0.7}}

/>

<Animated.View style={{...StyleSheet.absoluteFillObject, opacity}}>

<LinearGradient

colors={[colors.primary, colors.secondary, 'white']}

style={{...StyleSheet.absoluteFillObject}} //Toma toda la pantalla

start={{x: 0.1, y: 0.1}}

end={{x: 0.5, y: 0.7}}

/>

</Animated.View>

{children}

</View>

);

};

export default GradientBackground;

**SEC-12: COMPONENTES DE REACT NATIVE**

**CREAR STACK e INSTALAR ÍCONOS**

Ver la documentación acá:

* <https://reactnavigation.org/docs/getting-started>
* <https://reactnavigation.org/docs/stack-navigator/>
* <https://github.com/oblador/react-native-vector-icons>

Y sino en este Word está bien explicado en la sección 7.

**COMPONENTE – FLATLIST**

En **HomeScreen.tsx** vamos a utilizar el **<FlatList />** el cual recibe 3 propiedades fundamentales para su funcionamiento: la **data**, **renderItem** y **keyExtractor**:

import React from 'react';

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

interface MenuItem {

name: string;

icon: string;

component: string;

}

const menuItems = [

{

name: 'Animation 01',

icon: 'cube-outline',

component: 'Animation101Screen',

},

];

const HomeScreen = () => {

const renderMenuItem = (item: MenuItem) => {

return (

<View>

<Text>

{item.name} - {item.icon}

</Text>

</View>

);

};

return (

<View style={{flex: 1}}>

<FlatList

data={menuItems} //Tiene que ser algo iterable como un array

renderItem={({item}) => renderMenuItem(item)} // el item es cada dato del array

keyExtractor={item => item.name} //necesitamos un id y que sea un string

/>

</View>

);

};

export default HomeScreen;

**FLATLIST SEPARATOR y HEADER**

Ahora vamos a utilizer las propiedades **ListHeaderComponent** y **ItemSeparatorComponent** del FlatList. Los cuales renderizan un header y un separador entre ítems:

import React from 'react';

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

import {styles} from '../theme/appTheme';

import {useSafeAreaInsets} from 'react-native-safe-area-context';

interface MenuItem {

name: string;

icon: string;

component: string;

}

const menuItems = [

{

name: 'Animation 01',

icon: 'cube-outline',

component: 'Animation101Screen',

},

{

name: 'Animation 02',

icon: 'albums-outline',

component: 'Animation102Screen',

},

];

const HomeScreen = () => {

const {top} = useSafeAreaInsets();

const renderListHeader = () => {

return (

<View style={{marginTop: top + 20, marginBottom: 20}}>

<Text style={styles.title}>Opciones de menú</Text>

</View>

);

};

const renderMenuItem = (item: MenuItem) => {

return (

<View>

<Text style={{color: 'black'}}>

{item.name} - {item.icon}

</Text>

</View>

);

};

const itemSeparator = () => {

return (

<View style={{borderBottomWidth: 1, opacity: 0.4, marginVertical: 8}} />

);

};

return (

<View style={{flex: 1, ...styles.globalMargin}}>

<FlatList

data={menuItems} //Tiene que ser algo iterable como un array

renderItem={({item}) => renderMenuItem(item)} // el item es cada dato del array

keyExtractor={item => item.name} //necesitamos un id y que sea un string

ListHeaderComponent={() => renderListHeader()} // Renderiza un header

ItemSeparatorComponent={() => itemSeparator()} // Renderiza un separador entre items

/>

</View>

);

};

export default HomeScreen;

Además creamos la carpeta **theme** con el archivo **appTheme.tsx**:

import {StyleSheet} from 'react-native';

export const styles = StyleSheet.create({

globalMargin: {

marginHorizontal: 20,

},

title: {

fontSize: 35,

fontWeight: 'bold',

color: 'black',

},

});

**FlatListItem y NAVEGACIÓN**

Para que no crezca tanto la app vamos a crearnos un componente el cual va a renderizar el **renderItem**.

**FlatListMenuItem.tsx:**

import React from 'react';

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

import {MenuItem} from '../interfaces/appInterfaces';

import Icon from 'react-native-vector-icons/Ionicons';

interface Props {

menuItem: MenuItem;

}

const FlatListMenuItem = ({menuItem}: Props) => {

return (

<TouchableOpacity activeOpacity={0.8}>

<View style={styles.container}>

<Icon name={menuItem.icon} color="grey" size={23} />

<Text style={styles.itemText}>{menuItem.name}</Text>

<View style={{flex: 1}} />

<Icon name="chevron-forward-outline" color="grey" size={23} />

</View>

</TouchableOpacity>

);

};

const styles = StyleSheet.create({

container: {

flexDirection: 'row',

},

itemText: {

marginLeft: 10,

fontSize: 19,

color: 'black',

},

});

export default FlatListMenuItem;

En **HomeScreen.tsx** pasamos este componente en el renderItem del FlatList y ya no hará falta tanto código como antes:

import React from 'react';

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

import {styles} from '../theme/appTheme';

import {useSafeAreaInsets} from 'react-native-safe-area-context';

import {MenuItem} from '../interfaces/appInterfaces';

import FlatListMenuItem from '../components/FlatListMenuItem';

const menuItems: MenuItem[] = [

{

name: 'Animation 01',

icon: 'cube-outline',

component: 'Animation101Screen',

},

{

name: 'Animation 02',

icon: 'albums-outline',

component: 'Animation102Screen',

},

];

const HomeScreen = () => {

const {top} = useSafeAreaInsets();

const renderListHeader = () => {

return (

<View style={{marginTop: top + 20, marginBottom: 20}}>

<Text style={styles.title}>Opciones de menú</Text>

</View>

);

};

const itemSeparator = () => {

return (

<View style={{borderBottomWidth: 1, opacity: 0.4, marginVertical: 8}} />

);

};

return (

<View style={{flex: 1, ...styles.globalMargin}}>

<FlatList

data={menuItems} //Tiene que ser algo iterable como un array

renderItem={({item}) => <FlatListMenuItem menuItem={item} />} // el item es cada dato del array

keyExtractor={item => item.name} //necesitamos un id y que sea un string

ListHeaderComponent={() => renderListHeader()} // Renderiza un header

ItemSeparatorComponent={() => itemSeparator()} // Renderiza un separador entre items

/>

</View>

);

};

export default HomeScreen;

Además creamos la carpeta **interfaces** con el archivo **appInterfaces.tsx** la cual va a tener la interface del **MenuItem**:

export interface MenuItem {

name: string;

icon: string;

component: string;

}

**ANIMATED API**

Creamos la screen **Animation101Screen.tsx** la cual contendrá un View que será una caja violeta al que le aplicaremos el **Animated**:

import React, {useRef} from 'react';

import {View, StyleSheet, Animated} from 'react-native';

export const Animation101Screen = () => {

const opacity = useRef(new Animated.Value(0)).current;

return (

<View style={styles.container}>

<Animated.View style={{...styles.purpleBox, opacity}} />

</View>

);

};

const styles = StyleSheet.create({

container: {

flex: 1,

justifyContent: 'center',

alignItems: 'center',

},

purpleBox: {

backgroundColor: '#5856D6',

width: 150,

height: 150,

},

});

Luego debemos colocar esta Screen en **Navigator.tsx:**

...

export const Navigator = () => {

return (

<Stack.Navigator screenOptions={{headerShown: false}}>

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

<Stack.Screen name="Animation101Screen" component={Animation101Screen} />

</Stack.Navigator>

);

};

Y por último en el componente **FlatListMenuItem.tsx** debemos usar el hook **useNavigation()** para navegar a esta pantalla:

...

const FlatListMenuItem = ({menuItem}: Props) => {

const navigation = useNavigation<any>();

return (

<TouchableOpacity

activeOpacity={0.8}

onPress={() => navigation.navigate(menuItem.component)}>

...

**FADE IN y FADE OUT**

En nuestro **Animation101Screen.tsx** vamos a crear dos funciones una **fadeIn** y otra **fadeOut**, en donde vamos a utilizar el **Animated.timing()** con la finalidad de hacer el efecto:

import React, {useRef} from 'react';

import {View, StyleSheet, Animated, Button} from 'react-native';

export const Animation101Screen = () => {

const opacity = useRef(new Animated.Value(0)).current;

const fadeIn = () => {

Animated.timing(opacity, {

toValue: 1,

duration: 300,

useNativeDriver: true,

}).start();

};

const fadeOut = () => {

Animated.timing(opacity, {

toValue: 0,

duration: 300,

useNativeDriver: true,

}).start();

};

return (

<View style={styles.container}>

<Animated.View style={{...styles.purpleBox, opacity, marginBottom: 20}} />

<Button title="Fade In" onPress={fadeIn} />

<Button title="Fade Out" onPress={fadeOut} />

</View>

);

};

const styles = StyleSheet.create({

container: {

flex: 1,

justifyContent: 'center',

alignItems: 'center',

},

purpleBox: {

backgroundColor: '#5856D6',

width: 150,

height: 150,

},

});

**EASING – BOUNCE**

En **Animation101Screen.tsx** vamos a agregar otro **Animated.timing()** en la función **fadeIn** con la finalidad de que la caja violeta al aparecer, lo haga desde arriba y con u rebote. Por lo que vamos a usar la propiedades **easing** para eso.

Además debemos agregarle a **<Animated.View>** en sus estilos el **transform**:

import React, {useRef} from 'react';

import {View, StyleSheet, Animated, Button, Easing} from 'react-native';

export const Animation101Screen = () => {

const opacity = useRef(new Animated.Value(0)).current;

const top = useRef(new Animated.Value(-100)).current;

const fadeIn = () => {

Animated.timing(opacity, {

toValue: 1,

duration: 300,

useNativeDriver: true,

}).start();

Animated.timing(top, {

toValue: 0,

duration: 500,

useNativeDriver: true,

easing: Easing.bounce,

}).start();

};

const fadeOut = () => {

Animated.timing(opacity, {

toValue: 0,

duration: 300,

useNativeDriver: true,

}).start();

};

return (

<View style={styles.container}>

<Animated.View

style={{

...styles.purpleBox,

opacity,

marginBottom: 20,

transform: [{translateY: top}],

}}

/>

<Button title="Fade In" onPress={fadeIn} />

<Button title="Fade Out" onPress={fadeOut} />

</View>

);

};

const styles = StyleSheet.create({

container: {

flex: 1,

justifyContent: 'center',

alignItems: 'center',

},

purpleBox: {

backgroundColor: '#5856D6',

width: 150,

height: 150,

},

});

**USE ANIMATION**

Ahora para que el código en **Animation101Screen.tsx** no se haga tan largo, vamos a crear en la carpeta **hooks** el archivo **useAnimation.tsx** el cual contendrá toda la lógica del fadeIn, fadeOut, y podremos pasarle los valores de inicialización y así hacer a nuestro código reutilizable:

import {useRef} from 'react';

import {Animated, Easing} from 'react-native';

export const useAnimation = () => {

const opacity = useRef(new Animated.Value(0)).current;

const position = useRef(new Animated.Value(0)).current;

const fadeIn = () => {

Animated.timing(opacity, {

toValue: 1,

duration: 300,

useNativeDriver: true,

}).start();

};

const fadeOut = () => {

Animated.timing(opacity, {

toValue: 0,

duration: 300,

useNativeDriver: true,

}).start();

};

const startMovingPosition = (

initPosition: number,

duration: number = 300,

) => {

position.setValue(initPosition);

Animated.timing(position, {

toValue: 0,

duration: duration,

useNativeDriver: true,

//easing: Easing.bounce,

}).start();

};

return {

opacity,

position,

fadeIn,

fadeOut,

startMovingPosition,

};

};

Luego en **Animation101Screen.tsx** importamos este hook:

import React from 'react';

import {View, StyleSheet, Animated, Button, Easing} from 'react-native';

import {useAnimation} from '../hooks/useAnimation';

export const Animation101Screen = () => {

const {opacity, position, fadeIn, fadeOut, startMovingPosition} =

useAnimation();

return (

<View style={styles.container}>

<Animated.View

style={{

...styles.purpleBox,

opacity,

marginBottom: 20,

transform: [{translateY: position}],

}}

/>

<Button

title="Fade In"

onPress={() => {

fadeIn();

startMovingPosition(-100);

}}

/>

<Button title="Fade Out" onPress={fadeOut} />

</View>

);

};

const styles = StyleSheet.create({

container: {

flex: 1,

justifyContent: 'center',

alignItems: 'center',

},

purpleBox: {

backgroundColor: '#5856D6',

width: 150,

height: 150,

},

});

**ANIMATED VALUE XY**

Vamos a crear la animación que esta acá:

<https://reactnative.dev/docs/animatedvaluexy>

Para ello, vamos a crear la screen **Animation102Screen.tsx**:

import React, {useRef} from 'react';

import {View, StyleSheet, Animated, PanResponder} from 'react-native';

export const Animation102Screen = () => {

const pan = useRef(new Animated.ValueXY()).current;

const panResponder = PanResponder.create({

onStartShouldSetPanResponder: () => true,

onPanResponderMove: Animated.event(

[

null,

{

dx: pan.x, // x,y are Animated.Value

dy: pan.y,

},

],

{useNativeDriver: false},

),

onPanResponderRelease: () => {

Animated.spring(

pan, // Auto-multiplexed

{

toValue: {x: 0, y: 0},

useNativeDriver: false

}, // Back to zero

).start();

},

});

return (

<View style={styles.container}>

<Animated.View

{...panResponder.panHandlers}

style={[pan.getLayout(), styles.box]}

/>

</View>

);

};

const styles = StyleSheet.create({

container: {

flex: 1,

justifyContent: 'center',

alignItems: 'center',

},

box: {

backgroundColor: '#75CEDB',

width: 150,

height: 150,

},

});

**COMPONENTE – SWITCH**

Creamos una nueva screen llamada **SwitchScreen.tsx** en donde importamos el componente **<Switch />** de react native y lo configuramos:

import React, {useState} from 'react';

import {View, Switch, Platform} from 'react-native';

export const SwitchScreen = () => {

const [isEnabled, setIsEnabled] = useState(false);

const toggleSwitch = () => setIsEnabled(!isEnabled);

return (

<View style={{marginTop: 100}}>

<Switch

trackColor={{false: '#D9D9DB', true: '#5856D6'}}

thumbColor={Platform.OS === 'android' ? '#5856D6' : ''}

// ios\_backgroundColor="#3e3e3e"

onValueChange={toggleSwitch}

value={isEnabled}

/>

</View>

);

};

Luego lo importamos en **Navigator.tsx** para que se pueda navegar hacia él:

import React from 'react';

import {createStackNavigator} from '@react-navigation/stack';

import HomeScreen from '../screens/HomeScreen';

import {Animation101Screen} from '../screens/Animation101Screen';

import {Animation102Screen} from '../screens/Animation102Screen';

import {SwitchScreen} from '../screens/SwitchScreen';

const Stack = createStackNavigator();

export const Navigator = () => {

return (

<Stack.Navigator screenOptions={{headerShown: false}}>

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

<Stack.Screen name="Animation101Screen" component={Animation101Screen} />

<Stack.Screen name="Animation102Screen" component={Animation102Screen} />

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

</Stack.Navigator>

);

};

A continuación en **HomeScreen.tsx** teníamos la data de los componentes a mostrar, pero como esto empieza a crecer es mejor crearse una carpeta **data** con el archivo **menuItems.tsx** e importar esta data en el **HomeScreen**:

**menuItems.tsx:**

import {MenuItem} from '../interfaces/appInterfaces';

export const menuItems: MenuItem[] = [

{

name: 'Animation 01',

icon: 'cube-outline',

component: 'Animation101Screen',

},

{

name: 'Animation 02',

icon: 'albums-outline',

component: 'Animation102Screen',

},

{

name: 'Switches',

icon: 'toggle-outline',

component: 'SwitchScreen',

},

];

**HEADER RE-UTILIZABLE**

Vamos a crear en la carpeta **components** el archivo **HeaderTitle.tsx** con la finalidad de reutilizar los títulos. Vamos a copiar el código que teníamos en el **HomeScreen.tsx** y pegarlo en este componente:

import React from 'react';

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

import {styles} from '../theme/appTheme';

import {useSafeAreaInsets} from 'react-native-safe-area-context';

interface Props {

title: string;

}

export const HeaderTitle = ({title}: Props) => {

const {top} = useSafeAreaInsets();

return (

<View style={{marginTop: top + 20, marginBottom: 20}}>

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

</View>

);

};

Ahora lo usamos en **HomeScreen.tsx** y notemos que cada vez queda más chico nuestra screen:

import React from 'react';

import {View, FlatList} from 'react-native';

import {styles} from '../theme/appTheme';

import FlatListMenuItem from '../components/FlatListMenuItem';

import {menuItems} from '../data/menuItems';

import {HeaderTitle} from '../components/HeaderTitle';

const HomeScreen = () => {

const itemSeparator = () => {

return (

<View style={{borderBottomWidth: 1, opacity: 0.4, marginVertical: 8}} />

);

};

return (

<View style={{flex: 1, ...styles.globalMargin}}>

<FlatList

data={menuItems} //Tiene que ser algo iterable como un array

renderItem={({item}) => <FlatListMenuItem menuItem={item} />} // el item es cada dato del array

keyExtractor={item => item.name} //necesitamos un id y que sea un string

ListHeaderComponent={() => <HeaderTitle title="Opciones de menú" />} // Renderiza un header

ItemSeparatorComponent={() => itemSeparator()} // Renderiza un separador entre items

/>

</View>

);

};

export default HomeScreen;

Y también lo utilizamos en **SwitchScreen.tsx**:

import React, {useState} from 'react';

import {View, Switch, Platform} from 'react-native';

import {HeaderTitle} from '../components/HeaderTitle';

export const SwitchScreen = () => {

const [isEnabled, setIsEnabled] = useState(false);

const toggleSwitch = () => setIsEnabled(!isEnabled);

return (

<View style={{marginHorizontal: 20}}>

<HeaderTitle title="Switches" />

<Switch

trackColor={{false: '#D9D9DB', true: '#5856D6'}}

thumbColor={Platform.OS === 'android' ? '#5856D6' : ''}

// ios\_backgroundColor="#3e3e3e"

onValueChange={toggleSwitch}

value={isEnabled}

/>

</View>

);

};

**CUSTOM SWITCH**

Creamos en la carpeta **components** el archivo **CustomSwitch.tsx** para tener un Switch reutilizable. Va a contener dos propiedades por Props, la primera es un valor booleano para saber cómo va a estar el switch cuando se monte la aplicación y el otro es una función onChange que va a cambiar el state del componente donde estemos usando este customSwitch:

import React, {useState} from 'react';

import {Switch, Platform} from 'react-native';

interface Props {

isOn: boolean;

onChange: (value: boolean) => void;

}

export const CustomSwitch = ({isOn, onChange}: Props) => {

const [isEnabled, setIsEnabled] = useState(isOn);

const toggleSwitch = () => {

setIsEnabled(!isEnabled);

onChange(!isEnabled);

};

return (

<Switch

trackColor={{false: '#D9D9DB', true: '#5856D6'}}

thumbColor={Platform.OS === 'android' ? '#5856D6' : ''}

// ios\_backgroundColor="#3e3e3e"

onValueChange={toggleSwitch}

value={isEnabled}

/>

);

};

Y ahora en **SwitchScreen.tsx** utilizamos este customSwitch:

import React, {useState} from 'react';

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

import {HeaderTitle} from '../components/HeaderTitle';

import {CustomSwitch} from '../components/CustomSwitch';

export const SwitchScreen = () => {

const [state, setState] = useState({

isActive: true,

isHungry: false,

isHappy: true,

});

const {isActive, isHungry, isHappy} = state;

const onChange = (value: boolean, field: keyof typeof state) => {

setState({

...state,

[field]: value,

});

};

return (

<View style={{marginHorizontal: 20}}>

<HeaderTitle title="Switches" />

<View style={styles.switchRow}>

<Text style={styles.switchText}>isActive</Text>

<CustomSwitch

isOn={isActive}

onChange={value => onChange(value, 'isActive')}

/>

</View>

<View style={styles.switchRow}>

<Text style={styles.switchText}>isHungry</Text>

<CustomSwitch

isOn={isHungry}

onChange={value => onChange(value, 'isHungry')}

/>

</View>

<View style={styles.switchRow}>

<Text style={styles.switchText}>isHappy</Text>

<CustomSwitch

isOn={isHappy}

onChange={value => onChange(value, 'isHappy')}

/>

</View>

<Text style={styles.switchText}>{JSON.stringify(state, null, 5)}</Text>

</View>

);

};

const styles = StyleSheet.create({

switchText: {

fontSize: 25,

color: 'black',

},

switchRow: {

flexDirection: 'row',

justifyContent: 'space-between',

alignItems: 'center',

marginVertical: 10

},

});

**COMPONENTE – ALERT**

Creamos la screen **AlertScreen.tsx** en donde usamos el **Alert** de react native:

import React from 'react';

import {View, Button, Alert} from 'react-native';

import {HeaderTitle} from '../components/HeaderTitle';

import {styles} from '../theme/appTheme';

const showAlert = () => {

Alert.alert(

'Título de la alerta',

'Mensaje de la alerta',

[

{

text: 'Cancel',

onPress: () => console.log('Cancel Pressed'),

style: 'destructive',

},

{text: 'OK', onPress: () => console.log('OK Pressed')},

],

{

cancelable: true, //en android apretas por fuera de la alerta y se cierra

onDismiss: () => console.log('onDismiss'),

},

);

};

export const AlertScreen = () => {

return (

<View style={styles.globalMargin}>

<HeaderTitle title="Alerts" />

<Button title="Mostrar alerta" onPress={showAlert} />

</View>

);

};

Luego en **Navigator.tsx** no olvidarse de poner esta página para poder navegar hacia ella:

import React from 'react';

import {createStackNavigator} from '@react-navigation/stack';

...

import {AlertScreen} from '../screens/AlertScreen';

const Stack = createStackNavigator();

export const Navigator = () => {

return (

<Stack.Navigator screenOptions={{headerShown: false}}>

...

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

</Stack.Navigator>

);

};

Y en el archivo **data/menuItems.tsx** crear la configuración:

import {MenuItem} from '../interfaces/appInterfaces';

export const menuItems: MenuItem[] = [

...

{

name: 'Alerts',

icon: 'alert-circle-outline',

component: 'AlertScreen',

},

];

**COMPONENTE – ALERT PROMPT**

Este componente solo funciona en IOS.

**AlertScreen.tsx**:

import React from 'react';

import {View, Button, Alert} from 'react-native';

import {HeaderTitle} from '../components/HeaderTitle';

import {styles} from '../theme/appTheme';

...

const showPrompt = () => {

Alert.prompt(

'Esta seguro?',

'Esta acción no se puede revertir',

(value: string) => console.log('Info: ', value),

'plain-text', //tipo de texto donde se escribe

'', // defaultValue

'email-address', // podes especificar el teclado que aparecerá

);

};

export const AlertScreen = () => {

return (

<View style={styles.globalMargin}>

<HeaderTitle title="Alerts" />

<Button title="Mostrar alerta" onPress={showAlert} />

<Button title="Mostrar prompt" onPress={showPrompt} />

</View>

);

};

**PROMPT IOS y ANDROID**

Vamos a descargar el paquete: **npm i react-native-prompt-android**

Cabe aclarar que no hace falta instalarlo porque podríamos usar un modal en Android y listo.

Ver la documentación aquí: <https://www.npmjs.com/package/react-native-prompt-android>

**AlertScreen.tsx:**

import React from 'react';

import {View, Button, Alert} from 'react-native';

import prompt from 'react-native-prompt-android';

...

const showPrompt = () => {

prompt(

'Enter password',

'Enter your password to claim your $1.5B in lottery winnings',

[

{

text: 'Cancel',

onPress: () => console.log('Cancel Pressed'),

style: 'cancel',

},

{

text: 'OK',

onPress: password => console.log('OK Pressed, password: ' + password),

},

],

{

type: 'secure-text',

cancelable: false,

defaultValue: 'test',

placeholder: 'placeholder',

},

);

};

export const AlertScreen = () => {

return (

<View style={styles.globalMargin}>

<HeaderTitle title="Alerts" />

<Button title="Mostrar alerta" onPress={showAlert} />

<View style={{height: 10}} />

<Button title="Mostrar prompt" onPress={showPrompt} />

</View>

);

};

**COMPONENTE – TEXT INPUT**

Creamos la screen **TextInputScreen.tsx** con algunos estilos nada más por ahora:

import React from 'react';

import {StyleSheet, TextInput, View} from 'react-native';

import {styles} from '../theme/appTheme';

import {HeaderTitle} from '../components/HeaderTitle';

export const TextInputScreen = () => {

return (

<View style={styles.globalMargin}>

<HeaderTitle title="TextInputs" />

<TextInput style={stylesTI.inputStyle} />

</View>

);

};

const stylesTI = StyleSheet.create({

inputStyle: {

borderWidth: 1,

borderColor: 'rgba(0,0,0,0.3)',

height: 50,

paddingHorizontal: 10,

borderRadius: 10,

},

});

**Navigator.tsx:**

import React from 'react';

import {createStackNavigator} from '@react-navigation/stack';

...

import { TextInputScreen } from '../screens/TextInputScreen';

const Stack = createStackNavigator();

export const Navigator = () => {

return (

<Stack.Navigator screenOptions={{headerShown: false}}>

...

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

</Stack.Navigator>

);

};

**menuItems.tsx:**

import {MenuItem} from '../interfaces/appInterfaces';

export const menuItems: MenuItem[] = [

...

{

name: 'TextInputs',

icon: 'document-text-outline',

component: 'TextInputScreen',

},

];

**MÚLTIPLES TEXT INPUTS EN PANTALLA**

Ahora configuramos los **<TextInput />** en la screen **TextInputScreen.tsx**:

import React, {useState} from 'react';

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

import {styles} from '../theme/appTheme';

import {HeaderTitle} from '../components/HeaderTitle';

export const TextInputScreen = () => {

const [form, setForm] = useState({

name: '',

email: '',

phone: '',

});

const onChange = (value: string, field: string) => {

setForm({...form, [field]: value});

};

return (

<View style={styles.globalMargin}>

<HeaderTitle title="TextInputs" />

<TextInput

style={stylesTI.inputStyle}

placeholder="Ingrese su nombre"

autoCorrect={false} //no te corrige lo que escribas

autoCapitalize="words" //capitaliza las palabras

onChangeText={value => onChange(value, 'name')}

/>

<TextInput

style={stylesTI.inputStyle}

placeholder="Ingrese su email"

autoCorrect={false}

autoCapitalize="none" //no capitaliza las palabras

onChangeText={value => onChange(value, 'email')}

keyboardType="email-address" //le dice al teclado que debe aparecer el @

/>

<TextInput

style={stylesTI.inputStyle}

placeholder="Ingrese su teléfono"

onChangeText={value => onChange(value, 'phone')}

keyboardType="phone-pad"

/>

<Text style={stylesTI.stateView}>{JSON.stringify(form, null, 5)}</Text>

</View>

);

};

const stylesTI = StyleSheet.create({

inputStyle: {

borderWidth: 1,

borderColor: 'rgba(0,0,0,0.3)',

height: 50,

paddingHorizontal: 10,

borderRadius: 10,

marginVertical: 10,

},

stateView: {

fontSize: 28,

color: 'black',

fontWeight: 'bold',

},

});

**COMPONENTE – KEYBOARDAVOIDINGVIEW**

Ahora si tenemos varios inputs puede pasar que al abrirse el teclado nos tape la vista de ese input, por eso debemos envolver todo el JSX en las etiquetas **<KeyboardAvoidingView />**, **<ScrollView />** y **<TouchableWithoutFeedback />** y así no permitir este comportamiento. Además agregamos un **<View />** al final con una altura determinada para tener más espacio.

**TextInputScreen.tsx:**

...

return (

<KeyboardAvoidingView

behavior={Platform.OS === 'ios' ? 'padding' : 'height'}

>

<ScrollView>

<TouchableWithoutFeedback onPress={Keyboard.dismiss}>

<View style={styles.globalMargin}>

...

<View style={{height: 100}} />

</View>

</TouchableWithoutFeedback>

</ScrollView>

</KeyboardAvoidingView>

...

**USEFORM – TAREA**

En la carpeta **hooks** vamos a crear el archivo **useForm.tsx**, el cual va a manejar el state de los inputs:

import { useState } from 'react'

export const useForm = <T extends Object>(initState: T) => {

const [state, setState] = useState(initState);

const onChange = (value: string, field: string) => {

setState({...state, [field]: value});

};

return {

...state,

form: state,

onChange

}

}

Y ahora lo utilizamos en **TextInputScreen.tsx**:

...

import {useForm} from '../hooks/useForm';

export const TextInputScreen = () => {

const {form, onChange} = useForm({

name: '',

email: '',

phone: '',

});

...

**PULL TO REFRESH**

Creamos la screen **PullToRefreshScreen.tsx**, en la cual debemos envolver todo el JSX en un **<ScrollView />** y pasarle como propiedad en el **refreshControl** el componente **<RefreshControl />** de react native:

import React, {useState} from 'react';

import {View, ScrollView, RefreshControl} from 'react-native';

import {HeaderTitle} from '../components/HeaderTitle';

import {styles} from '../theme/appTheme';

export const PullToRefreshScreen = () => {

const [refreshing, setRefreshing] = useState(false);

const onRefresh = () => {

setRefreshing(true);

setTimeout(() => {

console.log('Terminamos el refresh');

setRefreshing(false);

}, 1500);

};

return (

<ScrollView

refreshControl={

<RefreshControl refreshing={refreshing} onRefresh={onRefresh} />

}>

<View style={styles.globalMargin}>

<HeaderTitle title="PullToRefresh" />

</View>

</ScrollView>

);

};

**Navigator.tsx:**

import React from 'react';

import {createStackNavigator} from '@react-navigation/stack';

...

import {PullToRefreshScreen} from '../screens/PullToRefreshScreen';

const Stack = createStackNavigator();

export const Navigator = () => {

return (

<Stack.Navigator

screenOptions={{

headerShown: false,

cardStyle: {backgroundColor: 'white'},

}}>

...

<Stack.Screen

name="PullToRefreshScreen"

component={PullToRefreshScreen}

/>

</Stack.Navigator>

);

};

**menuItems.tsx:**

import {MenuItem} from '../interfaces/appInterfaces';

export const menuItems: MenuItem[] = [

...

{

name: 'PullToRefresh',

icon: 'refresh-outline',

component: 'PullToRefreshScreen',

},

];

**PERSONALIZANDO EL REFRESH CONTROL**

Ahora en **PullToRefreshScreen.tsx** vamos a utilizar un useState para que cuando termine de cargar muestre una data, y además muestro todas las opciones que se le pueden pasar al componente **<RefreshControl />** para editar su style:

import React, {useState} from 'react';

import {View, ScrollView, RefreshControl, Platform, Text} from 'react-native';

import {HeaderTitle} from '../components/HeaderTitle';

import {styles} from '../theme/appTheme';

export const PullToRefreshScreen = () => {

const [refreshing, setRefreshing] = useState(false);

const [data, setData] = useState<string>();

const onRefresh = () => {

setRefreshing(true);

setTimeout(() => {

console.log('Terminamos el refresh');

setRefreshing(false);

setData('Hola mundo');

}, 1500);

};

return (

<ScrollView

refreshControl={

<RefreshControl

refreshing={refreshing}

onRefresh={onRefresh}

progressViewOffset={Platform.OS === 'ios' ? 50 : 0} //baja el logo de cargando

// progressBackgroundColor="#5856D6" //le cambia el color al background del signo de cargando, solo para android

// colors={['purple', 'red', 'orange']} //cambia el color del signo a medida que pasa el tiempo, solo para android

// style={{backgroundColor: '#5856D6'}} //cambia todo el backgroundcolor del espacio que esta utilizando el logo, solo ios CREO

// tintColor='orange' //cambia el color del signo, solo para IOS

// title='REFRESHING' //le da un titulo abajo del signo, solo IOS

// titleColor='orange' //le da color al titulo debajo del signo, solo IOS

/>

}>

<View style={styles.globalMargin}>

<HeaderTitle title="PullToRefresh" />

<Text style={{fontSize: 28, fontWeight: 'bold', color: 'black'}}>

{data}

</Text>

</View>

</ScrollView>

);

};

**COMPONENTE – SECTION LIST**

Creamos la screen **CustomSectionListScreen.tsx** en donde usamos el componente **<SectionList />**:

import React from 'react';

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

import {HeaderTitle} from '../components/HeaderTitle';

import {styles} from '../theme/appTheme';

import {ItemSeparator} from '../components/ItemSeparator';

interface Casas {

casa: string;

data: string[];

}

const casas: Casas[] = [

{

casa: 'DC Comics',

data: ['Batman', 'Superman', 'Robin', 'Flash', 'Aquaman', 'Wonder Woman'],

},

{

casa: 'Marvel Comics',

data: [

'Antman',

'Thor',

'Spiderman',

'Doctor Strange',

'Hulk',

'Iron Man',

'Black Panther',

],

},

{

casa: 'Anime',

data: [

'Naruto',

'Dragon Ball Z',

'My Hero Academia',

'Evangelion',

'Attack of Titan',

],

},

];

export const CustomSectionListScreen = () => {

return (

<View style={{...styles.globalMargin, flex: 1}}>

<SectionList

sections={casas}

keyExtractor={(item, i) => item + i} //identificador único

renderItem={({item}) => <Text style={{fontSize: 20}}>{item}</Text>} //renderiza el item

ListHeaderComponent={() => <HeaderTitle title="Header List" />} //Renderiza un Header

ListFooterComponent={() => (

<View style={{marginBottom: 50}}>

<HeaderTitle title="Footer List" />

</View>

)} //Renderiza un Footer

stickySectionHeadersEnabled //le da un efecto bonito

renderSectionHeader={({section}) => (

<View style={{backgroundColor: 'white', marginVertical: 10}}>

<Text style={{fontSize: 26, color: 'black', fontWeight: 'bold'}}>

{section.casa}

</Text>

</View>

)} //renderiza el titulo del item

renderSectionFooter={({section}) => (

<View style={{backgroundColor: 'white', marginVertical: 5}}>

<Text style={{fontSize: 22, color: 'black', fontWeight: 'bold'}}>

{`Total: ${section.data.length}`}

</Text>

</View>

)} //renderiza el footer del item

SectionSeparatorComponent={() => <ItemSeparator />} //renderiza un separador por debajo de los header del item

ItemSeparatorComponent={() => <ItemSeparator />} //renderiza un separador por debajo de los ítems

showsVerticalScrollIndicator={false} //elimina la barra de scroll

/>

</View>

);

};

Creamos en la carpeta **components** el **ItemSeparator.tsx** con la finalidad de tener un componente reutilizable de un separador:

import React from 'react';

import {View} from 'react-native';

export const ItemSeparator = () => {

return (

<View style={{borderBottomWidth: 1, opacity: 0.4, marginVertical: 8}} />

);

};

Y lo utilizamos en el **HomeScreen.tsx** también:

const HomeScreen = () => {

return (

<View style={{flex: 1, ...styles.globalMargin}}>

<FlatList

...

ItemSeparatorComponent={() => <ItemSeparator />} // Renderiza un separador entre items

/>

</View>

);

};

No olvidarse del **Navigator.tsx** y el **menuItems.tsx**:

import React from 'react';

import {createStackNavigator} from '@react-navigation/stack';

...

import { CustomSectionListScreen } from '../screens/CustomSectionListScreen';

const Stack = createStackNavigator();

export const Navigator = () => {

return (

<Stack.Navigator

screenOptions={{

headerShown: false,

cardStyle: {backgroundColor: 'white'},

}}>

...

<Stack.Screen

name="CustomSectionListScreen"

component={CustomSectionListScreen}

/>

</Stack.Navigator>

);

};

import {MenuItem} from '../interfaces/appInterfaces';

export const menuItems: MenuItem[] = [

...

{

name: 'Section List',

icon: 'list-outline',

component: 'CustomSectionListScreen',

},

];

**COMPONENTE - MODAL**

Creamos la screen **ModalScreen.tsx** donde utilizamos el componente **<Modal />**:

import React, {useState} from 'react';

import {View, Button, Modal, Text, StyleSheet} from 'react-native';

import {styles} from '../theme/appTheme';

import {HeaderTitle} from '../components/HeaderTitle';

export const ModalScreen = () => {

const [isVisible, setisVisible] = useState(false);

return (

<View style={styles.globalMargin}>

<HeaderTitle title="Modal Screen" />

<Button title="Abrir Modal" onPress={() => setisVisible(true)} />

<Modal

animationType="fade" //como se abre el modal, puede ser 'slide' tambien

visible={isVisible} //valor boolean que abre o cierra el modal

transparent={true} //se ve lo del fondo cuando se abre el modal

>

<View style={modalStyles.modalContainer}>

<View style={modalStyles.modalBoxStyle}>

<Text style={modalStyles.modalTitle}>Titulo</Text>

<Text style={modalStyles.modalBody}>Cuerpo</Text>

<Button title="Cerrar" onPress={() => setisVisible(false)} />

</View>

</View>

</Modal>

</View>

);

};

const modalStyles = StyleSheet.create({

modalContainer: {

flex: 1,

backgroundColor: 'rgba(0,0,0,0.3)',

justifyContent: 'center',

alignItems: 'center',

},

modalBoxStyle: {

backgroundColor: 'white',

width: 200,

height: 200,

justifyContent: 'center',

alignItems: 'center',

shadowOffset: {

width: 0,

height: 10,

},

shadowOpacity: 0.25,

elevation: 10,

borderRadius: 5,

},

modalTitle: {

fontSize: 20,

fontWeight: 'bold',

marginBottom: 10,

color: 'black',

},

modalBody: {

fontSize: 16,

marginBottom: 20,

},

});

**INFINITE SCROLL**

Creamos la screen **InfiniteScrollScreen.tsx** donde creamos un **<FlatList />** con las propiedades **onEndReached** y **onEndReachedThreshold**, para especificarle cuando va a empezar a cargar nueva data, y hacerlo infinito:

import React, {useState} from 'react';

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

import {styles} from '../theme/appTheme';

import {HeaderTitle} from '../components/HeaderTitle';

export const InfiniteScrollScreen = () => {

const [numbers, setNumbers] = useState([0, 1, 2, 3, 4, 5]);

const loadMore = () => {

const newArray: number[] = [];

for (let i = 0; i < 5; i++) {

newArray[i] = numbers.length + i;

}

setNumbers([...numbers, ...newArray]);

};

const renderItem = (item: number) => {

return <Text style={infiteStyles.textItem}>{item}</Text>;

};

return (

<View style={{flex: 1}}>

<FlatList

data={numbers}

keyExtractor={item => item.toString()}

renderItem={({item}) => renderItem(item)}

ListHeaderComponent={<HeaderTitle title="Infinite Scroll" />}

onEndReached={loadMore} //ejecuta la función cuando llega a la parte de la especificada en el onEndReachedThreshold

onEndReachedThreshold={0.5} //a partir de cuando va a cargar la nueva informacion, 0.5 mitad de la pantalla

/>

</View>

);

};

const infiteStyles = StyleSheet.create({

textItem: {

height: 150,

fontSize: 30,

},

});

No olvidarse de crear la navegación en **Navigator.tsx** y la data en **menuItems.tsx**:

import React from 'react';

import {createStackNavigator} from '@react-navigation/stack';

...

import { InfiniteScrollScreen } from '../screens/InfiniteScrollScreen';

const Stack = createStackNavigator();

export const Navigator = () => {

return (

<Stack.Navigator

screenOptions={{

headerShown: false,

cardStyle: {backgroundColor: 'white'},

}}>

...

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

</Stack.Navigator>

);

};

import {MenuItem} from '../interfaces/appInterfaces';

export const menuItems: MenuItem[] = [

...

{

name: 'Infinite Scroll',

icon: 'download-outline',

component: 'InfiniteScrollScreen',

},

];

**INFINITE SCROLL CON IMÁGENES**

Vamos a utilizar esta documentación para cargar imágenes de manera infinita:

<https://picsum.photos/>

Ahora en la función **renderItem** que tenemos en **InfiniteScrollScreen.tsx** en vez de devolver un **<Text />**, vamos a devolver un **<Image />** con la url dinámica de la documentación que está arriba.

Además agregamos un **setTimeOut** para que las cargue al 1.5s y mostramos un **loading** mientras eso pasa:

import React, {useState} from 'react';

import {

ActivityIndicator,

FlatList,

Image,

StyleSheet,

Text,View,

} from 'react-native';

import {styles} from '../theme/appTheme';

import {HeaderTitle} from '../components/HeaderTitle';

export const InfiniteScrollScreen = () => {

const [numbers, setNumbers] = useState([0, 1, 2, 3, 4, 5]);

const loadMore = () => {

const newArray: number[] = [];

for (let i = 0; i < 5; i++) {

newArray[i] = numbers.length + i;

}

setTimeout(() => {

setNumbers([...numbers, ...newArray]);

}, 1500);

};

const renderItem = (item: number) => {

return (

<Image

source={{uri: `https://picsum.photos/id/${item}/500/400`}}

style={infiteStyles.itemImage}

/>

);

};

return (

<View style={{flex: 1}}>

<FlatList

data={numbers}

keyExtractor={item => item.toString()}

renderItem={({item}) => renderItem(item)}

ListHeaderComponent={<HeaderTitle title="Infinite Scroll" />}

onEndReached={loadMore} //ejecuta la función cuando llega a la parte de la especificada en el onEndReachedThreshold

onEndReachedThreshold={0.5} //a partir de cuando va a cargar la nueva informacion, 0.5 mitad de la pantalla

ListFooterComponent={() => (

<View style={infiteStyles.loading}>

<ActivityIndicator size={25} color="#5856D6" />

</View>

)}

/>

</View>

);

};

const infiteStyles = StyleSheet.create({

itemImage: {

width: '100%',

height: 400,

},

loading: {

width: '100%',

height: 150,

justifyContent: 'center',

alignItems: 'center',

},

});

**ANIMATED IMAGE**

Nos vamos a crear el componente **FadeInImage.tsx** para hacer que las imágenes infinitas entren con un efecto de fadeIn y también que tengan un loading hasta que se carguen. Para eso, vamos a usar nuestro hook **useAnimation.tsx** que posee el efecto de fadeIn. También la estamos diciendo por **props** que puede recibir los estilos de la foto:

import React, {useState} from 'react';

import {useAnimation} from '../hooks/useAnimation';

import {

ActivityIndicator,

Animated,

ImageStyle,

StyleProp,

View,

} from 'react-native';

interface Props {

uri: string;

style?: StyleProp<ImageStyle>;

}

export const FadeInImage = ({uri, style}: Props) => {

const {opacity, fadeIn} = useAnimation();

const [isLoading, setIsLoading] = useState(true);

const finishLoading = () => {

setIsLoading(false);

fadeIn();

};

return (

<View style={{justifyContent: 'center', alignItems: 'center'}}>

{isLoading && (

<ActivityIndicator

style={{position: 'absolute'}}

color="#5856D6"

size={30}

/>

)}

<Animated.Image

source={{uri}}

onLoadEnd={finishLoading}

style={{

...(style as any),

opacity,

}}

/>

</View>

);

};

Luego lo aplicamos en **InfiniteScrollScreen.tsx** en la función de **renderItem**:

...

const renderItem = (item: number) => {

return (

<FadeInImage

style={{width: '100%', height: 400}}

uri={`https://picsum.photos/id/${item}/500/400`}

/>

);

};

...

**SEC-13: TEMAS y SLIDESHOW de PANTALLA COMPLETA**

**CONTINUACIÓN DEL PROYECTO – COMPONENTS APP**

Continuamos trabajando con el proyecto anterior.

Creamos en la carpeta **screens** el archivo **SlidesScreen.tsx**:

import React from 'react';

import {View} from 'react-native';

import {HeaderTitle} from '../components/HeaderTitle';

export const SlidesScreen = () => {

return (

<View>

<HeaderTitle title="Slides" />

</View>

);

};

Necesitamos poder navegar a ella en **Navigator.tsx**:

import React from 'react';

import {createStackNavigator} from '@react-navigation/stack';

...

import {SlidesScreen} from '../screens/SlidesScreen';

const Stack = createStackNavigator();

export const Navigator = () => {

return (

<Stack.Navigator

screenOptions={{

headerShown: false,

cardStyle: {backgroundColor: 'white'},

}}>

...

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

</Stack.Navigator>

);

};

Y poner la data en **menuItems.tsx**:

import {MenuItem} from '../interfaces/appInterfaces';

export const menuItems: MenuItem[] = [

...

{

name: 'Slides',

icon: 'flower-outline',

component: 'SlidesScreen',

},

];

**SNAP CAROUSEL**

Vamos a la documentación en: <https://github.com/dohooo/react-native-reanimated-carousel>

Y corremos los comandos:

**npm install react-native-reanimated-carousel**

(Hay que tener instalado react-native-reanimated: leer la documentación)

Y no olvidarse del comando **npx pod-install** para ios.

Luego en **SlidesScreen.tsx** utilizamos este carousel:

import React from 'react';

import {

Dimensions,

Image,

ImageSourcePropType,

SafeAreaView,

StyleSheet,

Text,

View,

} from 'react-native';

import Carousel from 'react-native-reanimated-carousel';

const {height: screenHeight, width: screenWidth} = Dimensions.get('window');

interface Slide {

title: string;

desc: string;

img: ImageSourcePropType;

}

const items: Slide[] = [

{

title: 'Titulo 1',

desc: 'Ea et eu enim fugiat sunt reprehenderit sunt aute quis tempor ipsum cupidatat et.',

img: require('../assets/slide-1.png'),

},

{

title: 'Titulo 2',

desc: 'Anim est quis elit proident magna quis cupidatat culpa labore Lorem ea. Exercitation mollit velit in aliquip tempor occaecat dolor minim amet dolor enim cillum excepteur. ',

img: require('../assets/slide-2.png'),

},

{

title: 'Titulo 3',

desc: 'Ex amet duis amet nulla. Aliquip ea Lorem ea culpa consequat proident. Nulla tempor esse ad tempor sit amet Lorem. Velit ea labore aute pariatur commodo duis veniam enim.',

img: require('../assets/slide-3.png'),

},

];

export const SlidesScreen = () => {

const renderItem = (item: Slide) => {

return (

<View style={styles.imageContainer}>

<Image

source={item.img}

style={{width: 350, height: 400, resizeMode: 'center'}}

/>

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

<Text style={styles.imageDescription}>{item.desc}</Text>

</View>

);

};

return (

<SafeAreaView style={{flex: 1, paddingTop: 50}}>

<Carousel

mode="parallax"

style={{width: screenWidth}}

pagingEnabled={false}

windowSize={2}

snapEnabled

width={screenWidth}

height={screenHeight}

modeConfig={{

parallaxScrollingScale: 0.9,

parallaxScrollingOffset: 40,

parallaxAdjacentItemScale: 0.75,

}}

data={items}

renderItem={({item}) => renderItem(item)}

/>

</SafeAreaView>

);

};

const styles = StyleSheet.create({

imageContainer: {

flex: 1,

backgroundColor: 'white',

borderRadius: 5,

padding: 40,

justifyContent: 'center',

},

imageTitle: {

fontSize: 30,

fontWeight: 'bold',

color: '#5856D6',

},

imageDescription: {

fontSize: 16,

color: 'black',

},

});

**PAGINACIÓN DE SLIDES – (PUNTOS INDICATIVOS)**

Agregamos debajo del **<Carousel/>** un map que recorra todos los ítems y por cada item haga un dot:

const [activeIndex, setActiveIndex] = useState(0);

...

<Carousel

...

data={items}

renderItem={({item}) => renderItem(item)}

onSnapToItem={i => {

setActiveIndex(i);

}}

defaultIndex={activeIndex}

/>

<View

style={{

flexDirection: 'row',

justifyContent: 'center',

alignItems: 'center',

height: 30,

}}>

{items.map((\_, i) => {

const isActive = i === activeIndex;

return (

<TouchableOpacity key={i} onPress={() => setActiveIndex(i)}>

<View

style={{

backgroundColor: isActive ? '#5856D6' : '#5856d684',

width: 10,

height: 10,

borderRadius: 10,

}}

/>

</TouchableOpacity>

);

})}

</View>

...

**BOTÓN EN EL ÚLTIMO SLIDE**

Para hacer esto vamos a agregar un **<Animated.View />** junto a un **<TouchableOpacity/>**, donde utilizaremos el hook **useAnimation()** para la opacidad y el fadeIn. Además creamos un **useRef** que nos va a servir de condiconal para permitirnos navegar hacia el HomeScreen:

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

import {

Animated,

Dimensions,

Image,

ImageSourcePropType,

SafeAreaView,

StyleSheet,

Text,

TouchableOpacity,

View,

} from 'react-native';

import Carousel from 'react-native-reanimated-carousel';

import Icon from 'react-native-vector-icons/Ionicons';

import {useAnimation} from '../hooks/useAnimation';

import {StackScreenProps} from '@react-navigation/stack';

const {width: screenWidth} = Dimensions.get('window');

interface Slide {

title: string;

desc: string;

img: ImageSourcePropType;

}

const items: Slide[] = [

{

title: 'Titulo 1',

desc: 'Ea et eu enim fugiat sunt reprehenderit sunt aute quis tempor ipsum cupidatat et.',

img: require('../assets/slide-1.png'),

},

{

title: 'Titulo 2',

desc: 'Anim est quis elit proident magna quis cupidatat culpa labore Lorem ea. Exercitation mollit velit in aliquip tempor occaecat dolor minim amet dolor enim cillum excepteur. ',

img: require('../assets/slide-2.png'),

},

{

title: 'Titulo 3',

desc: 'Ex amet duis amet nulla. Aliquip ea Lorem ea culpa consequat proident. Nulla tempor esse ad tempor sit amet Lorem. Velit ea labore aute pariatur commodo duis veniam enim.',

img: require('../assets/slide-3.png'),

},

];

interface Props extends StackScreenProps<any, any> {}

export const SlidesScreen = ({navigation}: Props) => {

const [activeIndex, setActiveIndex] = useState(0);

const isVisible = useRef(false);

const {opacity, fadeIn} = useAnimation();

const renderItem = (item: Slide) => {

return (

<View style={styles.imageContainer}>

<Image

source={item.img}

style={{width: 350, height: 400, resizeMode: 'center'}}

/>

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

<Text style={styles.imageDescription}>{item.desc}</Text>

</View>

);

};

return (

<SafeAreaView style={{flex: 1, paddingTop: 50}}>

<Carousel

mode="parallax"

style={{width: screenWidth}}

pagingEnabled={false}

windowSize={2}

snapEnabled

width={screenWidth}

modeConfig={{

parallaxScrollingScale: 0.9,

parallaxScrollingOffset: 40,

parallaxAdjacentItemScale: 0.75,

}}

data={items}

renderItem={({item}) => renderItem(item)}

onSnapToItem={i => {

setActiveIndex(i);

if (i === items.length - 1) {

isVisible.current = true;

fadeIn();

}

}}

defaultIndex={activeIndex}

/>

<View style={styles.dotsContainer}>

{items.map((\_, i) => {

const isActive = i === activeIndex;

return (

<TouchableOpacity key={i} onPress={() => setActiveIndex(i)}>

<View

style={{

backgroundColor: isActive ? '#5856D6' : '#5856d684',

width: 10,

height: 10,

borderRadius: 10,

}}

/>

</TouchableOpacity>

);

})}

<Animated.View style={{opacity}}>

<TouchableOpacity

style={styles.enterButton}

activeOpacity={0.8}

onPress={() => {

if (isVisible.current) {

navigation.navigate('HomeScreen');

}

}}>

<Text style={{fontSize: 25, color: 'white'}}>Entrar</Text>

<Icon name="chevron-forward-outline" color="white" size={30} />

</TouchableOpacity>

</Animated.View>

</View>

</SafeAreaView>

);

};

const styles = StyleSheet.create({

imageContainer: {

flex: 1,

backgroundColor: 'white',

borderRadius: 5,

padding: 40,

justifyContent: 'center',

},

imageTitle: {

fontSize: 30,

fontWeight: 'bold',

color: '#5856D6',

},

imageDescription: {

fontSize: 16,

color: 'black',

},

dotsContainer: {

flexDirection: 'row',

justifyContent: 'space-evenly',

alignItems: 'center',

top: -100,

},

enterButton: {

flexDirection: 'row',

justifyContent: 'center',

alignItems: 'center',

backgroundColor: '#5856D6',

width: 140,

height: 50,

borderRadius: 10,

},

});

**THEME LIKE y DARK – GENERALIDADES**

Vamos a la documentación: <https://reactnavigation.org/docs/themes/>

Para realizar esto, debemos ir al **App.tsx** e importar **DefaultTheme**, **DarkTheme** y **Theme** (este último es el tipado), y pasarle al **<NavigationContainer />** la prop **theme**:

import 'react-native-gesture-handler';

import React from 'react';

import {

NavigationContainer,

DefaultTheme,

DarkTheme,

Theme,

} from '@react-navigation/native';

import {Navigator} from './src/navigation/Navigator';

const customTheme: Theme = {

dark: true,

colors: {

...DarkTheme.colors,

// primary: '',

// background: '',

// card: '',

// text: '',

// border: '',

// notification: '',

},

};

const App = () => {

return (

<NavigationContainer theme={customTheme}>

<Navigator />

</NavigationContainer>

);

};

export default App;

Ahora para que nos aparezca el fondo negro porque seleccionamos el darkTheme, necesitamos ir al **Navigator.tsx** y comentar la línea del **cardStyle**:

...

return (

<Stack.Navigator

screenOptions={{

headerShown: false,

// cardStyle: {backgroundColor: 'white'},

}}>

...

**PANTALLA PARA CAMBIAR EL TEMA DE LA APP**

Vamos a crear la screen **ChangeThemeScreen.tsx**, la cual va a tener solo un botón para cambiar el tema de light a dark o viceversa:

import React from 'react';

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

import {styles} from '../theme/appTheme';

import {HeaderTitle} from '../components/HeaderTitle';

export const ChangeThemeScreen = () => {

return (

<View style={styles.globalMargin}>

<HeaderTitle title="Change Theme" />

<TouchableOpacity

activeOpacity={0.8}

style={{

backgroundColor: '#5856D6',

justifyContent: 'center',

width: 150,

height: 50,

borderRadius: 20,

}}>

<Text style={{color: 'white', textAlign: 'center', fontSize: 22}}>

Light / Dark

</Text>

</TouchableOpacity>

</View>

);

};

**Navigator.tsx:**

import React from 'react';

import {createStackNavigator} from '@react-navigation/stack';

...

import { ChangeThemeScreen } from '../screens/ChangeThemeScreen';

const Stack = createStackNavigator();

export const Navigator = () => {

return (

<Stack.Navigator

screenOptions={{

headerShown: false,

// cardStyle: {backgroundColor: 'white'},

}}>

...

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

</Stack.Navigator>

);

};

**menuItems.tsx:**

import {MenuItem} from '../interfaces/appInterfaces';

export const menuItems: MenuItem[] = [

...

{

name: 'Change Theme',

icon: 'flask-outline',

component: 'ChangeThemeScreen',

},

];

**THEME CONTEXT**

Creamos la carpeta **context** con el archivo **ThemeContext.tsx**, el cual devolverá dos funciones, una para poner el tema light y la otra para poner el tema dark:

import React, {createContext} from 'react';

interface ThemeContextProps {

theme: any; //toDo

setDarkTheme: () => void;

setLightTheme: () => void;

}

export const ThemeContext = createContext({} as ThemeContextProps);

export const ThemeProvider = ({children}: any) => {

const theme = {};

const setDarkTheme = () => {};

const setLightTheme = () => {};

return (

<ThemeContext.Provider value={{setDarkTheme, setLightTheme, theme}}>

{children}

</ThemeContext.Provider>

);

};

Ahora en el **App.tsx** debemos envolver toda nuestra aplicación en este Provider, y además vamos a sacar el **<NavigationContainer />** y lo colocaremos en el archivo **Navigator.tsx**

**App.tsx:**

import 'react-native-gesture-handler';

import React from 'react';

import {Navigator} from './src/navigation/Navigator';

import {ThemeProvider} from './src/context/ThemeContext';

const App = () => {

return (

<AppState>

<Navigator />

</AppState>

);

};

const AppState = ({children}: any) => {

return <ThemeProvider>{children}</ThemeProvider>;

};

export default App;

import React from 'react';

import {createStackNavigator} from '@react-navigation/stack';

...

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

const Stack = createStackNavigator();

export const Navigator = () => {

return (

<NavigationContainer>

<Stack.Navigator

screenOptions={{

headerShown: false,

// cardStyle: {backgroundColor: 'white'},

}}>

...

</Stack.Navigator>

</NavigationContainer>

);

};

Y recién ahora podemos utilizar el context en **ChangeThemeScreen.tsx**:

import React, {useContext} from 'react';

...

import {ThemeContext} from '../context/ThemeContext';

export const ChangeThemeScreen = () => {

const {setDarkTheme, setLightTheme} = useContext(ThemeContext);

...

**THEME REDUCER**

Dentro de la carpeta **context** vamos a crear el **themeReducer.tsx**:

import {Theme} from '@react-navigation/native';

type ThemeAction = {type: 'set\_light\_theme'} | {type: 'set\_dark\_theme'};

export interface ThemeState extends Theme {

currentTheme: 'light' | 'dark';

dividerColor: string;

}

export const lightTheme: ThemeState = {

currentTheme: 'light',

dark: false,

dividerColor: 'rgba(0, 0, 0, 0.7)',

colors: {

primary: '#5856D6',

background: 'white',

card: '',

text: 'black',

border: '',

notification: '',

},

};

export const themeReducer = (

state: ThemeState,

action: ThemeAction,

): ThemeState => {

switch (action.type) {

case 'set\_light\_theme':

return {...lightTheme};

default:

return state;

}

};

**CONSUMIR EL THEME CONTEXT**

En el **ThemeContext.tsx**, en las funciones podemos ejecutar el dispatch y también ya tenemos el tipado del theme:

import React, {createContext, useReducer} from 'react';

import {ThemeState, lightTheme, themeReducer} from './themeReducer';

interface ThemeContextProps {

theme: ThemeState;

setDarkTheme: () => void;

setLightTheme: () => void;

}

export const ThemeContext = createContext({} as ThemeContextProps);

export const ThemeProvider = ({children}: any) => {

const [theme, dispatch] = useReducer(themeReducer, lightTheme);

const setDarkTheme = () => {

dispatch({type: 'set\_dark\_theme'});

};

const setLightTheme = () => {

dispatch({type: 'set\_light\_theme'});

};

return (

<ThemeContext.Provider value={{setDarkTheme, setLightTheme, theme}}>

{children}

</ThemeContext.Provider>

);

};

En el **Navigator.tsx** podemos pasarle el **theme** al **<NavigationContainer />**:

import React, {useContext} from 'react';

import {createStackNavigator} from '@react-navigation/stack';

import {ThemeContext} from '../context/ThemeContext';

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

...

const Stack = createStackNavigator();

export const Navigator = () => {

const {theme} = useContext(ThemeContext);

return (

<NavigationContainer theme={theme}>

...

Y ahora por ejemplo en el **FlatListMenuItem.tsx** podemos usar el **theme**:

import React, { useContext } from 'react';

import { ThemeContext } from '../context/ThemeContext';

...

const {theme} = useContext(ThemeContext);

return (

<TouchableOpacity

activeOpacity={0.8}

onPress={() => navigation.navigate(menuItem.component)}>

<View style={styles.container}>

<Icon name={menuItem.icon} color={theme.colors.primary} size={23} />

<Text style={styles.itemText}>{menuItem.name}</Text>

<View style={{flex: 1}} />

<Icon name="chevron-forward-outline" color={theme.colors.primary} size={23} />

</View>

</TouchableOpacity>

);

};

**CAMBIAR ENTRE DARK Y LIGHT**

Ahora en el **themeReducer.tsx** debemos crear en el **switch** el caso para el darkTheme:

import {Theme} from '@react-navigation/native';

type ThemeAction = {type: 'set\_light\_theme'} | {type: 'set\_dark\_theme'};

export interface ThemeState extends Theme {

currentTheme: 'light' | 'dark';

dividerColor: string;

}

export const lightTheme: ThemeState = {

currentTheme: 'light',

dark: false,

dividerColor: 'rgba(0, 0, 0, 0.7)',

colors: {

primary: '#5856D6',

background: 'white',

card: '',

text: 'black',

border: '',

notification: '',

},

};

export const darkTheme: ThemeState = {

currentTheme: 'dark',

dark: true,

dividerColor: 'rgba(0, 0, 0, 0.7)',

colors: {

primary: '#5856D6',

background: 'black',

card: '',

text: 'white',

border: '',

notification: '',

},

};

export const themeReducer = (

state: ThemeState,

action: ThemeAction,

): ThemeState => {

switch (action.type) {

case 'set\_light\_theme':

return {...lightTheme};

case 'set\_dark\_theme':

return {...darkTheme};

default:

return state;

}

};

Y ahora con esto en el **ChangeThemeScreen.tsx** utilizamos las dos funciones para que el theme cambie:

import React, {useContext} from 'react';

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

import {styles} from '../theme/appTheme';

import {HeaderTitle} from '../components/HeaderTitle';

import {ThemeContext} from '../context/ThemeContext';

export const ChangeThemeScreen = () => {

const {setDarkTheme, setLightTheme, theme} = useContext(ThemeContext);

return (

<View style={styles.globalMargin}>

<HeaderTitle title="Change Theme" />

<View style={{flexDirection: 'row', justifyContent: 'space-evenly'}}>

<TouchableOpacity

onPress={setLightTheme}

activeOpacity={0.8}

style={{

backgroundColor: theme.colors.primary,

justifyContent: 'center',

width: 150,

height: 50,

borderRadius: 20,

}}>

<Text style={{color: 'white', textAlign: 'center', fontSize: 22}}>

Light

</Text>

</TouchableOpacity>

<TouchableOpacity

onPress={setDarkTheme}

activeOpacity={0.8}

style={{

backgroundColor: theme.colors.primary,

justifyContent: 'center',

width: 150,

height: 50,

borderRadius: 20,

}}>

<Text style={{color: 'white', textAlign: 'center', fontSize: 22}}>

Dark

</Text>

</TouchableOpacity>

</View>

</View>

);

};

**CAMBIO DE TEMA BASADO EN EL SISTEMA OPERATIVO**

Para eso vamos a usar **AppState** y **Appearance** de react native, y dependiendo que tema tengamos en configuraciones del dispositivo se va a modificar nuestro tema de aplicación.

También se puede hacer con **useColorScheme**, pero no funciona tan bien en Android.

**ThemeContext.tsx:**

import React, {createContext, useEffect, useReducer} from 'react';

import {ThemeState, darkTheme, lightTheme, themeReducer} from './themeReducer';

import {AppState, Appearance, useColorScheme} from 'react-native';

interface ThemeContextProps {

theme: ThemeState;

setDarkTheme: () => void;

setLightTheme: () => void;

}

export const ThemeContext = createContext({} as ThemeContextProps);

// const colorScheme = useColorScheme();

export const ThemeProvider = ({children}: any) => {

const [theme, dispatch] = useReducer(

themeReducer,

Appearance.getColorScheme() === 'dark' ? darkTheme : lightTheme,

);

useEffect(() => {

AppState.addEventListener('change', status => {

if (status === 'active') {

Appearance.getColorScheme() === 'light'

? setLightTheme()

: setDarkTheme();

}

});

}, []);

// SOLO EN IOS

// useEffect(() => {

// if (colorScheme === 'light') {

// setLightTheme();

// }

// if (colorScheme === 'dark') {

// setDarkTheme();

// }

// }, [colorScheme]);

const setDarkTheme = () => {

dispatch({type: 'set\_dark\_theme'});

};

const setLightTheme = () => {

dispatch({type: 'set\_light\_theme'});

};

return (

<ThemeContext.Provider value={{setDarkTheme, setLightTheme, theme}}>

{children}

</ThemeContext.Provider>

);

};

**SEC-14: POKEDEX**

**STACK NAVIGATOR e ÍCONOS**

Creamos el Proyecto **Pokedex**, junto con las carpetas **screens**, **components**, **hooks**, **api**, **navigation** y del material adjunto la carpeta **assets** que contiene 2 imágenes.

Luego instalamos **react navigation** desde acá: <https://reactnavigation.org/docs/stack-navigator/>

Y también los íconos de: <https://github.com/oblador/react-native-vector-icons>

(Ambos tutoriales están en este Word, buscar ‘íconos en nuestra aplicación’)

A continuación debemos crear dentro de la carpeta **navigation** el archivo **Navigator.tsx** el cual va a ser nuestra navegación a las distintas screens:

import React from 'react';

import {createStackNavigator} from '@react-navigation/stack';

import {HomeScreen} from '../screens/HomeScreen';

import {PokemonScreen} from '../screens/PokemonScreen';

const Stack = createStackNavigator();

export const Navigator = () => {

return (

<Stack.Navigator>

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

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

</Stack.Navigator>

);

};

Y envolver este **Navigator** en **<NavigationContainer/>** dentro de nuesto **App.tsx**:

import 'react-native-gesture-handler';

import React from 'react';

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

import {Navigator} from './src/navigation/Navigator';

const App = () => {

return (

<NavigationContainer>

<Navigator />

</NavigationContainer>

);

};

export default App;

**DISEÑO BÁSICO DEL HOMESCREEN**

En **HomeScreen.tsx** vamos a colocar una imagen y un título de la pokedex:

import React from 'react';

import {Text, Image} from 'react-native';

import {styles} from '../theme/appTheme';

import {useSafeAreaInsets} from 'react-native-safe-area-context';

export const HomeScreen = () => {

const {top} = useSafeAreaInsets();

return (

<>

<Image

source={require('../assets/pokebola.png')}

style={styles.pokebolaBG}

/>

<Text

style={{

...styles.title,

...styles.globalMargin,

top: top + 20,

color: 'black',

}}>

Pokedex

</Text>

</>

);

};

Además nos creamos unos estilos globales, para eso creamos la carpeta **theme** con el archivo **appTheme.tsx**:

import {StyleSheet} from 'react-native';

export const styles = StyleSheet.create({

globalMargin: {

marginHorizontal: 20,

},

pokebolaBG: {

position: 'absolute',

width: 300,

height: 300,

top: -100,

right: -100,

opacity: 0.2,

},

title: {

fontSize: 35,

fontWeight: 'bold',

},

});

**OBTENER LISTA DE POKEMONS EN UN CUSTOM HOOK**

En la carpeta **api** creamos el archivo **pokemonApi.tsx**, la cual solo crea la instancia de axios para las peticiones que vamos a realizar:

import axios from 'axios';

export const pokemonApi = axios.create();

Luego en la carpeta **hooks** creamos el archive **usePokemonPaginated.tsx**, en el cual se hace la petición de axios para obtener los pokemones:

import {useEffect, useRef} from 'react';

import {pokemonApi} from '../api/pokemonApi';

export const usePokemonPaginated = () => {

const nextPageUrl = useRef('https://pokeapi.co/api/v2/pokemon?limit=40');

useEffect(() => {

loadPokemons();

}, []);

const loadPokemons = async () => {

const resp = await pokemonApi.get(nextPageUrl.current);

console.log(resp.data);

};

return {};

};

Y ahora en el **HomeScreen.tsx** podemos llamar a este hook:

...

import {usePokemonPaginated} from '../hooks/usePokemonPaginated';

export const HomeScreen = () => {

const {top} = useSafeAreaInsets();

usePokemonPaginated();

...

**COLOCAR EL TIPO DE DATO QUE ROTORNA LA PETICIÓN HTTP**

Creamos la carpeta **interfaces** con el archivo **pokemonInterfaces.tsx** y con la ayuda de <https://app.quicktype.io/> pegamos la respuesta de postman de la petición http:

export interface PokemonPaginatedResponse {

count: number;

next: string;

previous: null;

results: Result[];

}

export interface Result {

name: string;

url: string;

}

export interface SimplePokemon {

id: string;

name: string;

picture: string;

color?: string;

}

Y ahora que tenemos el tipado, podemos utilizarlo en nuestro hook **usePokemonPaginated.tsx**:

import {useEffect, useRef, useState} from 'react';

import {pokemonApi} from '../api/pokemonApi';

import {

PokemonPaginatedResponse,

Result,

SimplePokemon,

} from '../interfaces/pokemonInterfaces';

export const usePokemonPaginated = () => {

const [simplePokemonList, setSimplePokemonList] = useState<SimplePokemon[]>(

[],

);

const nextPageUrl = useRef('https://pokeapi.co/api/v2/pokemon?limit=40');

useEffect(() => {

loadPokemons();

}, []);

const loadPokemons = async () => {

const resp = await pokemonApi.get<PokemonPaginatedResponse>(

nextPageUrl.current,

);

nextPageUrl.current = resp.data.next;

mapPokemonList(resp.data.results);

};

const mapPokemonList = (pokemonList: Result[]) => {

pokemonList.forEach(poke => console.log(poke.name));

};

return {

simplePokemonList

};

};

**MAPEAR RESULTADO A UN ARREGLO DE SIMPLE POKEMONS**

En el hook **usePokemonPaginated.tsx** vamos a setear un array de pokemones con el id, el nombre y la imagen.

import {useEffect, useRef, useState} from 'react';

import {pokemonApi} from '../api/pokemonApi';

import {

PokemonPaginatedResponse,

Result,

SimplePokemon,

} from '../interfaces/pokemonInterfaces';

export const usePokemonPaginated = () => {

const [simplePokemonList, setSimplePokemonList] = useState<SimplePokemon[]>([]);

const [isLoading, setIsLoading] = useState(true);

const nextPageUrl = useRef('https://pokeapi.co/api/v2/pokemon?limit=40');

useEffect(() => {

loadPokemons();

}, []);

const loadPokemons = async () => {

setIsLoading(true);

const resp = await pokemonApi.get<PokemonPaginatedResponse>(nextPageUrl.current);

nextPageUrl.current = resp.data.next;

mapPokemonList(resp.data.results);

};

const mapPokemonList = (pokemonList: Result[]) => {

const newPokemonList: SimplePokemon[] = pokemonList.map(({name, url}) => {

const urlParts = url.split('/');

const id = urlParts[urlParts.length - 2];

const picture = `https://raw.githubusercontent.com/PokeAPI/sprites/master/sprites/pokemon/other/official-artwork/${id}.png`;

return {

id,

picture,

name,

};

});

setSimplePokemonList([...simplePokemonList, ...newPokemonList]);

setIsLoading(false);

};

return {

simplePokemonList,

isLoading,

};

};

**MOSTRAR POKEMONS – INFINITE SCROLL**

En **HomeScreen.tsx** ahora tenemos todo lo necesario para poder implementar un **<FlatList />** y mostrar la data de los pokemons. Cuando llegamos al final de la pantalla el **onEndReached** ejecuta la función del hook que carga los nuevos pokemons:

import React from 'react';

import {Text, Image, FlatList, ActivityIndicator} from 'react-native';

import {styles} from '../theme/appTheme';

import {useSafeAreaInsets} from 'react-native-safe-area-context';

import {usePokemonPaginated} from '../hooks/usePokemonPaginated';

export const HomeScreen = () => {

...

const {simplePokemonList, loadPokemons} = usePokemonPaginated();

return (

<>

...

<FlatList

data={simplePokemonList}

keyExtractor={pokemon => pokemon.id}

showsVerticalScrollIndicator={false}

renderItem={({item}) => (

<Image

source={{uri: item.picture}}

style={{width: 100, height: 100}}

/>

)}

//infinite scroll

onEndReached={loadPokemons}

onEndReachedThreshold={0.4}

ListFooterComponent={

<ActivityIndicator style={{height: 100}} size={20} color="grey" />

}

/>

...

</>

);

};

**FADE IN IMAGE y USE ANIMATION**

En la carpeta **hooks** creamos el archivo **useAnimation.tsx**, el cual maneja el efecto de fadeIn:

import {useRef} from 'react';

import {Animated} from 'react-native';

export const useAnimation = () => {

const opacity = useRef(new Animated.Value(0)).current;

const position = useRef(new Animated.Value(0)).current;

const fadeIn = (duration: number = 300) => {

Animated.timing(opacity, {

toValue: 1,

duration,

useNativeDriver: true,

}).start();

};

const fadeOut = () => {

Animated.timing(opacity, {

toValue: 0,

duration: 300,

useNativeDriver: true,

}).start();

};

const startMovingPosition = (

initPosition: number,

duration: number = 300,

) => {

position.setValue(initPosition);

Animated.timing(position, {

toValue: 0,

duration,

useNativeDriver: true,

// easing: Easing.bounce

}).start();

};

return {

opacity,

position,

fadeIn,

fadeOut,

startMovingPosition,

};

};

Y en la carpeta **components** creamos el archivo **FadeInImage.tsx** el cual muestra un cargador hasta que la imagen aparece con el efecto:

import React, {useState} from 'react';

import {

ActivityIndicator,

Animated,

ImageErrorEventData,

ImageStyle,

NativeSyntheticEvent,

StyleProp,

View,

} from 'react-native';

import {useAnimation} from '../hooks/useAnimation';

interface Props {

uri: string;

style?: StyleProp<ImageStyle>;

}

export const FadeInImage = ({uri, style = {}}: Props) => {

const {opacity, fadeIn} = useAnimation();

const [isLoading, setIsLoading] = useState(true);

const finishLoading = () => {

setIsLoading(false);

fadeIn();

};

const onError = (err: NativeSyntheticEvent<ImageErrorEventData>) => {

setIsLoading(false);

};

return (

<View

style={{

justifyContent: 'center',

alignItems: 'center',

...(style as any),

}}>

{isLoading && (

<ActivityIndicator

style={{position: 'absolute'}}

color="grey"

size={30}

/>

)}

<Animated.Image

source={{uri}}

onError={onError}

onLoad={finishLoading}

style={{

...(style as any),

opacity,

}}

/>

</View>

);

};

Por último, en el **renderItem** del **<FlatList />** en el **HomeScreen.tsx** utilizamos este component:

...

<FlatList

data={simplePokemonList}

keyExtractor={pokemon => pokemon.id}

showsVerticalScrollIndicator={false}

renderItem={({item}) => (

<FadeInImage

uri={item.picture}

style={{width: 100, height: 100}}

/>

)}

...

/>

...

**POKEMONCARD – COMPONENTE**

En la carpeta **components** creamos el archivo **PokemonCard.tsx**, el cual será renderizado en el **renderItem** del FlatList, y contendrá toda la información de un Pokemon:

import React from 'react';

import {

View,

Text,

TouchableOpacity,

StyleSheet,

Dimensions,

Image,

} from 'react-native';

import {SimplePokemon} from '../interfaces/pokemonInterfaces';

import {FadeInImage} from './FadeInImage';

const windowWidth = Dimensions.get('window').width;

interface Props {

pokemon: SimplePokemon;

}

export const PokemonCard = ({pokemon}: Props) => {

return (

<TouchableOpacity activeOpacity={0.8}>

<View style={styles.cardContainer}>

<View>

<Text style={styles.name}>

{pokemon.name}

{'\n#' + pokemon.id}

</Text>

</View>

<View style={styles.pokebolaContainer}>

<Image

source={require('../assets/pokebola-blanca.png')}

style={styles.pokebola}

/>

</View>

<FadeInImage uri={pokemon.picture} style={styles.pokemonImage} />

</View>

</TouchableOpacity>

);

};

const styles = StyleSheet.create({

cardContainer: {

backgroundColor: 'white',

marginHorizontal: 10,

height: 120,

width: windowWidth \* 0.4,

marginBottom: 25,

borderRadius: 10,

shadowColor: '#000',

shadowOffset: {

width: 0,

height: 2,

},

shadowOpacity: 0.25,

shadowRadius: 3.84,

elevation: 5,

},

name: {

// color: 'white',

fontSize: 20,

fontWeight: 'bold',

top: 20,

left: 10,

},

pokebolaContainer: {

width: 100,

height: 100,

position: 'absolute',

bottom: 0,

right: 0,

overflow: 'hidden',

opacity: 0.5,

},

pokebola: {

width: 100,

height: 100,

position: 'absolute',

right: -25,

bottom: -25,

},

pokemonImage: {

width: 120,

height: 120,

position: 'absolute',

right: -8,

bottom: -5,

},

});

**HomeScreen.tsx:**

...

<View style={{alignItems: 'center'}}>

<FlatList

data={simplePokemonList}

keyExtractor={pokemon => pokemon.id}

showsVerticalScrollIndicator={false}

numColumns={2}

renderItem={({item}) => <PokemonCard pokemon={item} />}

//infinite scroll

onEndReached={loadPokemons}

onEndReachedThreshold={0.4}

//Header y Footer

ListHeaderComponent={

<Text

style={{

...styles.title,

...styles.globalMargin,

top: top + 20,

marginBottom: top + 20,

paddingBottom: 10,

color: 'black',

}}>

Pokedex

</Text>

}

ListFooterComponent={

<ActivityIndicator style={{height: 100}} size={20} color="grey" />

}

/>

</View>

...

**COLOCAR EL COLOR DE FONDO DE LA TARJETA BASADO EN EL POKEMON**

Vamos a instalar un paquete para analizar los colores de la imagen:

**npm install react-native-image-colors**

SI NO FUNCIONA INSTALAR LA VERSIÓN 1.5.2

Y luego si estamos en IOS ejecutar **npx pod-install**

Ver la documentación aquí: <https://github.com/osamaqarem/react-native-image-colors>

Ahora lo que hacemos es un stateque contenga el color del background, y luego en un useEffect ejecutamos la función del **ImageColors** para setearlo con el background correspondiente a cada imagen. Y luego se lo pasamos a los estilos del View.

**PokemonCard.tsx:**

...

import ImageColors from 'react-native-image-colors';

...

export const PokemonCard = ({pokemon}: Props) => {

const [bgColor, setbgColor] = useState('white');

useEffect(() => {

ImageColors.getColors(pokemon.picture, {fallback: 'white'}).then(colors => {

if (colors.platform === 'android') {

setbgColor(colors.dominant || 'white');

}

if (colors.platform === 'ios') {

setbgColor(colors.background || 'white');

}

});

}, []);

return (

<TouchableOpacity activeOpacity={0.8}>

<View style={{...styles.cardContainer, backgroundColor: bgColor}}>

...

**PROBLEMA CON LA ACTUALIZACIÓN DE ESTADO**

Para que no suceda un error de que no esté montado el componente aún antes de conseguir el color de fondo de la imagen, debemos hacer una validación extra. Vamos a utilizar un **useRef** que en el return del useEffect haga el desmontamiento.

**PokemonCard.tsx:**

export const PokemonCard = ({pokemon}: Props) => {

const [bgColor, setbgColor] = useState('white');

const isMounted = useRef(true);

useEffect(() => {

ImageColors.getColors(pokemon.picture, {fallback: 'white'}).then(colors => {

if (!isMounted.current) return;

if (colors.platform === 'android') {

setbgColor(colors.dominant || 'white');

}

if (colors.platform === 'ios') {

setbgColor(colors.background || 'white');

}

});

return () => {

isMounted.current = false;

};

}, []);

**NAVEGAR A LA PANTALLA DE DETALLE**

Para realizar esto debemos especificarle que props vamos a recibir en la screen **PokemonScreen**, para eso vamos al **Navigator.tsx** y creamos el **RootStackParamas**:

import React from 'react';

import {createStackNavigator} from '@react-navigation/stack';

import { HomeScreen } from '../screens/HomeScreen';

import { PokemonScreen } from '../screens/PokemonScreen';

import { SimplePokemon } from '../interfaces/pokemonInterfaces';

export type RootStackParams = {

HomeScreen: undefined;

PokemonScreen: {simplePokemon: SimplePokemon, color: string}

}

const Stack = createStackNavigator<RootStackParams>();

export const Navigator = () => {

return (

<Stack.Navigator

screenOptions={{

headerShown: false,

cardStyle: {backgroundColor: 'white'},

}}>

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

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

</Stack.Navigator>

);

};

Ahora en el **<TouchableOpacity/>** del componente **PokemonCard.tsx** hacemos la navegación hacia el detalle de cada pokemon:

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

...

const [bgColor, setbgColor] = useState('white');

...

const navigation = useNavigation<any>();

...

return (

<TouchableOpacity

activeOpacity={0.8}

onPress={() => {

navigation.navigate('PokemonScreen', {

simplePokemon: pokemon,

color: bgColor,

});

}}>

...

Y por último en el **PokemonScreen.tsx** creamos la interface Props que extienda de **StackScreenProps** para recibir las props que especificamos en el **Navigator.tsx**:

import React from 'react';

import {StackScreenProps} from '@react-navigation/stack';

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

import {RootStackParams} from '../navigation/Navigator';

interface Props extends StackScreenProps<RootStackParams, 'PokemonScreen'> {}

export const PokemonScreen = ({navigation, route}: Props) => {

const {simplePokemon, color} = route.params;

return (

<View>

<Text>{simplePokemon.name}</Text>

</View>

);

};

**PANTALLA DEL POKEMON**

En **PokemonScreen.tsx** empezamos a crear el header de la pantalla con la información del pokemon:

import React from 'react';

import {StackScreenProps} from '@react-navigation/stack';

import {View, Text, StyleSheet, TouchableOpacity, Image} from 'react-native';

import {RootStackParams} from '../navigation/Navigator';

import Icon from 'react-native-vector-icons/Ionicons';

import {useSafeAreaInsets} from 'react-native-safe-area-context';

import {FadeInImage} from '../components/FadeInImage';

interface Props extends StackScreenProps<RootStackParams, 'PokemonScreen'> {}

export const PokemonScreen = ({navigation, route}: Props) => {

const {simplePokemon, color} = route.params;

const {name, id, picture} = simplePokemon;

const {top} = useSafeAreaInsets();

return (

<View style={{flex: 1}}>

<View style={{...styles.headerContainer, backgroundColor: color}}>

<TouchableOpacity

onPress={() => navigation.pop()}

activeOpacity={0.8}

style={{...styles.backButton, top: top + 10}}>

<Icon name="arrow-back-outline" color="white" size={35} />

</TouchableOpacity>

<Text style={{...styles.pokemonName, top: top + 40}}>

{name + '\n'}#{id}

</Text>

<Image

source={require('../assets/pokebola-blanca.png')}

style={styles.pokeball}

/>

<FadeInImage uri={picture} style={styles.pokemonImage} />

</View>

</View>

);

};

const styles = StyleSheet.create({

headerContainer: {

height: 370,

zIndex: 999,

alignItems: 'center',

borderBottomRightRadius: 1000,

borderBottomLeftRadius: 1000,

},

backButton: {

position: 'absolute',

left: 20,

},

pokemonName: {

color: 'white',

fontSize: 40,

alignSelf: 'flex-start',

left: 20,

},

pokeball: {

width: 200,

height: 200,

bottom: -60,

opacity: 0.7,

},

pokemonImage: {

width: 250,

height: 250,

position: 'absolute',

bottom: -15,

},

});

**INFORMACIÓN COMPLETA DEL POKEMON**

Primero vamos a realizar el tipado, utilizamos la petición http de [**http://pokeapi.co/api/v2/pokemon/${id}**](http://pokeapi.co/api/v2/pokemon/$%7bid%7d) y lo pegamos acá: <https://app.quicktype.io/>

Una vez tengamos las interfaces las pegamos en **pokemonInterfaces.tsx**.

A continuación, creamos el hook **usePokemon.tsx**, la cual hace el pedido a ese endpoint y lo almacena en un state:

import {useEffect, useState} from 'react';

import {PokemonFull} from '../interfaces/pokemonInterfaces';

import {pokemonApi} from '../api/pokemonApi';

export const usePokemon = (id: string) => {

const [isLoading, setIsLoading] = useState(true);

const [pokemon, setPokemon] = useState<PokemonFull>({} as PokemonFull);

const loadPokemon = async () => {

const resp = await pokemonApi.get<PokemonFull>(

`https://pokeapi.co/api/v2/pokemon/${id}`,

);

setPokemon(resp.data);

setIsLoading(false);

};

useEffect(() => {

loadPokemon();

}, []);

return {

isLoading,

pokemon,

};

};

Y por último, en **PokemonScreen.tsx** utilizamos este hook y por ahora solo ponemos un **<ActivityIndicator />** para decirle al usuario que estamos cargando más información:

...

import {usePokemon} from '../hooks/usePokemon';

...

const {pokemon, isLoading} = usePokemon(id);

return (

...

{/\* Detalles del pokemon \*/}

<View style={styles.loadingIndicator}>

<ActivityIndicator color={color} size="large" />

</View>

...

const styles = StyleSheet.create({

...

loadingIndicator: {

flex: 1,

justifyContent: 'center',

alignItems: 'center',

},

});

**INFORMACIÓN ADICIONAL DEL POKEMON**

Vamos a crear el componente **PokemonDetails.tsx** el cual va a recibir por props toda la información de un pokemon y la vamos a mostrar en pantalla:

import React from 'react';

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

import {PokemonFull} from '../interfaces/pokemonInterfaces';

interface Props {

pokemon: PokemonFull;

}

export const PokemonDetails = ({pokemon}: Props) => {

return (

<ScrollView style={{...StyleSheet.absoluteFillObject}}>

<View style={{...styles.container, marginTop: 400}}>

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

<View style={{flexDirection: 'row'}}>

{pokemon.types.map(({type}) => {

return (

<Text

style={{...styles.regularText, marginRight: 10}}

key={type.name}>

{type.name}

</Text>

);

})}

</View>

</View>

<View style={{...styles.container}}>

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

</View>

</ScrollView>

);

};

const styles = StyleSheet.create({

container: {

marginHorizontal: 20,

},

title: {

fontSize: 22,

fontWeight: 'bold',

color: 'black',

marginTop: 20,

},

regularText: {

fontSize: 19,

},

});

Y ahora en la screen **PokemonScreen.tsx** vamos a utilizar el **isLoading** que tenemos del hook creado en el video pasado para renderizar el <ActivityIndicator /> o el <PokemonDetails />:

...

export const PokemonScreen = ({navigation, route}: Props) => {

const {simplePokemon, color} = route.params;

const {name, id, picture} = simplePokemon;

const {top} = useSafeAreaInsets();

const {pokemon, isLoading} = usePokemon(id);

...

{/\* Detalles del pokemon \*/}

{isLoading ? (

<View style={styles.loadingIndicator}>

<ActivityIndicator color={color} size="large" />

</View>

) : (

<PokemonDetails pokemon={pokemon} />

)}

...

**PESO, SPRITES y STATS del POKEMON**

Agregamos más información en **PokemonDetails.tsx**:

import React from 'react';

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

import {PokemonFull} from '../interfaces/pokemonInterfaces';

import {FadeInImage} from './FadeInImage';

interface Props {

pokemon: PokemonFull;

}

export const PokemonDetails = ({pokemon}: Props) => {

return (

<ScrollView

style={{...StyleSheet.absoluteFillObject}}

showsVerticalScrollIndicator={false}>

{/\* TYPES \*/}

<View style={{...styles.container, marginTop: 400}}>

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

<View style={{flexDirection: 'row'}}>

{pokemon.types.map(({type}) => {

return (

<Text

style={{...styles.regularText, marginRight: 10}}

key={type.name}>

{type.name}

</Text>

);

})}

</View>

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

<Text style={styles.regularText}>{pokemon.weight}lb</Text>

</View>

{/\* SPRITES \*/}

<View style={styles.container}>

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

</View>

<ScrollView horizontal={true} showsHorizontalScrollIndicator={false}>

<FadeInImage

uri={pokemon.sprites.front\_default}

style={styles.basicSprite}

/>

<FadeInImage

uri={pokemon.sprites.back\_default}

style={styles.basicSprite}

/>

<FadeInImage

uri={pokemon.sprites.front\_shiny}

style={styles.basicSprite}

/>

<FadeInImage

uri={pokemon.sprites.back\_shiny}

style={styles.basicSprite}

/>

</ScrollView>

{/\* SKILLS \*/}

<View style={styles.container}>

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

<View style={{flexDirection: 'row'}}>

{pokemon.abilities.map(({ability}) => {

return (

<Text

style={{...styles.regularText, marginRight: 10}}

key={ability.name}>

{ability.name}

</Text>

);

})}

</View>

</View>

{/\* MOVES \*/}

<View style={styles.container}>

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

<ScrollView horizontal={true} showsHorizontalScrollIndicator={false}>

{pokemon.moves.map(({move}) => {

return (

<Text

style={{...styles.regularText, marginRight: 10}}

key={move.name}>

{move.name}

</Text>

);

})}

</ScrollView>

</View>

{/\* STATS \*/}

<View style={{...styles.container, marginBottom: 80}}>

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

<View>

{pokemon.stats.map((stat, i) => {

return (

<View key={stat.stat.name + i} style={{flexDirection: 'row'}}>

<Text

style={{...styles.regularText, marginRight: 10, width: 150}}>

{stat.stat.name}

</Text>

<Text style={{...styles.regularText, fontWeight: 'bold'}}>

{stat.base\_stat}

</Text>

</View>

);

})}

</View>

</View>

</ScrollView>

);

};

const styles = StyleSheet.create({

container: {

marginHorizontal: 20,

},

title: {

fontSize: 22,

fontWeight: 'bold',

color: 'black',

marginTop: 20,

},

regularText: {

fontSize: 19,

color: 'black',

},

basicSprite: {

width: 100,

height: 100,

},

});

**SEC-15: DEBOUNCER y BÚSQUEDAS**

**IMPLEMENTAR TABS EN NUESTRA APLICACIÓN**

Vamos a instalar los **Bottom Tabs Navigator** desde acá:

<https://reactnavigation.org/docs/bottom-tab-navigator/#props>

Luego vamos a crear la screen **SearchScreen.tsx** la cual será nuestra página de búsqueda. (La vamos a editar en el próximo video).

Ahora en la carpeta **navigation** creamos el archivo **Tabs.tsx**, en el cual tendrá la navegación del **Navigator.tsx** y la del **SearchScreen.tsx**:

import React from 'react';

import {createBottomTabNavigator} from '@react-navigation/bottom-tabs';

import {Navigator} from './Navigator';

import {SearchScreen} from '../screens/SearchScreen';

import {Platform} from 'react-native';

import Icon from 'react-native-vector-icons/Ionicons';

const Tab = createBottomTabNavigator();

export const Tabs = () => {

return (

<Tab.Navigator

sceneContainerStyle={{

backgroundColor: 'white',

}}

screenOptions={{

tabBarActiveTintColor: '#5856D6',

tabBarLabelStyle: {

marginBottom: Platform.OS === 'android' ? 10 : 0,

},

tabBarStyle: {

position: 'absolute',

backgroundColor: 'rgba(255, 255, 255, 0.92)',

borderWidth: 0,

elevation: 0,

height: Platform.OS === 'ios' ? 80 : 60,

},

headerShown: false

}}>

<Tab.Screen

name="Navigator"

component={Navigator}

options={{

tabBarLabel: 'List',

tabBarIcon: ({color}) => (

<Icon name="list-outline" color={color} size={25} />

),

}}

/>

<Tab.Screen

name="SearchScreen"

component={SearchScreen}

options={{

tabBarLabel: 'Search',

tabBarIcon: ({color}) => (

<Icon name="search-outline" color={color} size={25} />

),

}}

/>

</Tab.Navigator>

);

};

Y por último, en el **App.tsx** en vez de renderizar el **<Navigator />** ahora renderizamos este **<Tabs />**:

import 'react-native-gesture-handler';

import React from 'react';

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

import {Tabs} from './src/navigation/Tabs';

const App = () => {

return (

<NavigationContainer>

<Tabs />

</NavigationContainer>

);

};

export default App;

**SEARCH SCREEN – DISEÑO**

Vamos a crear el componente **SearchInput.tsx** el cual contendrá el input para buscar el pokemon:

import React from 'react';

import {View, StyleSheet, TextInput, Platform} from 'react-native';

import Icon from 'react-native-vector-icons/Ionicons';

export const SearchInput = () => {

return (

<View style={styles.container}>

<View style={styles.textBackground}>

<TextInput

placeholder="Search Pokemon"

style={styles.textInput}

autoCapitalize="none"

autoCorrect={false}

/>

<Icon name="search-outline" color="grey" size={30} />

</View>

</View>

);

};

const styles = StyleSheet.create({

container: {},

textBackground: {

backgroundColor: '#F3F1F3',

borderRadius: 50,

height: 40,

paddingHorizontal: 20,

flexDirection: 'row',

justifyContent: 'center',

alignItems: 'center',

shadowColor: '#000',

shadowOffset: {

width: 0,

height: 3,

},

shadowOpacity: 0.27,

shadowRadius: 4.65,

elevation: 6,

},

textInput: {

flex: 1,

fontSize: 18,

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

},

});

Y luego lo utilizamos en **SearchScreen.tsx**:

import React from 'react';

import {View, Platform} from 'react-native';

import {useSafeAreaInsets} from 'react-native-safe-area-context';

import {SearchInput} from '../components/SearchInput';

export const SearchScreen = () => {

const {top} = useSafeAreaInsets();

return (

<View

style={{

flex: 1,

marginTop: Platform.OS === 'ios' ? top : top + 10,

marginHorizontal: 20,

}}>

<SearchInput />

</View>

);

};

**PRE-CARGAR LA INFORMACIÓN PARA BUSCAR POR NOMBRE**

Para hacer esto, vamos a crear un hook **usePokemonSearch.tsx** el cual nos devuelve el listado de todos los pokemon en un array:

import {useEffect, useState} from 'react';

import {pokemonApi} from '../api/pokemonApi';

import {

PokemonPaginatedResponse,

Result,

SimplePokemon,

} from '../interfaces/pokemonInterfaces';

export const usePokemonSearch = () => {

const [isFetching, setIsFetching] = useState(true);

const [simplePokemonList, setSimplePokemonList] = useState<SimplePokemon[]>(

[],

);

useEffect(() => {

loadPokemons();

}, []);

const loadPokemons = async () => {

const resp = await pokemonApi.get<PokemonPaginatedResponse>(

'https://pokeapi.co/api/v2/pokemon?limit=1281',

);

mapPokemonList(resp.data.results);

};

const mapPokemonList = (pokemonList: Result[]) => {

const newPokemonList: SimplePokemon[] = pokemonList.map(({name, url}) => {

const urlParts = url.split('/');

const id = urlParts[urlParts.length - 2];

const picture = `https://raw.githubusercontent.com/PokeAPI/sprites/master/sprites/pokemon/other/official-artwork/${id}.png`;

return {

id,

picture,

name,

};

});

setSimplePokemonList(newPokemonList);

setIsFetching(false);

};

return {

simplePokemonList,

isFetching,

};

};

Ahora creamos un componente **Loading.tsx** el cual muestra un <ActivityIndicator />:

import React from 'react';

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

export const Loading = () => {

return (

<View style={styles.activityContainer}>

<ActivityIndicator size="large" color="grey" />

<Text style={styles.loadingText}>Cargando...</Text>

</View>

);

};

const styles = StyleSheet.create({

activityContainer: {

flex: 1,

justifyContent: 'center',

alignItems: 'center',

},

loadingText: {

marginTop: 10,

},

});

Y luego en **SearchScreen.tsx** vamos a mostrar el componente **<Loading />** hasta que consiga toda la información y mostraremos un <FlatList /> de todos los pokemones, por ahora. Luego será por la búsqueda:

import React from 'react';

import {

View,

Platform,

StyleSheet,

Text,

FlatList,

Dimensions,

} from 'react-native';

import {useSafeAreaInsets} from 'react-native-safe-area-context';

import {SearchInput} from '../components/SearchInput';

import {usePokemonSearch} from '../hooks/usePokemonSearch';

import {styles} from '../theme/appTheme';

import {PokemonCard} from '../components/PokemonCard';

import {Loading} from '../components/Loading';

const screenWidth = Dimensions.get('window').width;

export const SearchScreen = () => {

const {top} = useSafeAreaInsets();

const {isFetching, simplePokemonList} = usePokemonSearch();

if (isFetching) {

return <Loading />;

}

return (

<View

style={{

...stylesSearch.searchContainer,

}}>

<SearchInput

style={{

position: 'absolute',

zIndex: 999,

width: screenWidth - 40,

top: Platform.OS === 'ios' ? top : top + 30,

}}

/>

<FlatList

data={simplePokemonList}

keyExtractor={pokemon => pokemon.id}

showsVerticalScrollIndicator={false}

numColumns={2}

renderItem={({item}) => <PokemonCard pokemon={item} />}

ListHeaderComponent={

<Text

style={{

...styles.title,

...styles.globalMargin,

marginTop: Platform.OS === 'ios' ? top + 60 : top + 80,

paddingBottom: 10,

color: 'black',

}}>

Pokedex

</Text>

}

/>

</View>

);

};

const stylesSearch = StyleSheet.create({

searchContainer: {

flex: 1,

marginHorizontal: 20,

},

});

Como vemos el **<SearchInput />** ahora recibe como props los style, asi que debemos crear la interface de esas props:

**SearchInput.tsx:**

import React from 'react';

import {

View,

StyleSheet,

TextInput,

Platform,

StyleProp,

ViewStyle,

} from 'react-native';

import Icon from 'react-native-vector-icons/Ionicons';

interface Props {

style?: StyleProp<ViewStyle>;

}

export const SearchInput = ({style}: Props) => {

return (

<View style={{...styles.container, ...(style as any)}}>

<View style={styles.textBackground}>

<TextInput

placeholder="Search Pokemon"

style={styles.textInput}

autoCapitalize="none"

autoCorrect={false}

/>

<Icon name="search-outline" color="grey" size={30} />

</View>

</View>

);

};

const styles = StyleSheet.create({

container: {},

textBackground: {

backgroundColor: '#F3F1F3',

borderRadius: 50,

height: 40,

paddingHorizontal: 20,

flexDirection: 'row',

justifyContent: 'center',

alignItems: 'center',

shadowColor: '#000',

shadowOffset: {

width: 0,

height: 3,

},

shadowOpacity: 0.27,

shadowRadius: 4.65,

elevation: 6,

},

textInput: {

flex: 1,

fontSize: 18,

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

},

});

**useDebounceValue – CUSTOM HOOK**

En la carpeta **hooks** creamos **useDebouncedValue.tsx** el cual recibe un string y una cantidad de tiempo, y luego aplica un useEffect que setea en un timeout el valor del input, y se reinicia cada vez que ese input cambia:

import {useEffect, useState} from 'react';

export const useDebouncedValue = (input: string = '', time: number = 500) => {

const [debouncedValue, setDebouncedValue] = useState(input);

useEffect(() => {

const timeout = setTimeout(() => {

setDebouncedValue(input);

}, time);

return () => {

clearTimeout(timeout);

};

}, [input]);

return {debouncedValue};

};

Y ahora en en el componente **SearchInput.tsx** creamos un state para guardar el valor del campo del **TextInput** y pasarlo al hook:

...

export const SearchInput = ({style}: Props) => {

const [textValue, setTextValue] = useState('');

const {debouncedValue} = useDebouncedValue(textValue, 500);

useEffect(() => {

console.log(debouncedValue)

}, [debouncedValue])

return (

<View style={{...styles.container, ...(style as any)}}>

<View style={styles.textBackground}>

<TextInput

placeholder="Search Pokemon"

style={styles.textInput}

autoCapitalize="none"

autoCorrect={false}

value={textValue}

onChangeText={setTextValue}

/>

...

**FILTRANDO POKEMONS BASADOS EN EL TÉRMINO DE BÚSQUEDA**

Ahora debemos hacer la conexión entre la screen **SearchScreen** y el componente **SearchInput**. Para lograr esto, debemos hacer que el SearchInput reciba una función la cual setee el valor que reciba desde el buscador.

Y luego en SearchScreen debemos hacer un useEffect que setee un nuevo array con los pokemones filtrados dependiendo del valor seteado en el buscador:

**SearchScreen.tsx**:

...

export const SearchScreen = () => {

const {top} = useSafeAreaInsets();

const {isFetching, simplePokemonList} = usePokemonSearch();

const [term, setTerm] = useState('');

const [pokemonFiltered, setPokemonFiltered] = useState<SimplePokemon[]>([]);

useEffect(() => {

if (term.length === 0) {

return setPokemonFiltered([]);

}

setPokemonFiltered(

simplePokemonList.filter(poke =>

poke.name.toLowerCase().includes(term.toLowerCase()),

),

);

}, [term]);

if (isFetching) {

return <Loading />;

}

return (

<View

style={{

...stylesSearch.searchContainer,

}}>

<SearchInput

onDebounce={value => setTerm(value)}

style={{

position: 'absolute',

zIndex: 999,

width: screenWidth - 40,

top: Platform.OS === 'ios' ? top : top + 30,

}}

/>

...

**SearchInput.tsx:**

...

interface Props {

onDebounce: (value: string) => void;

style?: StyleProp<ViewStyle>;

}

export const SearchInput = ({style, onDebounce}: Props) => {

const [textValue, setTextValue] = useState('');

const {debouncedValue} = useDebouncedValue(textValue, 500);

useEffect(() => {

onDebounce(debouncedValue);

}, [debouncedValue]);

...

**BUSCAR POR ID**

En el **useEffect** dentro de **SearchScreen.tsx** aplicamos un if y else, para que busque por string o por id:

...

useEffect(() => {

if (term.length === 0) {

return setPokemonFiltered([]);

}

if (isNaN(Number(term))) {

//si no es un numero busca por string

setPokemonFiltered(

simplePokemonList.filter(poke =>

poke.name.toLowerCase().includes(term.toLowerCase()),

),

);

} else {

// es un número

const pokemonById = simplePokemonList.find(poke => poke.id === term);

setPokemonFiltered(pokemonById ? [pokemonById] : []);

}

}, [term]);

...

**DETALLES ESTÉTICOS DE LA PANTALLA DE BÚSQUEDA**

Ahora para que no nos pase que al estar en la pantalla de búsqueda y clickeamos en el pokemon buscado y al volver nos lleve al listado de todos los pokemons, debemos hacer que nuestro search sea un **navigator**. Entonces nuestro **Navigator.tsx** pasa a ser **Tab1.tsx** y tenemos el **Tab2.tsx** que es el buscador y el detalle del pokemon y el **Tabs.tsx** que agarra a estos dos tabs:

**Tabs.tsx:**

import React from 'react';

import {createBottomTabNavigator} from '@react-navigation/bottom-tabs';

import {Tab1} from './Tab1';

import {Platform} from 'react-native';

import Icon from 'react-native-vector-icons/Ionicons';

import {Tab2Screen} from './Tab2';

const Tab = createBottomTabNavigator();

export const Tabs = () => {

return (

<Tab.Navigator

sceneContainerStyle={{

backgroundColor: 'white',

}}

screenOptions={{

tabBarActiveTintColor: '#5856D6',

tabBarLabelStyle: {

marginBottom: Platform.OS === 'android' ? 10 : 0,

},

tabBarStyle: {

position: 'absolute',

backgroundColor: 'rgba(255, 255, 255, 0.92)',

borderWidth: 0,

elevation: 0,

height: Platform.OS === 'ios' ? 80 : 60,

},

headerShown: false,

}}>

<Tab.Screen

name="Home"

component={Tab1}

options={{

tabBarLabel: 'List',

tabBarIcon: ({color}) => (

<Icon name="list-outline" color={color} size={25} />

),

}}

/>

<Tab.Screen

name="Search"

component={Tab2Screen}

options={{

tabBarLabel: 'Search',

tabBarIcon: ({color}) => (

<Icon name="search-outline" color={color} size={25} />

),

}}

/>

</Tab.Navigator>

);

};

**Tab1.tsx:**

import React from 'react';

import {createStackNavigator} from '@react-navigation/stack';

import {HomeScreen} from '../screens/HomeScreen';

import {PokemonScreen} from '../screens/PokemonScreen';

import {SimplePokemon} from '../interfaces/pokemonInterfaces';

export type RootStackParams = {

HomeScreen: undefined;

PokemonScreen: {simplePokemon: SimplePokemon; color: string};

};

const Stack = createStackNavigator<RootStackParams>();

export const Tab1 = () => {

return (

<Stack.Navigator

screenOptions={{

headerShown: false,

cardStyle: {backgroundColor: 'white'},

}}>

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

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

</Stack.Navigator>

);

};

**Tab2.tsx:**

import React from 'react';

import {createStackNavigator} from '@react-navigation/stack';

import {RootStackParams} from './Tab1';

import {SearchScreen} from '../screens/SearchScreen';

import {PokemonScreen} from '../screens/PokemonScreen';

const Tab2 = createStackNavigator<RootStackParams>();

export const Tab2Screen = () => {

return (

<Tab2.Navigator

screenOptions={{

headerShown: false,

cardStyle: {backgroundColor: 'white'},

}}>

<Tab2.Screen name="HomeScreen" component={SearchScreen} />

<Tab2.Screen name="PokemonScreen" component={PokemonScreen} />

</Tab2.Navigator>

);

};

**SEC-16: RUTAS APP – PERMISOS – APLICACIÓN CON MAPAS – GOOGLE Y APPLE MAPS**

**INICIO DE PROYECTO**

Creamos el proyecto e instalamos los íconos. Buscar “Íconos en nuestra aplicación” en este Word para ver los pasos de instalación.

**RUTAS y STACKNAVIGATOR**

Instalamos react navigation y el stack.

Luego creamos las screen **MapsScreen** y **PermissionsScreen** para utilizar en el archivo **Navigator.tsx** dentro de la carpeta **navigation**:

import React from 'react';

import {createStackNavigator} from '@react-navigation/stack';

import {MapScreen} from '../pages/MapScreen';

import {PermissionsScreen} from '../pages/PermissionsScreen';

const Stack = createStackNavigator();

export const Navigator = () => {

return (

<Stack.Navigator

screenOptions={{

headerShown: false,

cardStyle: {backgroundColor: 'white'},

}}>

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

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

</Stack.Navigator>

);

};

Y en nuestro **App.tsx** importamos este Stack:

import 'react-native-gesture-handler';

import React from 'react';

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

import {Navigator} from './src/navigation/Navigator';

const App = () => {

return (

<NavigationContainer>

<Navigator />

</NavigationContainer>

);

};

export default App;

**ANDROID: CONFIGURACIÓN INICIAL DE PERMISOS DE GPS**

Necesitamos instalar un paquete con el cual podemos manejar todos los permisos:

<https://www.npmjs.com/package/react-native-permissions>

Lo instalamos con el comando: **npm install react-native-permissions**

Debemos dirigirnos al archivo ubicado en **android/app/src/main/AndroidManifest.xml**

Y una vez ahí en la documentación están todos los permisos que podemos usar. Nosotros vamos a utilizar estos para el gps:

...

<uses-permission android:name="android.permission.ACCESS\_BACKGROUND\_LOCATION" />

<uses-permission android:name="android.permission.ACCESS\_COARSE\_LOCATION" />

<uses-permission android:name="android.permission.ACCESS\_FINE\_LOCATION" />

...

**IOS: CONFIGURACIÓN INICIAL DE PERMISOS DE GPS**

Necesitamos instalar un paquete con el cual podemos manejar todos los permisos:

<https://www.npmjs.com/package/react-native-permissions>

Lo instalamos con el comando: **npm install react-native-permissions**

Ahora nos dirigimos al **package.json** y pegamos los permisos que necesitamos:

"reactNativePermissionsIOS": [

"LocationAccuracy",

"LocationAlways",

"LocationWhenInUse"

],

A continuación corremos los comandos:

**npx react-native setup-ios-permissions**

**npx pod install**

Luego en el archivo ubicado en **ios/nombreDelProyecto/Info.plist** buscamos si tenemos algunos de los siguientes permisos y los reemplazamos por estos en este caso:

...

<key>NSLocationAlwaysAndWhenInUseUsageDescription</key>

<string>Necesito saber tu ubicacion para mostrar el mapa</string>

<key>NSLocationAlwaysUsageDescription</key>

<string>Necesito saber tu ubicacion para mostrar el mapa</string>

<key>NSLocationWhenInUseUsageDescription</key>

<string>Necesito saber tu ubicacion para mostrar el mapa</string>

...

**SOLICITAR y REVISAR PERMISOS – CHECK y REQUEST PERMISSION**

En la screen **PermissionsScreen.tsx** vamos a utilizar las funciones **check** y **request** de react-native-permissions, para comprobar dependiendo el dispositivo si tiene habilitado el gps en la aplicación:

import React from 'react';

import {View, Text, StyleSheet, Button, Platform} from 'react-native';

import {

PERMISSIONS,

PermissionStatus,

check,

request,

} from 'react-native-permissions';

export const PermissionsScreen = () => {

const checkLocationPermission = async () => {

let permissionStatus: PermissionStatus;

if (Platform.OS === 'ios') {

// permissionStatus = await check(PERMISSIONS.IOS.LOCATION\_WHEN\_IN\_USE);

permissionStatus = await request(PERMISSIONS.IOS.LOCATION\_WHEN\_IN\_USE);

console.log('IOS: ', permissionStatus);

}

if (Platform.OS === 'android') {

// permissionStatus = await check(PERMISSIONS.ANDROID.ACCESS\_FINE\_LOCATION);

permissionStatus = await request(

PERMISSIONS.ANDROID.ACCESS\_FINE\_LOCATION,

);

console.log('Android: ', permissionStatus);

}

};

return (

<View style={styles.container}>

<Text>PermissionsScreen</Text>

<Button title="Permiso" onPress={checkLocationPermission} />

</View>

);

};

const styles = StyleSheet.create({

container: {

flex: 1,

justifyContent: 'center',

alignItems: 'center',

},

});

**PERMISSION PROVIDER – CONTEXT API**

Creamos la carpeta **context** con el archivo **PermissionsContext.tsx** el cual va a devolver los permisos que tengamos, y dos funciones, una que pregunte si quiere permitir un permiso de gps y otra que chequee si ya tiene el permiso otorgado:

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

import {PermissionStatus} from 'react-native-permissions';

export interface PermissionsState {

locationStatus: PermissionStatus;

}

export const permissionsInitiState: PermissionsState = {

locationStatus: 'unavailable',

};

type PermissionsContextProps = {

permissions: PermissionsState;

askLocationPermission: () => void;

checkLocationPermission: () => void;

};

export const PermissionsContex = createContext({} as PermissionsContextProps);

export const PermissionsProvider = ({children}: any) => {

const [permissions, setPermissions] = useState(permissionsInitiState);

const askLocationPermission = () => {};

const checkLocationPermission = () => {};

return (

<PermissionsContex.Provider

value={{permissions, askLocationPermission, checkLocationPermission}}>

{children}

</PermissionsContex.Provider>

);

};

Y luego envolvemos toda nuestra aplicación con este context:

import 'react-native-gesture-handler';

import React from 'react';

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

import {Navigator} from './src/navigation/Navigator';

import {PermissionsProvider} from './src/context/PermissionsContext';

const AppState = ({children}: any) => {

return (

<PermissionsProvider>

{children}

</PermissionsProvider>

);

};

const App = () => {

return (

<NavigationContainer>

<AppState>

<Navigator />

</AppState>

</NavigationContainer>

);

};

export default App;

**USO DEL PERMISSION CONTEXT**

Ahora de la función que tenemos en **PermissionsScreen.tsx** para validar el permiso, debemos pasar toda esa lógica a la función **askLocationPermission** del contex:

**PermissionsContext.tsx:**

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

import {Platform} from 'react-native';

import {PERMISSIONS, PermissionStatus, request} from 'react-native-permissions';

export interface PermissionsState {

locationStatus: PermissionStatus;

}

export const permissionsInitiState: PermissionsState = {

locationStatus: 'unavailable',

};

type PermissionsContextProps = {

permissions: PermissionsState;

askLocationPermission: () => void;

checkLocationPermission: () => void;

};

export const PermissionsContex = createContext({} as PermissionsContextProps);

export const PermissionsProvider = ({children}: any) => {

const [permissions, setPermissions] = useState(permissionsInitiState);

const askLocationPermission = async () => {

let permissionStatus: PermissionStatus;

if (Platform.OS === 'ios') {

permissionStatus = await request(PERMISSIONS.IOS.LOCATION\_WHEN\_IN\_USE);

} else {

permissionStatus = await request(

PERMISSIONS.ANDROID.ACCESS\_FINE\_LOCATION,

);

}

setPermissions({

...permissions,

locationStatus: permissionStatus,

});

};

const checkLocationPermission = () => {};

return (

<PermissionsContex.Provider

value={{permissions, askLocationPermission, checkLocationPermission}}>

{children}

</PermissionsContex.Provider>

);

};

Y ahora en nuestro **PermissionsScreen.tsx** importamos el context con esta función:

import React, {useContext} from 'react';

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

import {PermissionsContex} from '../context/PermissionsContext';

export const PermissionsScreen = () => {

const {permissions, askLocationPermission} = useContext(PermissionsContex);

return (

<View style={styles.container}>

<Text>PermissionsScreen</Text>

<Button title="Permiso" onPress={askLocationPermission} />

<Text>{JSON.stringify(permissions, null, 5)}</Text>

</View>

);

};

const styles = StyleSheet.create({

container: {

flex: 1,

justifyContent: 'center',

alignItems: 'center',

},

});

**REVISAR EL PERMISO DE GPS AL REGRESAR A LA APP**

Ahora debemos trabajar con la función que chequee si están los permisos habilitados. Para eso, en el **PermissionsContex.tsx** creamos la función **checkLocationPermission** que se ejecute en un **useEffect** siempre y cuando el status de la aplicación sea **‘active’**.

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

import {AppState, Platform} from 'react-native';

import {

PERMISSIONS,

PermissionStatus,

check,

request,

} from 'react-native-permissions';

export interface PermissionsState {

locationStatus: PermissionStatus;

}

export const permissionsInitiState: PermissionsState = {

locationStatus: 'unavailable',

};

type PermissionsContextProps = {

permissions: PermissionsState;

askLocationPermission: () => void;

checkLocationPermission: () => void;

};

export const PermissionsContex = createContext({} as PermissionsContextProps);

export const PermissionsProvider = ({children}: any) => {

const [permissions, setPermissions] = useState(permissionsInitiState);

useEffect(() => {

checkLocationPermission(); //es para que no quede recargando

AppState.addEventListener('change', state => {

if (state !== 'active') return;

checkLocationPermission();

});

}, []);

const askLocationPermission = async () => {

let permissionStatus: PermissionStatus;

if (Platform.OS === 'ios') {

permissionStatus = await request(PERMISSIONS.IOS.LOCATION\_WHEN\_IN\_USE);

} else {

permissionStatus = await request(

PERMISSIONS.ANDROID.ACCESS\_FINE\_LOCATION,

);

}

setPermissions({

...permissions,

locationStatus: permissionStatus,

});

};

const checkLocationPermission = async () => {

let permissionStatus: PermissionStatus;

if (Platform.OS === 'ios') {

permissionStatus = await check(PERMISSIONS.IOS.LOCATION\_WHEN\_IN\_USE);

} else {

permissionStatus = await check(PERMISSIONS.ANDROID.ACCESS\_FINE\_LOCATION);

}

setPermissions({

...permissions,

locationStatus: permissionStatus,

});

};

return (

<PermissionsContex.Provider

value={{permissions, askLocationPermission, checkLocationPermission}}>

{children}

</PermissionsContex.Provider>

);

};

**DETECTAR QUÉ PANTALLA MOSTRAR AL INICIO DE FORMA AUTOMÁTICA**

Para renderizar una vista u otra dependiendo si tenemos los permisos habilitados o no, vamos a ir al **Navigator.tsx** y utilizar el state del Context que creamos, para mostrar nuestra screen de <Loading />, el <MapScreen /> o el <PermissionsScreen />:

import React, {useContext} from 'react';

import {createStackNavigator} from '@react-navigation/stack';

import {MapScreen} from '../pages/MapScreen';

import {PermissionsScreen} from '../pages/PermissionsScreen';

import {PermissionsContex} from '../context/PermissionsContext';

import {LoadingScreen} from '../pages/LoadingScreen';

const Stack = createStackNavigator();

export const Navigator = () => {

const {permissions} = useContext(PermissionsContex);

if (permissions.locationStatus === 'unavailable') {

return <LoadingScreen />;

}

return (

<Stack.Navigator

screenOptions={{

headerShown: false,

cardStyle: {backgroundColor: 'white'},

}}>

{permissions.locationStatus === 'granted' ? (

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

) : (

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

)}

</Stack.Navigator>

);

};

**LoadingScreen.tsx:**

import React from 'react';

import {View, ActivityIndicator} from 'react-native';

export const LoadingScreen = () => {

return (

<View style={{flex: 1, justifyContent: 'center', alignItems: 'center'}}>

<ActivityIndicator size="large" color="black" />

</View>

);

};

**ABRIR AJUSTES DE LOCALIZACIÓN SI EL PERMISO ESTÁ BLOQUEADO POR EL USUARIO**

Para poder abrir los settings cuando el permiso es ‘blocked’ debemos usar la función **openSettings** de react-native-permissions. Lo vamos a utilizar en la función **askLocationPermission** del **PermissionsContext.tsx**:

import { PERMISSIONS, PermissionStatus, check, openSettings, request } from 'react-native-permissions';

...

const askLocationPermission = async () => {

let permissionStatus: PermissionStatus;

if (Platform.OS === 'ios') {

permissionStatus = await request(PERMISSIONS.IOS.LOCATION\_WHEN\_IN\_USE);

} else {

permissionStatus = await request(

PERMISSIONS.ANDROID.ACCESS\_FINE\_LOCATION,

);

}

if (permissionStatus === 'blocked') {

openSettings();

}

setPermissions({

...permissions,

locationStatus: permissionStatus,

});

};

...

**BOTÓN ESTILIZADO**

Creamos en la carpeta **component** el archivo **BlackButton.tsx**:

import React from 'react';

import {

StyleProp,

StyleSheet,

Text,

TouchableOpacity,

View,

ViewStyle,

} from 'react-native';

interface Props {

title: string;

onPress: () => void;

style?: StyleProp<ViewStyle>;

}

const BlackButton = ({title, onPress, style = {}}: Props) => {

return (

<TouchableOpacity

activeOpacity={0.9}

onPress={onPress}

style={{...(style as any), ...styles.blackButton}}>

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

</TouchableOpacity>

);

};

export default BlackButton;

const styles = StyleSheet.create({

blackButton: {

height: 45,

width: 200,

backgroundColor: 'black',

borderRadius: 50,

justifyContent: 'center',

alignItems: 'center',

shadowColor: '#000',

shadowOffset: {

width: 0,

height: 3,

},

shadowOpacity: 0.27,

elevation: 6,

},

buttonText: {

color: 'white',

fontSize: 18,

},

});

Y en **PermissionsScreen.tsx** lo utilizamos por el botón que teníamos. Además agregamos algunos estilos al título y al state que mostramos en pantalla:

import React, {useContext} from 'react';

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

import {PermissionsContex} from '../context/PermissionsContext';

import BlackButton from '../components/BlackButton';

export const PermissionsScreen = () => {

const {permissions, askLocationPermission} = useContext(PermissionsContex);

return (

<View style={styles.container}>

<Text style={styles.title}>

Es necesario el uso del GPS para usar esta aplicación

</Text>

<BlackButton title="Permiso" onPress={askLocationPermission} />

<Text style={{marginTop: 20}}>

{JSON.stringify(permissions, null, 5)}

</Text>

</View>

);

};

const styles = StyleSheet.create({

container: {

flex: 1,

justifyContent: 'center',

alignItems: 'center',

},

title: {

width: 250,

fontSize: 18,

textAlign: 'center',

marginBottom: 20,

},

});

**SEC-17: RUTAS APP – APP CON MAPAS – GOOGLE y APPLE MAPS**

**CONFIGURACIÓN DE GOOGLE MAPS**

Vamos a la documentación en: <https://github.com/react-native-maps/react-native-maps> y nos dirigimos a **See installation instructions**, el cual es un link que nos lleva a:

<https://github.com/react-native-maps/react-native-maps/blob/master/docs/installation.md>

Corremos el comando: **npm install react-native-maps**

Luego clickeamos en el link que dice **API key for the IOS SDK**, y vamos a clickear ahora en **Ir a la página Credenciales**.

Vamos a aparecer en un dashboard, en el cual debemos clickear en **Selecciona un proyecto>Crear Proyecto**, el cual está ubicado arriba en el header. Le damos un nombre y lo creamos.

Una vez creado vamos en el menú lateral izquierdo a la sección **APIs y Servicios habilitados** y luego clickeamos en  **+ Habilitar APIs y Servicios**. Ahora debemos habilitar el **Maps SDK for Android** y **Maps SDK for iOS**.

Ahora nos dirigimos en el menú lateral izquierdo a **credenciales**, y creamos una nueva credencial de **Clave de API**. Luego debemos restringir esa clave para el SDK de Android e iOS.

**ANDROID: CONFIGURACIÓN de GOOGLE MAPS**

Nos dirigimos en el proyecto a la ruta **Android/app/src/main/AndroidManifest.xml** y dentro de la etiqueta **<application>** pegamos la siguiente etiquea con el valor de la API hecha en el video anterior:

...

<meta-data

android:name="com.google.android.geo.API\_KEY"

android:value="AIzaSyD-jwxTRqNX3mhwy1LwBT5tjULyK1wnJIM"/>

...

Y ahora podemos usar nuestro mapa con la importación de **MapView** en la screen **MapScreen.tsx** (solo lo usaremos como prueba para que funcione, después haremos un componente):

import React from 'react';

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

import MapView, { PROVIDER\_GOOGLE } from 'react-native-maps';

export const MapScreen = () => {

return (

<View style={{flex: 1}}>

<MapView

provider={PROVIDER\_GOOGLE} // remove if not using Google Maps

style={styles.map}

region={{

latitude: 37.78825,

longitude: -122.4324,

latitudeDelta: 0.015,

longitudeDelta: 0.0121,

}}

>

</MapView>

</View>

);

};

const styles = StyleSheet.create({

map: {

...StyleSheet.absoluteFillObject,

},

});

**iOS: CONFIGURACIÓN DE GOOGLE MAPS**

Después de haber hecho la instalación de **npm install react-native-maps** debemos ejecutar el comando **npx pod-install**.

Con esto ya nos debería andar el **Apple Maps**, pero si queremos el **Google Maps** debemos ir a la ruta **ios/nombreDelProyecto/AppDelegate.mm** y copiar la siguientes líneas:

...

#import <GoogleMaps/GoogleMaps.h>

...

@implementation AppDelegate

- (BOOL)application:(UIApplication \*)application didFinishLaunchingWithOptions:(NSDictionary \*)launchOptions

{

[GMSServices provideAPIKey:@"\_YOUR\_API\_KEY"];

...

}

...

Y ahora en el archivo **ios/Podfile** vamos a pegar las siguientes líneas, primero casi arriba de todo debemos verificar que tengamos esta versión:

platform :ios, '13.0'

Y luego arriba de **config = use\_native\_modules!**:

...

target 'RutasApp' do

pod 'Google-Maps-iOS-Utils', :git => 'https://github.com/Simon-TechForm/google-maps-ios-utils.git', :branch => 'feat/support-apple-silicon'

rn\_maps\_path = '../node\_modules/react-native-maps'

pod 'react-native-google-maps', :path => rn\_maps\_path

config = use\_native\_modules!

...

Y ejecutamos nuevamente el **npx pod-install**

Y no olvidarse de cuando usemos el **<MapView/>** de pasarle el **provider** de Google:

import MapView, { PROVIDER\_GOOGLE } from 'react-native-maps';

...

<MapView

provider={PROVIDER\_GOOGLE} // remove if not using Google Maps

style={styles.map}

region={{

latitude: 37.78825,

longitude: -122.4324,

latitudeDelta: 0.015,

longitudeDelta: 0.0121,

}}

/>

...

**FIX LOADING INFINITO EN OCASIONES**

En el **useEffect** que tenemos en el **PermissionsContext.tsx** debemos agregar antes de del **AppState.addEventListener** la función **checkLocationPermission()** y listo:

...

useEffect(() => {

checkLocationPermission();

AppState.addEventListener('change', state => {

if (state !== 'active') return;

checkLocationPermission();

});

}, []);

...

**MARCADORES y COMPONENTE MAP RE-UTILIZABLE**

En la carpeta **components** creamos el archivo **Map.tsx**, para que tengamos el mapa de manera reutilizable. Y podemos usar el **<Marker/>** para agregar un marcador si quisiéramos en el mapa:

import React from 'react';

import MapView, {Marker, PROVIDER\_GOOGLE} from 'react-native-maps';

export const Map = () => {

return (

<>

<MapView

provider={PROVIDER\_GOOGLE} // remove if not using Google Maps

style={{flex: 1}}

region={{

latitude: 37.78825,

longitude: -122.4324,

latitudeDelta: 0.015,

longitudeDelta: 0.0121,

}}>

<Marker

image={require('../assets/custom-marker.png')} //esto es por si queres un custom marcador

coordinate={{

latitude: 37.78825,

longitude: -122.4324,

}}

title="Titulo del Marcador"

description="Descripcion del marcador"

/>

</MapView>

</>

);

};

Notar que estamos usando un marcador el cual descargamos y guardamos dentro de una carpeta **assets**.

(No lo vamos a usar al Marker, era solo demostración)

Y ahora en el **MapScreen.tsx** utilizamos este componente **<Map/>**:

import React from 'react';

import {View} from 'react-native';

import {Map} from '../components/Map';

export const MapScreen = () => {

return (

<View style={{flex: 1}}>

<Map />

</View>

);

};

**MapView – MOSTRAR LA UBICACIÓN DEL USUARIO**

Debemos colocarle al **<MapView/>** la propiedad **showsUserLocation** y nada más:

<MapView

...

showsUserLocation

...

/>

**OBTENER LAS COORDENADAS DEL USUARIO**

Vamos a utilizar el paquete **react-native-geolocation**:

<https://github.com/michalchudziak/react-native-geolocation>

Lo instalamos con el comando: **npm install @react-native-community/geolocation**

Y ejecutamos el **npx pod-install**

Luego todas las demás configuraciones que dice la documentación las hicimos en los primeros videos de esta sección.

Ahora vamos a usarlo solo de prueba en **Map.tsx**, luego haremos un hook. Debemos importar **Geolocation** y utilizar la función **getCurrentPosition()**, la cual recibe 3 parámetros: el callback que se ejecuta en caso de éxito, el callback en caso de error, y las opciones.

**Map.tsx:**

...

import Geolocation from '@react-native-community/geolocation';

export const Map = () => {

useEffect(() => {

Geolocation.getCurrentPosition(

info => console.log(info),

err => console.log({err}),

{

enableHighAccuracy: true, //nos da la ubicación exacta

},

);

}, []);

...

**CUSTOM HOOK – useLocation**