01 REACT NATIVE - BASICS

- Para crear el proyecto

> npx react-native@latest init Proyecto

Zustand - Gestor de estado

- En src creo las carpetas api, components, hookes, interfaces, **store**
- Para instalar

> npm i zustand

- Para crear el store global creo en /store/**auth.store.ts** una interfaz con los tipos de lo que voy a guardar en el stado
- Importo create de zustand y le paso el AuthState. Luego abro dos pares de paréntesis, y dentro del segundo par coloco un callback que devuelve un objeto entre paréntesis (return implícito)
- El argumento set me servirá para disparar la creación de un nuevo estado
- Podría colocar solo el status porque las otras dos propiedades son opcionales. En el momento inicial no tengo usuario ni user, le pongo undefined

```
import {create} from 'zustand'

interface AuthState{
    status: 'authenticated' | 'unauthenticated' | 'checking'
    token?: string
    user?:{
        name: string
        email: string
    }
}

export const useAuthStore = create<AuthState>()((set)=>({
    status: 'checking',
    token: undefined,
    user: undefined
}))
```

- Ahora tengo acceso a este useAuthStore desde cualquier lugar, no necesito envolver nada en providers ni similares
- Creo en components/**LoginPage.tsx**, lo renderizo en **App**

```
import React from 'react'
import { useAuthStore } from '../store/auth.store'
import { Text, View } from 'react-native'

const LoginPage = () => {
```

```
    const status = useAuthStore(state=> state.status)

  return (
    <View>
        <Text>{status}</Text>
    </View>
  )
}

export default LoginPage
```

- Para poner en marcha el proyecto uso **npm run start**
- **NOTA:** si da problemas con el HOME de JAVA colocar en gradle.properties el path dónde se encuentra JAVA (solución temporal)

```
org.gradle.java.home=C:\\Program Files\\Java\\jdk-17
```

Login

- Puedo renderizar condicionalmente un Loading

```
import React from 'react'
import { useAuthStore } from '../store/auth.store'
import { Text, View } from 'react-native'

const LoginPage = () => {

    const authStatus = useAuthStore(state=> state.status)

    if(authStatus=== 'checking'){
      return <View>
              <Text>Loading...</Text>
              </View>
    }

  return (
    <View>
        <Text>{authStatus}</Text>
    </View>
  )
}

export default LoginPage
```

- Los métodos en zustand pueden ser async
- Creo los métodos login y logout

- Suponiendo que el login se valida en algún backend, usamos el set para regresar un nuevo estado

```
import {create} from 'zustand'

interface AuthState{
    status: 'authenticated' | 'unauthenticated' | 'checking'
    token?: string
    user?:{
        name: string
        email: string
    }
    login : (email: string,password: string) => void
    logout: ()=> void
}

export const useAuthStore = create<AuthState>()((set)=>({
    status: 'checking',
    token: undefined,
    user: undefined,

    login: (email: string,password: string) => {
        set({
            status: 'authenticated',
            token: 'abc123',
            user:{
                name: 'John Doe',
                email: 'john@example.com'
            }
        })
    },

    logout: ()=>{
        set({
            status: 'unauthenticated',
            token: undefined,
            user: undefined
        })
    }

}))
```

- No estoy usando el password (por ahora)
- Para tomar las funciones puedo usar desestructuración para extraer todo en una sola linea
- **Pero zustand recomienda hacerlo por separado**, ya que si no puede disparar rerenders innecesarios
- Coloco un useEffect con un setTimeOut para llamar al logout y quitar el Loading... al estar checking
- Renderizo condicionalmente el user.
    - Al tratarse de un objeto uso JSON.stringify con el user, el replacement en null y un espacio de 2

```
import React, { useEffect } from 'react'
import { useAuthStore } from '../store/auth.store'
```

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

const LoginPage = () => {

    const authStatus = useAuthStore(state=> state.status)
    const user = useAuthStore(state => state.user)
    const login = useAuthStore(state => state.login)
    const logout = useAuthStore(state => state.logout)

    useEffect(()=>{
      setTimeout(()=>{
        logout();
      }, 1500)
    }, [])

    if(authStatus=== 'checking'){
      return <View>
              <Text>Loading...</Text>
              </View>
    }

  return (
    <View>
        <Text>Login Page</Text>
        {(authStatus === 'authenticated')
        ? <Text>Autenticado como : {JSON.stringify(user, null, 2)}</Text>
        : <Text>No autenticado</Text>
        }
    </View>
  )
}

export default LoginPage
```

- Hago la misma condición para mostrar un botón de logout y login
- Como le paso argumentos al login lo llamo con un callback

```
return (
    <View>
        <Text>Login Page</Text>
        {(authStatus === 'authenticated')
        ? <Text>Autenticado como : {JSON.stringify(user, null, 2)}</Text>
        : <Text>No autenticado</Text>
        }

        {(authStatus === 'authenticated')
        ? <View><Button  title="logout" onPress={logout}/></View>
        : <View><Button  title="login" onPress={()=>login("mail@example.com",
"1234")}/></View>
      }
```

```
    </View>
  )
```

- Zustand es muy útil. Se verá más durante el curso

Peticiones HTTP - Axios

- Creo UsersPage en /components
- Usaremos la web reqres.in
- Primero lo haremos con fetch

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

const UsersPage = () => {

    useEffect(()=>{
        fetch("https://reqres.in/api/users?page=2")
        .then(resp=> resp.json())
        .then(data=> console.log(data))
    }, [])

  return (
    <View>
        <Text>Usuarios</Text>
        <Text>Nombre</Text>
        <Text>Email</Text>
    </View>
  )
}

export default UsersPage
```

- Ahora con axios!
- Por defecto la respuesta de axios es de tipo any
- Para tiparla podemos ir a la url, copiar la data y usar la aplicación de Paste JSON as code para sacar la interfaz

```
export interface ReqUserListResponse {
    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;
}
```

- Llamo a get con axios
- Tipándolo ahora obtengo las opciones disponibles de resp.data.

```
import React, { useEffect } from 'react'
import { Text, View } from 'react-native'
import axios from 'axios'
import { ReqUserListResponse } from '../interfaces/DataAxios'

const UsersPage = () => {

    useEffect(()=>{
      axios.get<ReqUserListResponse>("https://reqres.in/api/users?page=2")
      .then(resp=> console.log(resp.data))
    }, [])

  return (
    <View>
        <Text>Usuarios</Text>
        <Text>Nombre</Text>
        <Text>Email</Text>
    </View>
  )
}

export default UsersPage
```

- el código que no tiene que estar necesariamente en el componente normalmente se exporta a otro archivo
- Las peticiones http suelen ir en archivos aparte o en una función

```
import React, { useEffect } from 'react'
import { Text, View } from 'react-native'
import axios from 'axios'
import { ReqUserListResponse } from '../interfaces/DataAxios'
```

```
const loadUsers = async ()=>{
  try {
    const {data} = await axios.get<ReqUserListResponse>
("https://reqres.in/api/users") //axios regresa data por defecto
    return data.data //.data porque es dónde están los Users[]

  } catch (error) {
    console.log(error)
    return
  }
}

const UsersPage = () => {


    useEffect(()=>{
     loadUsers().then(users=>console.log(users))
    }, [])

  return (
    <View>
        <Text>Usuarios</Text>
        <Text>Nombre</Text>
        <Text>Email</Text>
    </View>
  )
}

export default UsersPage
```

Mostrar usuarios

- Lo haremos con un useState, podríamos hacerlo con zustand
- Puedo tipar loadUsers como una promesa que devuelve un arreglo de User, retorno un arreglo vacío en el catch
- Lo mismo puedo tipar el useState con User[]
- Cuando tengo un argumento (o varios) que es pasado a la función que llama el callback, puedo poner solo la función (sin invocarla)
- Ahora ya tengo los users en una pieza de state
- En React Native usaríamos una FlatList
- Hago un map de los users del state, **le paso el key que debe de ser un string**, renderizo en un Text el nombre en **renderItems**
- Por supuesto podría crear otro componente para renderizar en renderItems con last_name, avatar, etc
- Renderizo UsersPage en App

```
import React, { useEffect, useState } from 'react'
import { FlatList, Text, View } from 'react-native'
import axios from 'axios'
import { ReqUserListResponse, User } from '../interfaces/DataAxios'
```

```
const loadUsers = async (): Promise<User[]>=>{
  try {
    const {data} = await axios.get<ReqUserListResponse>
("https://reqres.in/api/users") //axios regresa data por defecto
    return data.data //.data porque es dónde están los Users[]

  } catch (error) {
    console.log(error)
    return []
  }
}

const UsersPage = () => {

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

    useEffect(()=>{
     loadUsers().then(setUsers) //le paso el arreglo de users al state
    }, [])

  return (
    <View>
      <FlatList
        data= {users.map(({email, first_name})=>({
            key: email,
            first_name
        }))}
        renderItem={({item})=><Text>{item.first_name}</Text>}
      />
    </View>
  )
}

export default UsersPage
```

- Podemos crear una paginación con un par de botones
- En el caso de esta reqres.in, solo hay hasta página 2, la página 3 regresa 0 resultados
- Para acceder a la página 2 sólo hay que añadir a la url "/users?page=2"
- Podemos hacerlo a través de la url o el objeto de configuración de axios
- Puedo decir que recibo la página como parámetro de loadUsers y si no recibo nada es 1 por defecto
- Le paso el valor page al objeto params: page
- Uso useRef para saber en que página me encuentro, la inicio en 1
- Le paso el ref.current a loadUsers
- Para llamar a la siguiente página creo la función nextPage y prevoiusPage
- En nextPage, si el arreglo.length es mayor que cero se lo paso al state. Si no que vuelva a la página anterior
- En el previousPage establezco la condición para que si es la página 1 no pueda seguir descendiendo

```
import React, { useEffect, useRef, useState } from 'react'
import { Button, FlatList, Text, View } from 'react-native'
import axios from 'axios'
import { ReqUserListResponse, User } from '../interfaces/DataAxios'


const loadUsers = async (page: number = 1): Promise<User[]>=>{
  try {
    const {data} = await axios.get<ReqUserListResponse>
("https://reqres.in/api/users",{
      params:{
        page: page
      }
    })
    return data.data

  } catch (error) {
    console.log(error)
    return []
  }
}

const UsersPage = () => {

  const [users, setUsers] = useState<User[]>([])
  const currentPageRef = useRef(1)

  const nextPage = async () =>{
    currentPageRef.current ++;

    const users = await loadUsers(currentPageRef.current)
    if(users.length > 0){
      setUsers(users)
    }else{
      currentPageRef.current --;
    }
  }

  const previousPage = async () =>{
    if(currentPageRef.current < 1) return

    currentPageRef.current --;
    const users = await loadUsers(currentPageRef.current)
    setUsers(users)
  }

    useEffect(()=>{
     loadUsers(currentPageRef.current).then(setUsers)
    }, [])

  return (
    <View>
      <FlatList
```

```
          data= {users.map(({email, first_name})=>({
              key: email,
              first_name
          }))}
          renderItem={({item})=><Text>{item.first_name}</Text>}
        />

        <Button title="Previous" onPress={previousPage} />
        <Button title="Next" onPress={nextPage} />

      </View>
    )
}

export default UsersPage
```

Custom Hook - useUsers

- Cuando hay tanto código hay que modularizar para que sea más legible
- Muevo todo el código encima del return de UsersPage a useUsers
- Hago las importaciones necesarias, retorno las funciones nextPage, previousPage y users

```
import React from 'react'
import { useRef, useState, useEffect } from 'react'
import { ReqUserListResponse, User } from '../interfaces/DataAxios'
import axios from 'axios'

const loadUsers = async (page: number = 1): Promise<User[]>=>{
    try {
      const {data} = await axios.get<ReqUserListResponse>
("https://reqres.in/api/users",{
        params:{
          page: page
        }
      })
      return data.data

    } catch (error) {
      console.log(error)
      return []
    }
  }

const useUsers = () => {
    const [users, setUsers] = useState<User[]>([])
    const currentPageRef = useRef(1)

  const nextPage = async () =>{
    currentPageRef.current ++;
```

```
    const users = await loadUsers(currentPageRef.current)
    if(users.length > 0){
      setUsers(users)
    }else{
      currentPageRef.current --;
    }
  }

  const previousPage = async () =>{
    if(currentPageRef.current < 1) return

    currentPageRef.current --;
    const users = await loadUsers(currentPageRef.current)
    setUsers(users)

  }

    useEffect(()=>{
     loadUsers(currentPageRef.current).then(setUsers)
    }, [])

    return {
        nextPage,
        previousPage,
        users
    }
}

export default useUsers
```

- Ahora solo tengo que desestructurar la info del custom hook en UsersPage

```
const {nextPage, previousPage, users} = useUsers()
```

---

02 REACT NATIVE - COUNTER APP

- Usaremos Material 3
- Cambio la declaración de function App por una función de flecha y lo exporto en lugar de usar la exportación por defecto

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


export const App= (): React.JSX.Element=> {

  return(
    <View>
```

```
      <Text>Hola Mundo</Text>
    </View>
   )

}
```

- Debo cambiar también la importación de App en index.js y colocarlo entre llaves

```
/**
 * @format
 */

import {AppRegistry} from 'react-native';
import {App} from './App';
import {name as appName} from './app.json';

AppRegistry.registerComponent(appName, () => App);
```

Explicación de archivos y directorios

- .watchmanconfig (no se suele necesitar modificarlo)
- app.json es info básica de la app, es usado en otros directorios, pone la info en ios y android
- babel.config.js permite escribir javascript moderno
- Gemfile es el archivo de configuración de Ruby
- index.js es el main, el archivo de inicio. Cuando se lance todo la app empezará por aquí
- metro.config.js es raro que se tenga que modificar
- En las carpetas android o ios tengo la aplicación en si

Crear pantallas independientes

- En lugar del View uso el SafeAreaView para que se renderice correctamente con el notch de ios
- En src/**presentation/screens** creo el HelloWorldScreen.tsx y lo renderizo en App.tsx

```
import React from 'react';
import { SafeAreaView, Text} from 'react-native';
import HelloWorldScreen from './src/presentation/HelloWorldScreen';


export const App= (): React.JSX.Element=> {

  return(
    <SafeAreaView>
      <HelloWorldScreen />
    </SafeAreaView>
   )
}
```

- Para los estilos creo el snippet con la extensión Easy Snippets, uso **stless**

```
// @prefix stless
// @description
/* eslint-disable */


const styles = StyleSheet.create({
  container: {
    flex: 1,
    justifyContent: 'center'
  }
})
```

- Para agregar estilos solo tengo que usar la propiedad style y añadir con notación de punto los estilos
- Si el flex: 1 no funciona, mirar que el componente padre (como es en este caso el SafeAreaView en App no lo esté restringiendo)

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

const HelloWorldScreen = () => {
  return (
    <View style={styles.container} >
        <Text style={styles.title}>Hello World</Text>
    </View>
  )
}

const styles = StyleSheet.create({
  container: {
    flex: 1,
    justifyContent: 'center',
    alignItems: 'center'
  },

  title:{
    fontSize: 45,
    textAlign: 'center',
    color: 'black'
  }
})

export default HelloWorldScreen
```

Propiedades

- Para usar propiedades en los componentes en Typescript hay que usar una interfaz
- Puedo hacer la propiedad opcional con ?
- La desestructuro y añado el tipado de la interfaz
- Puedo ponerle un valor por defecto

```
interface Props{
  name?: string
}

const HelloWorldScreen = ({name= "World"}: Props) => {
  return (
    <View style={styles.container} >
        <Text style={styles.title}>Hello {name}</Text>
    </View>
  )
}
```

- Ahora puedo pasarle la propiedad name al componente

```
export const App= (): React.JSX.Element=> {

  return(
    <SafeAreaView>
      <HelloWorldScreen name='Ismael Berón' />
    </SafeAreaView>
   )
}
```

- Los componentes nativos tienen sus propias propiedades
- Por ejemplo, Text tiene **numberOfLines** que son el número de lineas que quiero y si no cabe corta el texto con una elipsis (...)
    - Puedo controlar esta elipsis con **ellipsizeMode**,
    - **tail** es por defecto, quita el texto al final, **middle** hace el corte en medio, **head** lo quita del principio, **clip**lo acomoda como puede

---

Crear un contador

- Creo el componente CounterScreen en **src/presentation/screens** con un View y un Text
    - Usar el snippet personalizado **rncc** hecho con *Easy Snippets*

```
// @prefix rncc
// @description
/* eslint-disable */


import React from 'react'
```

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

export const CounterScreen = () => {
  return (
    <View>
        <Text></Text>
    </View>
  )
}
```

- Añado unos estilos y coloco unos botones. Usaremos Button para este caso pero no es el que usaremos normalmente
- Contador

```
export const CounterScreen = () => {

  const [counter, setCounter] = useState(10)


  const increment = ()=> {
    setCounter(counter +1)
  }

  const decrement = ()=> {
    setCounter(counter -1)
  }

  return (
    <View style={styles.container}>
        <Text style={styles.title}>Counter: {counter} </Text>

        <Button title="increment" onPress={()=>increment()}/>
        <Button title="decrement" onPress={()=> decrement()}/>
    </View>
  )
}
```

- Puedo usar el Pressable que es más personalizable.
- Aparecen sin estilos. Creo button en styles, y buttonText para el texto del botón
- Puedo usar la prop onLongPress para resetear el contador usando el setCounter

```
import React, { useState } from 'react'
import { Button, Pressable, StyleSheet, Text, View} from 'react-native'

export const CounterScreen = () => {

  const [counter, setCounter] = useState(10)
```

```
  const increment = ()=> {
    setCounter(counter +1)
  }

  const decrement = ()=> {
    setCounter(counter -1)
  }

  return (
    <View style={styles.container}>
        <Text style={styles.title}>Counter: {counter}</Text>

        <Pressable style={styles.button} onPress={increment}>
          <Text style={styles.buttonText}>Increment</Text>
        </Pressable>

        <Pressable style={styles.button} onPress={decrement} onLongPress=
{()=>setCounter(0)}>
          <Text style={styles.buttonText}>Decrement</Text>
        </Pressable>
    </View>
  )
}

const styles = StyleSheet.create({
  container: {
    flex:1,
    justifyContent: 'center',
    alignItems: 'center',
  },
  title:{
    fontSize: 45,
    color:'black',
    fontWeight: '300'
  },
  button:{
    backgroundColor: 'purple',
    paddingHorizontal: 20,
    paddingVertical: 10,
    borderRadius: 10,
    marginVertical: 10
  },
  buttonText:{
    fontSize: 25,
    color: 'white'
  }
})
```

- Para que aparezca el color picker habilitar Editor:color decorators en VSCode
- Puedo usar en el style de Pressable un callback del que puedo desestructurar **pressed**, devuelve un arreglo.

- Le paso los estilos del botón y puedo usar pressed (boolean por defecto en true) para aplicar estilos condicionalmente cuando se apriete el botón

```
  return (
    <View style={styles.container}>
        <Text style={styles.title}>Counter: {counter}</Text>

        <Pressable
        style={({pressed})=>[
            styles.button,
            pressed && styles.buttonPressed]}
            onPress={increment}>
             <Text style={styles.buttonText}>Increment</Text>
        </Pressable>
        <Pressable
        style={({pressed})=>[
            styles.button,
            pressed && styles.buttonPressed]}
            onPress={decrement} onLongPress={()=>setCounter(0)}>
          <Text style={styles.buttonText}>Decrement</Text>
        </Pressable>
    </View>
  )
```

- Es más práctico crear un componente para tener personalizado el botón, que sea customizable
- Para renderizar estilos condicionalmente si estamos en android o ios uso **Platform** de 'react-native'
- Ejemplo:

```
 buttonPressed:{
    backgroundColor: Platform.OS === 'android'? 'rgb(126, 0, 78)': 'white'
  }
```

- De esta manera tengo el estilo aplicado de manera independiente por plataforma

---

Componente Personalizad

- Conviene tener una biblioteca de componentes personalizados (y personalizables) para no estar reinventando la rueda
- Creemos este botón en /components/**shared** con un archivo de barril **index.ts**
- Dentro de share creo PrimaryButton.tsx con el snippet **rafc** o el snippet personalizado **rncc**
- Lo coloco en el archivo de barril

```
export * from './PrimaryButton'
```

- Creo la interfaz para las props, hago el longPress opcional

- Coloco las props donde corresponden
- Copio los estilos que había definido y se los coloco al componente

```
import React from 'react'
import { GestureResponderEvent, Platform, Pressable, StyleSheet, Text, View } from
'react-native'

interface Props{
  label: string
  onPress: (event: GestureResponderEvent)=> void
  onLongPress?: (event: GestureResponderEvent)=> void
}

export const PrimaryButton = ({label, onPress, onLongPress}: Props) => {
  return (
    <View>
        <Pressable style={({pressed})=>[
          styles.button,
          pressed && styles.buttonPressed
        ]} onPress={onPress} onLongPress={onLongPress} >
          <Text style={styles.buttonText} >{label}</Text>
        </Pressable>
    </View>
  )
}

const styles = StyleSheet.create({
  container: {
    flex: 1,
    justifyContent: 'center'
  },
  button:{
    backgroundColor: 'purple',
    paddingHorizontal: 20,
    paddingVertical: 10,
    borderRadius: 10,
    marginVertical: 10
  },
  buttonText:{
    fontSize: 25,
    color: 'white'
  },
  buttonPressed:{
    backgroundColor: Platform.OS === 'android' ?'rgb(126, 0, 78)': 'white'
  }

})
```

- Añado el componente a CounterScreen

```
export const CounterScreen = () => {

  const [counter, setCounter] = useState(10)


  const increment = ()=> {
    setCounter(counter +1)
  }

  const decrement = ()=> {
    setCounter(counter -1)
  }

  return (
    <View style={styles.container}>
        <Text style={styles.title}>Counter: {counter}</Text>
      <PrimaryButton label="increment" onPress={increment} onLongPress=
{()=>setCounter(0)} />
      <PrimaryButton label="decrement" onPress={decrement} />
    </View>
  )
}
```

React Native Paper - Instalación

- Instalamos

> npm i react-native-paper npm i react-native-safe-area-context

- Hay que usar el PaperProvider como provider en el punto más alto de la aplicación que va a contener los componentes

```
import React from 'react';
import { CounterScreen } from './src/presentation/screens/CounterScreen';
import {PaperProvider} from 'react-native-paper'

export const App=(): React.JSX.Element=>{

  return(
      <PaperProvider>
         <CounterScreen />
      </PaperProvider>
   )
}
```

- Si tienes un gestor de estado tipo Redux-toolkit, colocar el PaperProvider dentro del gestor de estado y dentro del PaperProvider el App
- React Native Paper se puede **customizar**

- Puedo mirar la documentación para evr cómo usar un Button de React Native Paper
- Uso el mode contained
- Los componentes de React Native Paper siguen los standares de React Native

```
import React, { useState } from 'react'
import { StyleSheet, Text, View} from 'react-native'
import { PrimaryButton } from '../../components/shared'
import {Button} from 'react-native-paper'

export const CounterScreen = () => {

  const [counter, setCounter] = useState(10)


  const increment = ()=> {
    setCounter(counter +1)
  }

  const decrement = ()=> {
    setCounter(counter -1)
  }

  return (
    <View style={styles.container}>
        <Text style={styles.title}>Counter: {counter}</Text>
      <PrimaryButton label="increment" onPress={increment} onLongPress=
{()=>setCounter(0)} />
      <Button onPress={decrement} mode='contained'>Decrementar</Button>
    </View>
  )
}

const styles = StyleSheet.create({
  container: {
    flex:1,
    justifyContent: 'center',
    alignItems: 'center',
  },
  title:{
    fontSize: 45,
    color:'black',
    fontWeight: '300'
  }
})
```

Floating Action Button FAB

- Para crear estilos de manera global, creo una carpeta en presentation/**theme**

```
import { StyleSheet } from "react-native"

export const Globalstyles = StyleSheet.create({
    centerContainer: {
        flex: 1,
        justifyContent: 'center',
        alignItems: 'center'
    },
    title: {
        fontSize: 45,
        fontWeight: '300',
        color: 'black'
    }
})
```

- Ahora solo debo importarlo y usar GlobalStyles.centerContainer
- Para crear un FAB creo un nuevo componente en /**shared**, lo exporto en el archivo de barril
- Mirar la documentación de **React Native Paper**!!

```
import React from 'react'
import { StyleSheet } from 'react-native'
import {FAB} from 'react-native-paper'

export const FloatingAB = () => {
  return (
    <FAB
    style={styles.fab}
    onPress={()=>console.log("FAB!")}
    label="FAB"
    />
  )
}

const styles = StyleSheet.create({
  fab: {
    position: 'absolute',
    margin: 16,
    right: 0,
    bottom: Platform.OS === 'android'? 15: 0
  }
})
```

- Puedo también usar GlobalStyles para los estilos del fab

Configurar iconos

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

- En *android/app/build.gradle* añadir

```
project.ext.vectoricons =[
  iconFontNames: ['Ionicons.ttf']
]

apply from: file ("../../node_modules/react-native-vector-icons/fonts.gradle")
```

- Ahora ya puedo usar iconos. Importo Icon de 'react-native-vector-icons/Ionicons'

```
<Icon name="accessibility-outline" size={35} />
```

- Icon tiene un montón de propiedades
- En el FAB puedo colocar el icon de varias maneras, incluso puedo mandar un funtional Component
- Pero quiero hacerlo de otra forma
- Hay que configurar Paper

Configurar Iconos Globles

- En **PaperProvider** uso la propiedad **settings**

```
import React from 'react';
import { CounterScreen } from './src/presentation/screens/CounterScreen';
import {PaperProvider} from 'react-native-paper'
import IonIcon from 'react-native-vector-icons/Ionicons'

export const App=(): React.JSX.Element=>{

  return(
      <PaperProvider
      settings={({
         icon: (props)=> <IonIcon {...props} />
      })}
      >
         <CounterScreen />
      </PaperProvider>
   )
}
```

- Ahora solo tengo que añadir el icono que quiero a la propiedad icon. Ya no necesita el label

```
export const FloatingAB = () => {
  return (
    <FAB
```

```
      style={styles.fab}
      onPress={()=>console.log("FAB!")}
      icon='add'
      />
    )
  }
```

03 REACT NATIVE - FLEX

Fundamentos

Tenemos: - **Box Object Model**: alto, ancho. margin, border - **Position**: absolute, relative, top, left, right, bottom - **Flexbox**: dirección, ubicación, alineamiento, estirar, encoger, proporciones

Box Object Model

- Es el width, el height, el padding, el border, y el margin
- Tenemos
    - margin
    - marginLeft
    - marginRight
    - marginTop
    - marginBottom
    - marginVertical
    - marginHorizontal
- Lo mismo con el padding
- Con el border es un poco diferente
    - borderWidth
    - borderLeftWidth
-
    - right, bottom y top
- Para el color uso borderColor: 'color'
- Siempre es el padre quien determina el tamaño
    - Si al padre le pongo un height de 30 y el fontSize del hijo es de 45, las letras se verán cortadas
    - Si el hijo tiene un tamaño relativo, será relativo al padre
- Con **flex:1** toma todo el tamaño posible
- Con el SafeAreaView en App, si coloco un flex:1 en el componente hijo **NO SE VE** porque el SafeAreaView no tiene un tamaño
- Esto se soluciona dándole un height al SafeAreaView

```
<SafeAreaView style={{height: 500}} >
```

- Si aplico un borderWidth **sin un padding** el border de la caja tapará las letras
- El padding hace una separación interna, el margin hace una separación con el padre

Height, width porcentual y dimensiones de la pantalla

- Los porcentajes son relativos al padre
- Los porcentajes se escriben como string width: '50%'
- Si quiero que la caja tome la mitad de la pantalla indistintamente de lo que mida el padre de ancho
- Puedo usar **Dimensions**

```
import {Dimensions} from 'react-native'

const windowWith = Dimensions.get('window').width
const windowHeight = Dimensions.get('window').height

//o puedo usar desestructuración

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

- Puedo tomar todas las propiedades de CSS que haya escrito en la StyleSheet **usando el operador spread**
- Las propiedades que escriba dentro del objeto se sobreescribirán

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

<View style={{
    ...styles.purplebox,
    width: width * 0.5   //así ocupa la mitad de la pantalla
}} >
```

Position

- El position puede ser absolute o relative
- La position por defecto es relative (al padre)
- Inclusive cuando es absolute es relative al padre
- Con **position relative** el bottom: 0 es en el mismo punto dónde se encuentra
- Si le pongo valores positivos al bottom va a empezar a subir.
    - Todo es relativo a la posición en la que se encuentra la caja
    - Si tiene otra caja no la empujará al moverlo, pero si al colocarlas inicialmente en su construcción
    - Se verá según el orden en el que fueron construidos (se solaparán)
    - Por defecto se colocan en columna

- Con **position absolute** el bottom: 0 es relativo al padre, es decir, iría abajo del todo al bottom del padre a la izquierda
    - Por defecto el left es 0.
    - Si tenemos dos elementos y el primero tiene el position en absolute, es como si no ocupara un espacio.
    - Es decir, que el segundo elemento se colocará encima
    - Puedo usar el absolute para colocar la caja donde quiera relativa al padre indistintamente usando top: 0, right: 0, etc

---

Flexbox en React Native

- Puedo usar varios Views y distribuir el espacio con flex:1, flex:2, flex:3, donde flex:1 = 1/6, flex:2 = 2/6, y flex:3 = 3/6
- Tenemos en **direcciones**: column, row, column-reverse y row-reverse
- En **justifyContent** tenemos: flex-start, flex-end, center, space-between, space-around, space-evenly
- En **alignItems** tenemos: stretch (por defecto, similar al auto), flex-start, flex-end, center, baseline (se usa poco)
- El eje de justifyContent y alignItems varia si estoy en row o column
- **alignSelf**: es un alineamiento que le damos a un objeto en particular sobreescribiendo el del padre. Son los mismos que alignItems
- **alignContent** funciona igual que el justifyContent pero en el eje vertical en el caso de estar en row
- **flexWrap**: limita los objetos al espacio disponible
- **flexBasis, Grow and Shrink**: como quieres que se estire, que se encoja
- **Row Gap, Column Gap and Gap**: añade espacios

Tips

- Si pongo alignItems en stretch (estando en row) y quiero que se estiren en su totalidad verticalmente, le quito el height al objeto
- Si le quito el width desaparecen
- Para que cada caja ocupe el máximo disponible de alto y ancho pongo flex: 1
- Con alignSelf (estando en row) si quiero que uno de los elementos esté en el bottom, puedo usar flex-end
- Con alignSelf sigue siendo vigente la dirección. Si cambio a row-reverse se invertirán
- Con flexWrap los objetos se alinearán en una fila o columna siguiente. Tengo 'wrap' y 'no-wrap'
    - Usándolo conjuntamente con el gap se crea un grid
    - Tengo gap, columnGap y rowGap

REACT NATIVE - NAVEGACION

Tipos de Navegación

- Usaremos el paquete React Navigation
- Tenemos:

    - **Stack Navigation**: las pantallas se apilan como en una baraja de cartas
    - **Drawer Navigation**: abre un menú lateral
    - **BottomTab Navigation**: tienes las opciones en el bottom
    - **MaterialTop Navigation**: tienes las opciones en el top
- Se pueden combinar. Tiene su ciencia!
- Con navigation.navigate navegamos entre las pantallas
- En Stack Navigation podemos deshacernos de las pantallas con **navigation.pop**, con **popToTop** volvemos a la primera, con **goBack** a la anterior

---

Instalaciones

- Creo los directorios en src
    - **presentation**: la capa de presentacion, lo que está cerca del usuario. Componentes, visualización y demás
        - Dentro de presentation creo **screens** para las pantallas, **theme** para los estilos globales con theme.ts, **routes** para las rutas y **components**
    - Dentro de screens creo una carpeta **home** con la **HomeScreen.tsx**
    - Creo otras dos carpetas dentro de screens llamada products con ProductScreen.tsx y ProductsScreen.tsx y tabs con Tab1Screen, 2 y 3
    - Creo también la carpeta about dentro de screens con la AboutScreen.tsx
    - Creo otra carpeta dentro de screens llamada profiles con ProfileScreen.tsx y settings con SettingsScreen.tsx

> npm i @react-navigation/native npm install react-native-gesture-handler npm i react-native-screens react-native-safe-area-context npm i @react-native-community/masked-view

- Usa import 'react-native-gesture-handler' en App.tsx **encima de todos los import**
- Lanza la app para ver si hay algún error
- Si da error con la ruta HOME de JAVA colocar la ruta en gradle.properties

```
org.gradle.java.home=C:\\Program Files\\Java\\jdk-17
```

- Si da error de React.jsx: type is invalid, cambiar App por una función de flecha, no exportarlo por default y cambiar la exportación en index.js

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


export const App=(): React.JSX.Element=> {


  return (
    <View>
      <Text>Hello world</Text>
```

```
    </View>
  );
}
```

- index.js

```
import {AppRegistry} from 'react-native';
import {App} from './App';
import {name as appName} from './app.json';

AppRegistry.registerComponent(appName, () => App);
```

React Navigation Stack

- Instalación

> npm i @react-navigation/stack npm install @react-native-masked-view/masked-view

- En presentation/routes creo el archivo stackNavigator.tsx
- Creo una navegación con el ejemplo de la documentación y le añado mis componentes

```
import { createStackNavigator } from '@react-navigation/stack';
import { HomeScreen } from '../screens/home/HomeScreen';
import { ProductsScreen } from '../screens/products/ProductsScreen';
import { SettingsScreen } from '../screens/settings/SettingsScreen';

const Stack = createStackNavigator();

export const StackNavigator =()=> {
  return (
    <Stack.Navigator>
      <Stack.Screen name="Home" component={HomeScreen} />
      <Stack.Screen name="Products" component={ProductsScreen} />
      <Stack.Screen name="Settings" component={SettingsScreen} />
    </Stack.Navigator>
  );
}
```

- Coloco el stackNavigator en App dentro de NavigationContainer

```
import 'react-native-gesture-handler';
import React from 'react';
import { StackNavigator } from './src/presentation/routes/StackNavigator';
import { NavigationContainer } from '@react-navigation/native';


export const App=(): React.JSX.Element=> {
```

```
  return (
    <NavigationContainer>
        <StackNavigator />
    </NavigationContainer>
  );
}
```

Navegar a otras pantallas

- Si hago un console.log de las props de HomeScreen hay muchas debido a que estoy dentro del NavigationContainer

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

export const HomeScreen = (props: any) => {
  console.log(props)

  return (
    <View>
        <Text>HomeScreen</Text>
    </View>
  )
}
```

- Tengo el **navigation**, **navigate**, el **pop** para cerrar la pantalla actual y regresar a la otra
- Tipar estas props puede ser un poco retador. Tenemos **hooks** que nos permiten **acceder a esa info**
- Creo un botón que me llevará a productos
- Le añado unos estilos desde los estilos globales

```
import {StyleSheet} from 'react-native'

export const globalStyles = StyleSheet.create({
  container:{
    flex:1,
    padding: 20,
    justifyContent: 'center'
  },

  primaryButton:{
    backgroundColor: 'orange',
    borderRadius: 5,
    padding: 10,
    width: '100%',
    alignItems: 'center'
  },
```

```
  buttonText:{
    color: 'black',
    fontSize: 18
  }
})
```

- Luego crearemos este botón personalizado
- Hagamos la navegación. Sale más fácil hacerlo con el hook **useNavigation**
- Tipo **as never** el componente al que navego, luego lo arreglaremos

```
import React from 'react'
import { Pressable, Text, View } from 'react-native'
import { globalStyles } from '../../theme/theme'
import { useNavigation } from '@react-navigation/native'

export const HomeScreen = () => {

  const navigation = useNavigation()

  return (
    <View style={globalStyles.container} >
        <Pressable
        onPress={()=> navigation.navigate("Products" as never)}
        style={globalStyles.primaryButton} >
            <Text style={globalStyles.buttonText} >Productos</Text>
        </Pressable>
    </View>
  )
}
```

- Podemos personalizar el Home que se ve arriba de la pantalla, más adelante
- Creo el componente del botón personalizado
- Dentro de components creo la carpeta /**shared** y dentro PrimaryButton.tsx

```
import React from 'react'
import { Pressable, Text, View } from 'react-native'
import { globalStyles } from '../../theme/theme'


interface Props{
    onPress: ()=> void
    label: string
}

export const PrimaryButton  = ({label, onPress}: Props) => {


  return (
    <Pressable
```

```
    onPress={onPress}
    style={globalStyles.primaryButton} >
        <Text style={globalStyles.buttonText}>{label}</Text>
    </Pressable>
  )
}
```

- En HomeScreen añado el componente y creo otro para Settings

```
import React from 'react'
import { Pressable, Text, View } from 'react-native'
import { globalStyles } from '../../theme/theme'
import { useNavigation } from '@react-navigation/native'
import { PrimaryButton } from '../../components/shared/PrimaryButton'

export const HomeScreen = () => {

  const navigation = useNavigation()

  return (
    <View style={globalStyles.container} >
      <PrimaryButton label={"Products"} onPress={()=>
navigation.navigate("Products" as never)} />
      <PrimaryButton label={"Settings"} onPress={()=>
navigation.navigate("Settings" as never)} />
    </View>
  )
}
```

Estilizando Stack Navigator

- Configuraciones que podemos hacer al Stack Navigator
- Con screenOptions tienes un montón de propiedades

```
import { createStackNavigator } from '@react-navigation/stack';
import { HomeScreen } from '../screens/home/HomeScreen';
import { ProductsScreen } from '../screens/products/ProductsScreen';
import { ProductScreen } from '../screens/products/ProductScreen';
import { SettingsScreen } from '../screens/settings/SettingsScreen';

const Stack = createStackNavigator();

export const StackNavigator=()=> {
  return (
    <Stack.Navigator screenOptions={{
      headerShown: true,  //false no muestra el header
      headerStyle:{
        elevation: 0, //0 elimina la linea divisoria del header
```

```
        shadowColor: 'transparent', //hace que desaparezca en ios

      }
    }}>
      <Stack.Screen name="Home" component={HomeScreen} />
      <Stack.Screen name="Products" component={ProductsScreen} />
      <Stack.Screen name="Product" component={ProductScreen} />
      <Stack.Screen name="Settings" component={SettingsScreen} />
    </Stack.Navigator>
  );
}
```

- Stack.Screen también tiene la propiedad **options**

FlatList - Pantalla de productos

- En data le añado el arreglo, en renderItem puedo desestructurar el item para acceder al objeto
- Si extraigo la data (en lugar de desestructurar item) puedo acceder a los separators (mirar documentación)
- Puedo renderizar en cada botón el name del arreglo
- Notar que el callback de renderItem está entre paréntesis y no llaves para hacer **implícito el return**
- Coloco un texto Ajustes en el bottom y otro botón

```
import React from 'react'
import { Text, View } from 'react-native'
import { FlatList } from 'react-native-gesture-handler'
import { globalStyles } from '../../theme/theme'
import { PrimaryButton } from '../../components/shared/PrimaryButton'
import { useNavigation } from '@react-navigation/native'

const products=[
  {id:1, name: "Spaceship"},
  {id:2, name: "Car"},
  {id:3, name: "Plane"},
  {id:4, name: "MotorCycle"},
  {id:5, name: "Bike"},
]

export const ProductsScreen = () => {

  const navigation = useNavigation()
  return (
    <View style={globalStyles.container}>
      <Text>Productos</Text>
        <FlatList
        data={products}
        renderItem={({item})=>(
        <PrimaryButton
          label={item.name}
          onPress={()=>{navigation.navigate("Product" as never) } } />)}
```

```
        />

        <Text style={{marginBottom: 10, fontSize:30}}>Ajustes</Text>
        <PrimaryButton label={"Ajustes"} onPress=
{()=>navigation.navigate("Settings" as never)} />
    </View>
  )
}
```

Enviar argumentos entre pantallas

- Con el **navigation.navigate**, los argumentos que siguen al componente son los que le va a mandar a esa pantalla
- Los argumentos hay que tiparlos previamente

```
onPress= {()=> {navigation.navigate("Product" as never, {id: item.id ,name:
item.name})}}
```

- Para evitar el **as never** en el tipado me creo un tipo **RootStackParams** en el Navigator
- Le paso el RootStackParams al **createStackNavigator**

```
import { createStackNavigator } from '@react-navigation/stack';
import { HomeScreen } from '../screens/home/HomeScreen';
import { ProductsScreen } from '../screens/products/ProductsScreen';
import { SettingsScreen } from '../screens/settings/SettingsScreen';
import { ProductScreen } from '../screens/products/ProductScreen';


export type RootstackParams ={
  Home: undefined,
  Product: {id: number, name: string},
  Products: undefined,
  Settings:undefined,
}
const Stack = createStackNavigator<RootstackParams>();

export const StackNavigator=()=> {
  return (
    <Stack.Navigator screenOptions={{
      headerShown: true,  //false no muestra el header
      headerStyle:{
        elevation: 0, //0 elimina la linea divisoria del header
        shadowColor: 'transparent', //hace que desaparezca en ios

      }
    }}>
      <Stack.Screen name="Home" component={HomeScreen} />
      <Stack.Screen name="Products" component={ProductsScreen} />
```

```
      <Stack.Screen name="Product" component={ProductScreen} />
      <Stack.Screen name="Settings" component={SettingsScreen} />
    </Stack.Navigator>
  );
}
```

- En ProductsScreen tengo que decirle al useNavigation **cuáles son las properties disponibles**
- Para eso **le paso el genérico NavigationProp y a este el RootStackParams**
- Puedo **quitar el as never**
- Le puedo poner type en la importación cuando es un tipo, ayuda a la transpilación ya que no se convierte en código real

```
import React from 'react'
import { Text, View } from 'react-native'
import { FlatList } from 'react-native-gesture-handler'
import { globalStyles } from '../../theme/theme'
import { PrimaryButton } from '../../components/shared/PrimaryButton'
import { NavigationProp, useNavigation } from '@react-navigation/native'
import { type RootstackParams } from '../../routes/StackNavigator'

const products=[
  {id:1, name: "Spaceship"},
  {id:2, name: "Car"},
  {id:3, name: "Plane"},
  {id:4, name: "MotorCycle"},
  {id:5, name: "Bike"},
]

export const ProductsScreen = () => {

  const navigation = useNavigation<NavigationProp<RootstackParams>>() //aqui!

  return (
    <View style={globalStyles.container}>
      <Text>Productos</Text>
        <FlatList
        data={products}
        renderItem={({item})=>(
        <PrimaryButton
          label={item.name}
          onPress={()=>{navigation.navigate("Product", {id: item.id, name:
item.name}) } } />)}
        />

        <Text style={{marginBottom: 10, fontSize:30}}>Ajustes</Text>
        <PrimaryButton label={"Ajustes"} onPress=
{()=>navigation.navigate("Settings")} />
    </View>
  )
}
```

- También podemos quitar el as never en Home haciendo el mismo proceso

```
import React from 'react'
import { Pressable, Text, View } from 'react-native'
import { globalStyles } from '../../theme/theme'
import { type NavigationProp, useNavigation } from '@react-navigation/native'
import { PrimaryButton } from '../../components/shared/PrimaryButton'
import type {RootstackParams } from '../../routes/StackNavigator'

export const HomeScreen = () => {

  const navigation = useNavigation<NavigationProp<RootstackParams>>()

  return (
    <View style={globalStyles.container} >
      <PrimaryButton label={"Products"} onPress={()=>
navigation.navigate("Products")} />
      <PrimaryButton label={"Settings"} onPress={()=>
navigation.navigate("Settings")} />
    </View>
  )
}
```

- **Cómo obtenemos los argumentos que pasamos entre pantallas?**
- Vienen en las **props.params**, pero puede ser que estés en una situación en la cual no puedas extraerlo de las props
- Para ello usaremos **useRoute**
- Para tiparlo uso como genérico el RouteProp, y dentro cómo genérico le paso el RootstackParams, del componente Product
- Sin el tipado quedaría useRoute().params
- Con el tipado ya puedo desestructurar el id y el name y pasarselo al Text
- Para colocar el nombre del producto en el header usaré un **useEffect**
- Accedo al header a través del **useNavigation** con el método **setOptions y la propiedad title**

```
import { type RouteProp, useRoute, useNavigation } from '@react-navigation/native'
import React, { useEffect } from 'react'
import { Text, View } from 'react-native'
import { type RootstackParams } from '../../routes/StackNavigator'
import { globalStyles } from '../../theme/theme'

export const ProductScreen = () => {

  const {id, name} = useRoute<RouteProp<RootstackParams, 'Product'>>().params
//los argumentos estan en .params
  const navigation = useNavigation()

  useEffect(()=>{
    navigation.setOptions({
      title: name
```

```
    })
  },[])

  return (
    <View style={globalStyles.container} >
        <Text style={{fontSize: 20, textAlign: 'center', marginTop: 20}}>{id} -
{name} </Text>
    </View>
  )
}
```

Stack PopToPop

- Para volver al Home ya no tenemos el popToTop directamente del navigator. Tenemos que usar el dispatch
- Tenemos ciertas acciones específicas que estan en el StackActions. Así nos ahorramos tipar el useNavigation
- También puedo usar el .goBack() para volver atrás

```
import React from 'react'
import { Text, View } from 'react-native'
import { globalStyles } from '../../theme/theme'
import { PrimaryButton } from '../../components/shared/PrimaryButton'
import { StackActions, useNavigation } from '@react-navigation/native'

export const SettingsScreen = () => {

  const navigator = useNavigation()

  return (
    <View style={globalStyles.container} >
        <Text style={{marginBottom: 10}} >SettingsScreen</Text>

        <PrimaryButton
        label="Regresar"
        onPress={()=>navigator.goBack()} />

        <PrimaryButton
        label="Regresar"
        onPress={()=>navigator.dispatch(StackActions.popToTop())} />
    </View>
  )
}
```

Drawer Navigation

- Es el menu lateral

- Se puede personalizar, pero el comportamiento que viene predefinido es el de mostrar enlaces
- Configuración

> npm i @react-navigation/drawer react-native-reanimated

- Creo el componente en /routes/SideMenuNavigator
- Coloco el StackNavigator

```
import React from 'react'
import {createDrawerNavigator} from '@react-navigation/drawer'
import { StackNavigator } from './StackNavigator'
import { ProfileScreen } from '../screens/profiles/ProfileScreen'

const Drawer = createDrawerNavigator()

const SideMenuNavigator = () => {

    return (
    <Drawer.Navigator>
        <Drawer.Screen name="StackNavigator" component={StackNavigator} />
        <Drawer.Screen name="Profile" component={ProfileScreen} />
    </Drawer.Navigator>
  )
}

export default SideMenuNavigator
```

- Dentro del NavigationContainer, en lugar del StackNavigator muestor el SideMenuNavigator

```
import 'react-native-gesture-handler';
import React from 'react';
import { NavigationContainer } from '@react-navigation/native';
import SideMenuNavigator from './src/presentation/routes/SideMenuNavigator';


export const App=(): React.JSX.Element=> {

  return (
    <NavigationContainer>
        <SideMenuNavigator />
    </NavigationContainer>
  );
}
```

- A veces da problemas el BUILD con el reanimated, prueba de mover el proyecto a un Path más corto y un nombre de carpeta mas corto
- También puede ayudar este comando para que borre la info preprocesada
- Para resolver el error de can´t read isConfigured of undefined ir a babel.config.js y añadir

> npx react-native start --resetCache

```
module.exports = {
  presets: ['module:@react-native/babel-preset'],
  plugins: ['react-native-reanimated/plugin']
};
```

Toggle Drawer - Mostrar/Ocultar

- Quiero simular el botón del Drawer debajo del Header, donde dice Home. No lo quiero en el Header
- Voy a HomeScreen, creo un botón Menu con un useEffect y usando el **navigation.setOptions** lo coloco a la izquierda
- Para hacer que tenga el comportamiento del Drawer uso el onPress
- Para tiparlo en el useNavigation puede ser un poco complicado, por eso uso el **dispatch con el DrawerActions**
- Notar que el callback de headerLeft no está entre llaves si no **entre paréntesis haciendo el return implícito**

```
import React, { useEffect } from 'react'
import { Pressable, Text, View } from 'react-native'
import { globalStyles } from '../../theme/theme'
import { type NavigationProp, useNavigation, DrawerActions } from '@react-
navigation/native'
import { PrimaryButton } from '../../components/shared/PrimaryButton'
import type {RootstackParams } from '../../routes/StackNavigator'

export const HomeScreen = () => {

  const navigation = useNavigation<NavigationProp<RootstackParams>>()

  useEffect(() => {

    navigation.setOptions({
      headerLeft:()=>(
        <Pressable onPress={()=>navigation.dispatch(DrawerActions.toggleDrawer)} >
        <Text>Menu</Text>
        </Pressable>
      )
    })

  }, [])


  return (
    <View style={globalStyles.container} >
      <PrimaryButton label={"Products"} onPress={()=>
navigation.navigate("Products")} />
      <PrimaryButton label={"Settings"} onPress={()=>
```

```
navigation.navigate("Settings")} />
    </View>
  )
}
```

- En ios, al desplegar el drawer, empuja el contenido, en Android se coloca por encima

---

Drawer Personalizado

- En **Drawer.Navigator** tengo un montón de propiedades. Una de ellas es **screenOptions**
- Con drawerType en slide empuja los elementos al desplegar el menú lateral

```
import React from 'react'
import {createDrawerNavigator} from '@react-navigation/drawer'
import { StackNavigator } from './StackNavigator'
import { ProfileScreen } from '../screens/profiles/ProfileScreen'

const Drawer = createDrawerNavigator()

const SideMenuNavigator = () => {

    return (
    <Drawer.Navigator screenOptions={{
      headerShown: false,
      drawerType: 'slide',
      drawerActiveBackgroundColor: 'orange',
      drawerActiveTintColor: 'white',
      drawerInactiveTintColor: 'black',
      drawerItemStyle:{
        borderRadius: 100,
        paddingHorizontal: 20
      }
      }} >
        <Drawer.Screen name="MenuItem1" component={StackNavigator} />
        <Drawer.Screen name="Profile" component={ProfileScreen} />
    </Drawer.Navigator>
  )
}

export default SideMenuNavigator
```

- Cuando necesito un nivel más alto de personalización, puedo crear un nuevo componente y colocarlo en la prop drawerContent
- Le paso las props DrawerContentComponentProps
- Dentro creo un DrawerContentScrollview y le meto un View con estilos
- A DrawerItemList le esparzo las props
- Le paso el componente a drawerContent y esparzo las props, que en este caso es lo que hay en **screenOptions**

```
import React from 'react'
import {DrawerContentComponentProps, DrawerContentScrollView, DrawerItemList,
createDrawerNavigator} from '@react-navigation/drawer'
import { StackNavigator } from './StackNavigator'
import { ProfileScreen } from '../screens/profiles/ProfileScreen'
import { View, Text } from 'react-native'


const Drawer = createDrawerNavigator()

const SideMenuNavigator = () => {

    return (
    <Drawer.Navigator
      drawerContent={(props)=> <CustomDrawerContent {...props} />}
    screenOptions={{
      headerShown: false,
      drawerType: 'slide',
      drawerActiveBackgroundColor: 'orange',
      drawerActiveTintColor: 'white',
      drawerInactiveTintColor: 'black',
      drawerItemStyle:{
        borderRadius: 100,
        paddingHorizontal: 20
      }
      }} >
        <Drawer.Screen name="MenuItem1" component={StackNavigator} />
        <Drawer.Screen name="Profile" component={ProfileScreen} />
    </Drawer.Navigator>
  )
}

const CustomDrawerContent = (props: DrawerContentComponentProps)=>{

  return(<DrawerContentScrollView>
    <View style={{
      height: 200,
      backgroundColor: '#ded1a9',
      margin: 30,
      borderRadius: 50
    }}></View>

    <DrawerItemList {...props} />
  </DrawerContentScrollView>)
}

export default SideMenuNavigator
```

- Cuando pongo la pantalla horizontal queda mucho espacio abajo, podría mostrar el menú
- No es para nada complicado. Utilizo **useWindowDimensions** para obtener las dimensiones de la pantalla

- Si en lugar de 'slide' en la propiedad drawerTpe coloco 'permanent' siempre se muestra el menú
- renderizo condicionalmente según las dimensiones de la pantalla para que siempre se muestre el menú cuando esté en horizontal

```
import React from 'react'
import {DrawerContentComponentProps, DrawerContentScrollView, DrawerItemList,
createDrawerNavigator} from '@react-navigation/drawer'
import { StackNavigator } from './StackNavigator'
import { ProfileScreen } from '../screens/profiles/ProfileScreen'
import { View, Text, useWindowDimensions } from 'react-native'


const Drawer = createDrawerNavigator()

const SideMenuNavigator = () => {

  const dimensions = useWindowDimensions();

    return (
    <Drawer.Navigator
      drawerContent={(props)=> <CustomDrawerContent {...props} />}
    screenOptions={{
      headerShown: false,
      drawerType: dimensions.width >= 758? 'permanent': 'slide', //siempre se
mostrará el menú en horizontal
      drawerActiveBackgroundColor: 'orange',
      drawerActiveTintColor: 'white',
      drawerInactiveTintColor: 'black',
      drawerItemStyle:{
        borderRadius: 100,
        paddingHorizontal: 20
      }
      }} >
        <Drawer.Screen name="MenuItem1" component={StackNavigator} />
        <Drawer.Screen name="Profile" component={ProfileScreen} />
    </Drawer.Navigator>
  )
}

const CustomDrawerContent = (props: DrawerContentComponentProps)=>{

  return(<DrawerContentScrollView>
    <View style={{
      height: 200,
      backgroundColor: '#ded1a9',
      margin: 30,
      borderRadius: 50
    }}><Text style={{color: 'white', fontSize:30, lineHeight: 200, textAlign:
'center'}}>Image</Text>
    </View>

    <DrawerItemList {...props} />
```

```
    </DrawerContentScrollView>)
}

export default SideMenuNavigator
```

---

useSafeAreaInserts

- Voy a ProfileScreen
- A veces hay teléfonos que tienen un notch muy grande y no es seguro que haya contenido en esa zona del teléfono
- El SafeAreaView limita como muestro el contenido, por ejemplo con un ScrollView
- Para que no limite el scroll usamos useSfaeAreaInsets
- Uso el dispatch con DrawerActions para disparar en el onPress el Menu
- Puedo tipar el useNavigation si así lo deseo
- También se puede trabajar siempre con el dispatch, con rutas específicas (se verá más adelante)

```
import React from 'react'
import { Text, View } from 'react-native'
import { useSafeAreaInsets } from 'react-native-safe-area-context'
import { PrimaryButton } from '../../components/shared/PrimaryButton'
import { DrawerActions, useNavigation } from '@react-navigation/native'

export const ProfileScreen = () => {

  const {top} = useSafeAreaInsets()
  const navigator = useNavigation()

  return (
    <View style={{flex: 1, paddingHorizontal: 20, marginTop: top+10}}>
        <Text style={{textAlign: 'center', fontSize:30, marginBottom: 10}}
>Profile</Text>

        <View style={{flex: 1, flexDirection: 'column', justifyContent: 'flex-
end', marginBottom: 10}}>
          <PrimaryButton label="Abrir Menú" onPress=
{()=>navigator.dispatch(DrawerActions.toggleDrawer)}/>
        </View>
    </View>
  )
}
```

Bottom Tabs, Material Top Tabs

- Instalación

```
npm i @react-navigation/bottom-tabs
```

- En src/routes creo BottomNavigator.tsx

```
import { createBottomTabNavigator } from '@react-navigation/bottom-tabs';
import { Tab1Screen } from '../screens/tabs/Tab1Screen';
import { Tab2Screen } from '../screens/tabs/Tab2Screen';
import { Tab3Screen } from '../screens/tabs/Tab3Screen';

const Tab = createBottomTabNavigator();

export const BottomNavigator=()=> {
  return (
    <Tab.Navigator>
      <Tab.Screen name="Tab1" component={Tab1Screen} />
      <Tab.Screen name="Tab2" component={Tab2Screen} />
      <Tab.Screen name="Tab3" component={Tab3Screen} />
    </Tab.Navigator>
  );
}
```

- Hay que determinar en qué momento voy a mostrar el Bottom
- Para mantener el Drawer, en liugar del StackNavigator coloco el BottomNavigator
- App.tsx

```
import React from 'react'
import {DrawerContentComponentProps, DrawerContentScrollView, DrawerItemList,
createDrawerNavigator} from '@react-navigation/drawer'
import { StackNavigator } from './StackNavigator'
import { ProfileScreen } from '../screens/profiles/ProfileScreen'
import { View, Text, useWindowDimensions } from 'react-native'
import { BottomNavigator } from './BottomNavigator'


const Drawer = createDrawerNavigator()

const SideMenuNavigator = () => {

  const dimensions = useWindowDimensions();

    return (
    <Drawer.Navigator
      drawerContent={(props)=> <CustomDrawerContent {...props} />}
    screenOptions={{
      headerShown: false,
      drawerType: dimensions.width >= 758? 'permanent': 'slide', //siempre se
mostrará el menú en horizontal
      drawerActiveBackgroundColor: 'orange',
      drawerActiveTintColor: 'white',
      drawerInactiveTintColor: 'black',
      drawerItemStyle:{
        borderRadius: 100,
```

```
        paddingHorizontal: 20
      }
      }} >
        <Drawer.Screen name="Home" component={BottomNavigator} />
        <Drawer.Screen name="Profile" component={ProfileScreen} />
    </Drawer.Navigator>
  )
}

const CustomDrawerContent = (props: DrawerContentComponentProps)=>{

  return(<DrawerContentScrollView>
    <View style={{
      height: 200,
      backgroundColor: '#ded1a9',
      margin: 30,
      borderRadius: 50
    }}><Text style={{color: 'white', fontSize:30, lineHeight: 200, textAlign:
'center'}}>Image</Text>
    </View>

    <DrawerItemList {...props} />
  </DrawerContentScrollView>)
}

export default SideMenuNavigator
```

- Necesito el menú de hamburguesa para desplegar el menú lateral
- Primero personalicemos el BottomNavigator
- Uso sceneContainerStyle para el background de la pantalla
- Uso otras props para personalizar
- En options, puedo pasarle el título para que se muestre arriba
- También el tabBarIcon que pide un functional component, le puedo pasar un icono o lo que yo quiera
    - Puedo desestructurar el color de las props.
    - Esto hace que el color del icono (en este caso el texto) sea visible cuando la tab esté activa

```
import { createBottomTabNavigator } from '@react-navigation/bottom-tabs';
import { Tab1Screen } from '../screens/tabs/Tab1Screen';
import { Tab2Screen } from '../screens/tabs/Tab2Screen';
import { Tab3Screen } from '../screens/tabs/Tab3Screen';
import { Text } from 'react-native';

const Tab = createBottomTabNavigator();

export const BottomNavigator=()=> {
  return (
    <Tab.Navigator
    sceneContainerStyle={{
        backgroundColor: 'rgba(241, 237, 231, 0.2)',
    }}
```

```
    screenOptions={{
        //headerShown: false,
        tabBarLabelStyle:{
            marginBottom: 5
        },
        headerStyle: { //eliminar la linea
            elevation: 0,
            borderColor: 'transparent',
            shadowColor: 'transparent'
        },
        tabBarStyle:{ //para eliminar la linea en ios
            borderTopWidth: 0,
            elevation: 0
        }

    }}
    >
      <Tab.Screen name="Tab1" options={{title: "Tab1", tabBarIcon: ({color})=>
(<Text style={{color: color}}>Tab</Text>)}} component={Tab1Screen} />
      <Tab.Screen name="Tab2" options={{title: "Tab2", tabBarIcon: ({color})=>
(<Text style={{color: color}}>Tab</Text>)}}  component={Tab2Screen} />
      <Tab.Screen name="Tab3" options={{title: "Tab3", tabBarIcon: ({color})=>
(<Text style={{color: color}}>Tab</Text>)}}  component={Tab3Screen} />
    </Tab.Navigator>
  );
}
```

Menú de hamburguesa

- En Tab1Screen llamo al useNavigation
- Uso de nuevo el useEffect con navigation.setOptions

```
import { DrawerActions, useNavigation } from '@react-navigation/native'
import React, { useEffect } from 'react'
import { Pressable, Text, View } from 'react-native'

export const Tab1Screen = () => {

  const navigation = useNavigation()

  useEffect(()=>{
    navigation.setOptions({
      headerLeft: ()=>(
       <Pressable onPress={()=>navigation.dispatch(DrawerActions.toggleDrawer)}>
        <Text>Menu</Text>
       </Pressable>
      )
    })
  }, [])
```

```
  return (
    <View>
        <Text>Tab1Screen</Text>
    </View>
  )
}
```

- Ahora aparece el Menu que si toco me muestra el menú lateral
- Hacer esto en cada uno de los tabs no sería lo adecuado. Mejor crear un componente para ello
- Creo en componentes/shared/HamburguerMenu.tsx

```
import { DrawerActions, useNavigation } from '@react-navigation/native'
import React, { useEffect } from 'react'
import { Pressable, Text } from 'react-native'

const HamburguerMenu = () => {

    const navigation = useNavigation()

    useEffect(()=>{
      navigation.setOptions({
        headerLeft: ()=>(
         <Pressable onPress={()=>navigation.dispatch(DrawerActions.toggleDrawer)}>
          <Text>Menu</Text>
         </Pressable>
        )
      })
    }, [])

    return (<></>) //hay que regresar siempre un JSX, se puede usar un Fragment
vacío
}

export default HamburguerMenu
```

- Lo coloco en el return de Tab1, Tab2 y Tab3

Material Top Navigator

- Instalación

> npm i @react-navigation/material-top-tabs react-native-tab-view npm i react-native-pager-view

- Copio el ejemplo de la documentación y lo adapto a mi contenido

```
import { createMaterialTopTabNavigator } from '@react-navigation/material-top-
tabs';
import { ProfileScreen } from '../screens/profiles/ProfileScreen';
```

```
import { AboutScreen } from '../screens/about/AboutScreen';

const Tab = createMaterialTopTabNavigator();

export const  TopNavigator=()=> {
  return (
    <Tab.Navigator>
      <Tab.Screen name="Perfil" component={ProfileScreen} />
      <Tab.Screen name="About" component={AboutScreen} />
    </Tab.Navigator>
  );
}
```

- Para usarlo, pongamos que quiero mostrarlo en el Tab2
- Entonces, voy a BottomNavigator, y en lugar de mostrar el Tab2Screen, coloco el TopNavigator

```
<Tab.Screen name="Tab2"  options={{title: "Tab2", tabBarIcon: ({color})=>(<Text
style={{color: color}}>Tab</Text>)}}  component={TopNavigator} />
```

Instalar iconos

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

- Editar android/app/build.gradle

```
project.ext.vectoricons =[
  iconFontNames: ['Ionicons.ttf']
]

apply from: file ("../../node_modules/react-native-vector-icons/fonts.gradle")
```

- Para usarlo importo Icon

```
import React from 'react'
import {Text, View } from 'react-native'
import HamburguerMenu from '../../components/shared/HamburguerMenu'
import Icon from 'react-native-vector-icons/Ionicons'

export const Tab3Screen = () => {
  return (
    <View>
      <HamburguerMenu />
        <Text style={{fontSize:40, textAlign: 'center', marginVertical:
10}}>Tab3Screen</Text>
        <Icon name='rocket' size={30} color="#2d6ab9" style={{textAlign:
'center'}} />
```

```
    </View>
  )
}
```

- Para colocar los iconos en las Tabs puedo usar Icon o customizar un componente

```
import React from 'react'
import { Icon } from 'react-native-vector-icons/Icon'

interface Props{
    name: string
    size: number
    color: string
}


const IconComponent = ({name,size,color}: Props) => {
  return (
    <Icon name={name} size={size} color={color}     />
  )
}

export default IconComponent
```

- Ahora solo tengo que colocarlo.

```
import React from 'react'
import {Text, View } from 'react-native'
import HamburguerMenu from '../../components/shared/HamburguerMenu'
import Icon from 'react-native-vector-icons/Ionicons'

export const Tab3Screen = () => {
  return (
    <View>
      <HamburguerMenu />
        <Text style={{fontSize:40, textAlign: 'center', marginVertical:
10}}>Tab3Screen</Text>
        <Icon name='rocket' size={30} color="#2d6ab9" style={{textAlign:
'center'}} />
    </View>
  )
}
```

- Para colocarlo en las tabs

```
<Tab.Screen name="Tab3" options={{title: "Tab3", tabBarIcon:()=><IconComponent
name="bicycle" size={30} color="orange" /> }}  component={Tab3Screen} />
```

REACT NATIVE - COMPARTIR ESTADO GLOBAL ZUSTAND

- Practiquemos como comunicar componentes con Zustand
- **NOTA**: si da problemas *evaluating project ':app'* indicar donde se encuentra java en gradle.properties

```
org.gradle.java.home=C:\\Program Files\\Java\\jdk-17
```

- Creo src/Main.tsx

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

export const Main = () => {
  return (
    <View>
        <Text>Hello world!</Text>
    </View>
  )
}
```

- Lo renderizo en index.js en lugar de App

```
/**
 * @format
 */

import {AppRegistry} from 'react-native';
import {App} from './App';
import {name as appName} from './app.json';
import { Main } from './src/Main';

AppRegistry.registerComponent(appName, () => Main);
```

- Creo la estructura de directorios
  - presentation
    - store
    - screens
      - home/HomeScreen.tsx
      - profile/ProfileScreen.tsx
      - settings/SettingsScreen.tsx
  - config/appTheme.tsx
- Creo en appTheme los estilos globales

```
import {StyleSheet} from 'react-native'

export const styles = StyleSheet.create({
  container:{
    paddingHorizontal: 20,
    marginTop: 10
  },
  title:{
    fontSize: 20,
    fontWeight: 'bold',
    color: 'black'
  },

  primaryButton:{
    backgroundColor: 'purple',
    padding: 10,
    borderRadius: 5,
    marginVertical: 10
  }
})
```

- Aplico los estilos de los containers a los View de SettingsScreen, ProfileScreen y HomeScreen
- Configuro el BottomTabNavigator para mostrar estos tres componentes

> npm install @react-navigation/native @react-navigation/native-stack npm install react-native-screens react-native-safe-area-context npm install @react-navigation/bottom-tabs

- Creo la carpeta presentation/routes/BottomTabNavigator.tsx

```
import { createBottomTabNavigator } from '@react-navigation/bottom-tabs';
import { HomeScreen } from '../screens/home/HomeScreens';
import Settingsscreen from '../screens/settings/Settingsscreen';
import { ProfileScreen } from '../profile/ProfileScreen';

const Tab = createBottomTabNavigator();

export const BottomTabNavigator=()=>{
  return (
    <Tab.Navigator>
      <Tab.Screen name="Home" component={HomeScreen} />
      <Tab.Screen name="Profile" component={ProfileScreen} />
      <Tab.Screen name="Settings" component={Settingsscreen} />
    </Tab.Navigator>
  );
}
```

- Envuelvo el Main en un NavigationContainer

```
import React from 'react'
import { NavigationContainer } from '@react-navigation/native'
import { BottomTabNavigator } from './presentation/routes/BottomTabNavigator'

export const Main = () => {
  return (
   <NavigationContainer>
      <BottomTabNavigator />
   </NavigationContainer>
  )
}
```

Zustand - Gestor de estado

> npm i zustand

- Creo en store/profile-store.tsx
- Para crear un store usamos la funcion create de Zustand, lo usaremos como un hook
- Usamos una interfaz para tiparlo y doble paréntesis (uno para el tipado y otro para el callback)
- Notar que el callback envuelve las llaves en paréntesis para hacer implícito el return
- **set** es el dispatch (o la función) que debo llamar para cambiar los valores de mi store
- **get** me va a permitir obtener el store

```
import { create } from "zustand"


interface ProfileState{
    name: string
    email: string
}

export const useProfileStore = create<ProfileState>()((set, get)=>({
    name: 'Bill Murray',
    email: "billybilly@gmail.com"
}))
```

- En ProfileScreen renderizo el estado
- Puedo usar desestructuración pero a veces dispara rerenders indeseados

```
import React from 'react'
import { Text, View } from 'react-native'
import { styles } from '../../config/appTheme'
import { useProfileStore } from '../store/profile-store'

export const ProfileScreen  = () => {
```

```
  const name = useProfileStore(get=> get.name)
  const email = useProfileStore(get=> get.email)

  return (
    <View style={styles.container} >
      <Text style={styles.title}>{name}</Text>
      <Text style={styles.title}>{email}</Text>
    </View>
  )
}
```

Cambios en el Store

- En ProfileScreen creo un par de botones
- En el onPress, llamando en el callback a useProfileStore tengo **subscribe** y estar suscrito a los cambios del state
- Tengo el **setState** y el **getState**
- Prefiero crear las funciones que voy a usar en el store para que la lógica siempre esté definida en mi store
- No siempre es la mejor opción, a criterio del desarrollador/a

```
import React from 'react'
import { Pressable, Text, View } from 'react-native'
import { styles } from '../../config/appTheme'
import { useProfileStore } from '../store/profile-store'

export const ProfileScreen  = () => {

  const name = useProfileStore(get=> get.name)
  const email = useProfileStore(get=> get.email)

  return (
    <View style={styles.container} >
      <Text style={styles.title}>{name}</Text>
      <Text style={styles.title}>{email}</Text>

      <Pressable style={styles.primaryButton}
      onPress={()=>useProfileStore.setState({name: "Miguel Castaño"})}
      >
        <Text style={styles.title}>Cambio Nombre</Text>
      </Pressable>

      <Pressable style={styles.primaryButton}
      onPress={()=> useProfileStore.setState({email: "migue@gmail.com"})}
      >
        <Text style={styles.title}>Cambio email</Text>
      </Pressable>
```

```
    </View>
  )
}
```

- En la documentación de Zustand hay veces que define la lógica en la interfaz y otras crea un export type Actions donde define las acciones
- Uso **set** para cambiar el estado

```
import { create } from "zustand"


interface ProfileState{
    name: string
    email: string
    changeProfile: (name: string, email: string)=> void
}

export const useProfileStore = create<ProfileState>()((set, get)=>({
    name: 'Bill Murray',
    email: "billybilly@gmail.com",

    changeProfile: (name: string, email: string)=>{
        set({name, email})
    }
}))
```

- Si necesito acceso a esa función la extraigo de useProfileStore

```
import React from 'react'
import { Pressable, Text, View } from 'react-native'
import { styles } from '../../config/appTheme'
import { useProfileStore } from '../store/profile-store'

export const ProfileScreen  = () => {

  const name = useProfileStore(get=> get.name)
  const email = useProfileStore(get=> get.email)
  const changeProfile = useProfileStore(get=>get.changeProfile)

  return (
    <View style={styles.container} >
      <Text style={styles.title}>{name}</Text>
      <Text style={styles.title}>{email}</Text>

      <Pressable style={styles.primaryButton}
      onPress={()=>useProfileStore.setState({name: "Miguel Castaño"})}
      >
        <Text style={styles.title}>Cambio Nombre</Text>
```

```
      </Pressable>

      <Pressable style={styles.primaryButton}
      onPress={()=> useProfileStore.setState({email: "migue@gmail.com"})}
      >
        <Text style={styles.title}>Cambio email</Text>
      </Pressable>
      <Pressable style={styles.primaryButton}
      onPress={()=> changeProfile("Bill Murray", "billybilly@gmail.com")}
      >
        <Text style={styles.title}>regresar estado</Text>
      </Pressable>
    </View>
  )
}
```

- Zustand además tiene muchos middlewares poderosos
- Creemos un counter!

```
import { create } from "zustand"


interface CounterState{
    count: number
    increment: (value: number)=> void
}

export const useCounterStore = create<CounterState>()((set,get)=>({

    count: 1,
    increment: (value)=>{set(state=> ({count:state.count + value}))},
}))
```

- Otra manera sería

```
increment: (value)=>{set({count: get().count + value})},
```

REACT NATIVE - App Películas parte 1

- Vamos a hacer el esqueleto y la estructura necesaria para la aplicación
- Lo haremos bajo principios de arquitectura DDD y patrones (haremos uso del patrón repositorio)
- Veremos las diferencias entre config, core, infraestructure y presentation y cómo se integran
- Nos vamos a centrar en traer la info de movieDB
- La app debe ser lo suficiente flexible para tolerar cambios

Configuración de pantallas y directorios

- Creo en src
    - config
        - adapters (dónde guardaré los archivos de patrón adaptador)
        - helpers
    - core (casos de uso)
    - presentation
        - hooks
        - components
        - navigation
        - screens
            - details (detalles de peliculas)
            - home
    - infraestructure (es nuestro middleman)
- Creo los componentes HomeScreen y Detailsscreen
- Muevo App dentro de src

Navegación entre pantallas

- Hago las instalaciones necesarias para integrar la navegación en la aplicación

> npm i @react-navigation/native npm i react-native-screens react-native-safe-area-context

- Para la navegación Stack

> npm i @react-navigation/stack npm i react-native-gesture-handler @react-native-masked-view/masked-view

- Puedo configurar mi NavigationContainer
- Hay que colocar este import en el top del archivo principal

```
import 'react-native-gesture-handler'
import React from 'react';
import {Text} from 'react-native'
import { NavigationContainer } from '@react-navigation/native';

export const App=(): React.JSX.Element=>{

  return(

   <NavigationContainer>
      <Text>Hello World</Text>
   </NavigationContainer>

   )
}
```

- Uso el ejemplo de la documentación para crear el Stack Navigator
- Tipo el Navigator con RootstackParams
- Empleo screenOptions para ocultar los headers

```
import { createStackNavigator } from '@react-navigation/stack';
import { HomeScreen } from '../screens/home/Homescreen';
import { Detailsscreen } from '../screens/details/DetailsScreen';

const Stack = createStackNavigator();

export const Navigation =()=>{
  return (
    <Stack.Navigator>
      <Stack.Screen name="Home" component={HomeScreen} />
      <Stack.Screen name="Details" component={Detailsscreen} />
    </Stack.Navigator>
  );
}
```

- Coloco Navigation dentro de App

Obtener Peliculas - TheMovieDB

- Creo una cuenta en theMovieDB.org
- En Editar Perfil / API / le doy cick al enlace para generar la llave
- La guardo (y el token también) en .env
- Creo .env.template para guardar las variables vacias que digan que es lo que necesito configurar
- Añado .env a .gitignore
- Creo el README

```
# README

1. Clonar el proyecto
2. Instalar dependencias `npm install`
3. Clonar el archivo .env.template a .env y configurar las variables de entorno
4. Ejecutar el proyecto con `npm run start`
```

- Voy a la documentación de MovieDB
- En Authentication tengo el token de autenticación (no es el mismo que obtuve con la llave). Lo copio en .env
- Puedo seleccionar Node como lenguaje
- Para buscar un endpoint en especifico puedo ir al buscador JUMP TO y , en este caso, escribo **now playing**
- Donde me da el ejemplo de código puedo elegir que quiero usar, si fetch, axios. Selecciono Axios

- Este es el ejemplo

```
const axios = require('axios');

const options = {
  method: 'GET',
  url: 'https://api.themoviedb.org/3/movie/now_playing?language=en-US&page=1',
  headers: {
    accept: 'application/json',
    Authorization: 'Bearer eyJh....'
  }
};

axios
  .request(options)
  .then(function (response) {
    console.log(response.data);
  })
  .catch(function (error) {
    console.error(error);
  });
```

- Implementaremos Axios con el patrón adaptador
- Por ahora solo necesito el endpoint que es `https://api.themoviedb.org/3/movie/now_playing`
- Debo mandar mi API_KEY como params en POSTMAN o similares bajo el nombre de api_key
- En THUNDERCLIENT es en Query
- Puedo añadirle language es para que aparezca en español

Patrón Adaptador

- En adapters creo en la carpeta http/http.adapter.tsx
- Porqué lo hago con una clase abstracta?
  - Porque yo no quiero crear instancias de esta clase
  - Lo que aqui defina no va estar implementado
  - Voy a definir las reglas de los métodos y propiedades que deben de tener las clases que extiendan de mi adaptador
  - Options lo hago opcional. Puedo usar Record para decirle que el tipo de dato será de tipo string y el valor al que apunta unknown
  - Si quisiera hacer algo muy genérico usaria tipo any
  - Resuelve en una promesa de tipo genérico

```
export abstract class HttpAdapter{

    abstract get<T>(url: string, options?: Record<string, unknown>) : Promise<T>
}
```

- Instalo axios
- Creo en http/axios.adapter.tsx
- Implemento el método get, uso async await y un trycatch
- Extraigo la data del fetch de axios
- Tipo la respuesta de axios con el mismo genérico que el método
- Retorno la data

```
import { HttpAdapter } from "./http.adapter";
import axios, { AxiosInstance } from 'axios'

interface Options{
    baseURL: string,
    params: Record<string,string>
}

export class AxiosAdapter implements HttpAdapter{

    private axiosInstance : AxiosInstance;

    constructor(options: Options){
        this.axiosInstance = axios.create({
            baseURL: options.baseURL,
            params: options.params
        })
    }


    async get<T>(url: string, options?: Record<string, unknown>) : Promise<T>{

        try {
            const {data} = await this.axiosInstance.get<T>(url,options)

            return data

        } catch (error) {
            throw new Error(`Error fetching get: ${url}` )
        }
    }
}
```

Casos de uso - Now Playing

- En /core/use-cases/movies/now-playing.use-case.tsx
- Now playing debe traer las peliculas usando el adaptador de axios
- Se busca que los casos de uso sean agnósticos (que no necesiten paquetes de terceros para funcionar)
- Le paso de parámetro *fetcher* que serña del tipo HttpAdapater (el mismo que mi adaptador de axios)
- Podría pasar el url como parámetro para hacerlo más genérico, pero yo se que en este caso de uso siempre voy a llamar al mismo endpoint

- Como la cabecera de la url la voy a configurar en el adaptador, añado solo la parte de la url de '/now_playing'
- Hay que tipar la respuesta, por lo que hago un llamado al endpoint desde THUNDERCLIENT y uso paste JSON as code
- La pego en infraestructure/interfaces/movieDBResponses.tsx
- Cambio algunos nombres y en lugar de usar guiones bajos uso camelCase

```
export interface NowPlayingResponse {
    dates:         Dates;
    page:          number;
    results:       Result[];
    totalPages:   number;
    totalResults: number;
}

export interface Dates {
    maximum: string;
    minimum: string;
}

export interface Result { //Esto sería Movie
    adult:             boolean;
    backdrop_path:     string;
    genre_ids:         number[];
    id:                number;
    original_language: string;
    original_title:    string;
    overview:          string;
    popularity:        number;
    poster_path:       string;
    release_date:      string;
    title:             string;
    video:             boolean;
    vote_average:      number;
    vote_count:        number;
}

/*
export enum OriginalLanguage {
    En = "en",
    Es = "es",
}
*/
```

- Lo que va a regresar el fetcher va a ser de tipo NowPlayingResponse
- Para tipar la respuesta de la promisa bien podría usar la interfaz de Movie, pero podría ser que la DB cambiara en un futuro, y eso podría darme un dolor de cabeza
- En lugar de usar una interfaz proveniente de la db, creo /core/entities/movie.entity.tsx
- Usualmente la entity es una clase, pero también puedo trabajar con una interfaz y luego transformarla en una clase

```
export interface Movie{
    id: number
    title: string
    description: string
    releaseDate: Date
    rating: number
    poster: string
    backdrop: string
}
```

- Entonces, voy a usar mi propio modelo. En la respuesta los nombres vienen diferente, por ejemplo *poster_path* en lugar de poster
- Voy a tener que transformar la data de la respuesta para que concuerde con mi entity, ya que la respuesta de la DB no es exactamente igual
- Todo esto lo hago porque si el día de mañana la interfaz que extraje de la DB cambia, mi aplicación crashearía

```
import { HttpAdapter } from "../../../config/adapters/http/http.adapter";
import { NowPlayingResponse } from
"../../../infraestructure/interfaces/movieDBResponses";
import { Movie } from "../../entities/movie.entity";


export const moviesNowPlayingUseCase = async(fetcher: HttpAdapter):
Promise<Movie[]>=>{
    try {
        const nowPlaying = await fetcher.get<NowPlayingResponse>('/now_playing')

        console.log({nowPlaying})

        return []//Aquí voy a tener que transformar la data para adaptarlo a como
lo he tipado yo en mi entidad
                //y no con los nombres y el conjunto de propiedades que viene
desde la DB

    } catch (error) {
        throw new Error (`Error fetching movies - Now Playing`)
    }
}
```

Custom Hook useMovies

- Creo en src/presentation/hooks/useMovies.tsx
- Llamo al caso de uso en un useEffect
- Antes creo un index.ts en use-cases y lo exporto todo

```
export * from './movies/now-playing.use-case'
```

- Puedo hacer el import com UseCases para disponer de los distintos casos de uso que vaya creando con la notación de punto
- La función me pide el fetcher de tipo HttpAdapter.
- Creo un nuevo adaptador llamado MovieDBAdapter con un ainstancia de mi adaptador Axios, le paso lo que necesita
    - En baseURL le paso la url hasta now_playing (sin incluir este ni el slash que le precede)
    - En params le paso la api_key y el language

```
import { AxiosAdapter } from "./axios.adapter";

export const MovieDBFetcher = new AxiosAdapter({
    baseURL: "https://api.themoviedb.org/3/movie",
    params: {
        api_key: "mi_api_key", //será una variable de entorno
        language: 'es'
    }
})
```

- Le paso el fetcher (MovieDBFetcher) al caso de uso
- Llamo la función dentro del useEffect

```
import React, { useEffect, useState } from 'react'
import { Movie } from '../../core/entities/movie.entity'
import * as UseCases from '../../core/use-cases'
import { MovieDBFetcher } from '../../config/adapters/http/movieDB.adapter'

const useMovies = () => {

    const [isLoading, setIsLoading] = useState(true)
    const [nowPlaying, setNowPlaying] = useState<Movie[]>([])

    useEffect(()=>{
        initLoad()
    },[])

    const initLoad= async()=>{
        const nowPlayingMovies = await
UseCases.moviesNowPlayingUseCase(MovieDBFetcher)
    }

    return {


    }
}
```

```
export default useMovies
```

- En HomeScreen llamo al hook, no desestructuro nada porque todavñia no regresa nada

```
import React from 'react'
import { Text, View } from 'react-native'
import useMovies from '../../hooks/useMovies'

export const HomeScreen  = () => {

  const {} = useMovies()

  return (
    <View>
      <Text>HomeScreen</Text>
    </View>
  )
}
```

- Ahora en consola (por el console.log de nowPlaying en el caso de uso) deberían aparecer los resultados pero con Object Object
- Falta transformar la data resultante en algo que luzca como mi interfaz de Movie, para usar mi propia implementación

---

Patrón Mapper - MovieMapper

- Creo en src/infraestructure/mappers/movie.mapper.ts
- Puedo hacer el mapper con una función pero lo haré con una clase
- Creo un método estático. Al ser estático no necesito instanciar la clase
- El método recibe result de tipo Result (mirar la interfaz de MovieDB) y regresará una Movie.
- Si Movie fuera una clase tendría que instanciarla pero hasta ahora la hemos manejado con una interfaz
- releaseDate me regresa un dato de tipo string, por lo que creo la fecha con new Date
- Para ver la imagen de la propiedad poster, esta solo devuelve un string que no me sirve, para visualizarla necesito añadir https://image.tmdb.org/t/p/w500/url_de_posterPath
- Uso un template string, quito el slash final porque el string vendrá con el slash inicial

```
import { Movie } from "../../core/entities/movie.entity";
import { Result } from "../interfaces/movieDBResponses";

export class MovieMapper{

    static fromMovieDBResultToEntity(result: Result): Movie{
        return {
            id: result.id,
            title: result.title,
            description: result.overview,
```

```
            releaseDate: new Date(result.release_date),
            rating: result.vote_average,
            poster: `https://image.tmdb.org/t/p/w500${result.poster_path}`,
            backdrop: `https://image.tmdb.org/t/p/w500${result.backdrop_path}`
        }
    }
}
```

- Si la url de poster no viniera, puedo manejar la excepción en el mapper
- Voy al caso de uso.
- Se recomienda que los casos de uso sean funciones puras que resuelvan con los argumentos dados
    - Pero en este caso quiero transformar la data por lo que usaré el mapper
    - En .results tengo la data, hago un map y la paso por el mapper
    - En JS cuando tengo el mismo argumento pasado como parámetro para una función puedo obviar ambos

```
import { HttpAdapter } from "../../../config/adapters/http/http.adapter";
import { NowPlayingResponse } from
"../../../infraestructure/interfaces/movieDBResponses";
import { MovieMapper } from "../../../infraestructure/mappers/movie.mapper";
import { Movie } from "../../entities/movie.entity";


export const moviesNowPlayingUseCase = async(fetcher: HttpAdapter):
Promise<Movie[]>=>{
    try {
        const nowPlaying = await fetcher.get<NowPlayingResponse>('/now_playing')



     //return
nowPlaying.results.map(result=>MovieMapper.fromMovieDBResultToEntity(result))

       return nowPlaying.results.map(MovieMapper.fromMovieDBResultToEntity)

    } catch (error) {
        throw new Error (`Error fetching movies - Now Playing`)
    }
}
```

- Hago un console.log en la función initLoad del hook useMovies para visualizar el resultado en consola
- Retorno el state de isLoading y nowPlaying que es donde estan mis películas

```
import React, { useEffect, useState } from 'react'
import { Movie } from '../../core/entities/movie.entity'
import * as UseCases from '../../core/use-cases'
import { MovieDBFetcher } from '../../config/adapters/http/movieDB.adapter'
```

```
const useMovies = () => {

    const [isLoading, setIsLoading] = useState(true)
    const [nowPlaying, setNowPlaying] = useState<Movie[]>([])

    useEffect(()=>{
        initLoad()
    },[])

    const initLoad= async()=>{
        const nowPlayingMovies = await
UseCases.moviesNowPlayingUseCase(MovieDBFetcher)
        setNowPlaying(nowPlayingMovies)
        //console.log({nowPlayingMovies})
    }

    return {
        isLoading,
        nowPlaying
    }
}

export default useMovies
```

Casos de uso restantes

- Quiero implementar tres nuevos casos de uso
    - /upcoming
    - /top_rated
    - /popular moviesPopularUseCase
- Creo el caso de uso y le paso el fetcher, devuelvo una promesa del tipo entity que yo me cree basándome en la interfaz que extraigo con paste JSON as code
- Hago un llamado al endpoint, copio el resultado y uso pasteJSON as code para obtener las interfaces
- Creo mi entity para luego crear el mapper y obtener la data como yo quiero sin depender de terceros
- Expongo el caso de uso Upcoming
- upcoming.inteface.ts

```
// Generated by https://quicktype.io

export interface UpcomingResults {
    dates:         Dates;
    page:          number;
    results:       Result[];
    total_pages:   number;
```

```
    total_results: number;
}

export interface Dates {
    maximum: string;
    minimum: string;
}

export interface Result {
    adult:              boolean;
    backdrop_path:      string;
    genre_ids:          number[];
    id:                 number;
    original_language:  string;
    original_title:     string;
    overview:           string;
    popularity:         number;
    poster_path:        string;
    release_date:       string;
    title:              string;
    video:              boolean;
    vote_average:       number;
    vote_count:         number;
}
```

- Creo mi entity

```
export interface UpcomingMovie{
    genreIds: number[]
    id: number
    originalLanguage: string
    originalTitle: string
    posterPath: string,
    backdropPath: string
    releaseDate: Date
    rate: number
}
```

- Creo el Mapper

```
import { UpcomingMovie } from "../../core/entities/upcoming.entity";
import { Result } from "../interfaces/upcomingMovies.interface";

export class UpcomingMovieMapper{

    static getUpcomingResultToEntity (result: Result ) : UpcomingMovie{
        return {
            genreIds: result.genre_ids,
            id: result.id,
```

```
            originalLanguage: result.original_language,
            originalTitle: result.original_title,
            posterPath: `https://image.tmdb.org/t/p/w500${result.poster_path}`,
            backdropPath:
`https://image.tmdb.org/t/p/w500${result.backdrop_path}`,
            releaseDate: new Date(result.release_date),
            rate: result.vote_average
        }
    }
}
```

- Voy al caso de uso y utilizo .map
- Lo exporto en el archivo de barril

```
import { HttpAdapter } from "../../../config/adapters/http/http.adapter"
import { UpcomingResults } from
"../../../infraestructure/interfaces/upcomingMovies.interface"
import { UpcomingMovieMapper } from
"../../../infraestructure/mappers/upcoming.mapper"
import { UpcomingMovie } from "../../entities/upcoming.entity"


export const moviesUpcomingUseCase= async(fetcher: HttpAdapter):
Promise<UpcomingMovie[] | undefined>=>{
    try {

        const upcomingMovies  = await fetcher.get<UpcomingResults>('/upcoming')

        return
upcomingMovies.results.map(UpcomingMovieMapper.getUpcomingResultToEntity)

    } catch (error) {
     throw new Error(`fetch error to upcoming movies use case`)
    }
}
```

- index.ts

```
export * from './movies/now-playing.use-case'
export * from './movies/upcoming.use-case'
```

- Creo el adaptador

```
import { AxiosAdapter } from "./axios.adapter";

export const UpcomingAdapter = new AxiosAdapter({
    baseURL: 'https://api.themoviedb.org/3/movie',
```

```
    params:{
        api_key: "fe1099f21bfaef01ab67feb300828d6d",
        language: 'es'
    }
})
```

- Creo el hook

```
import React, { useEffect, useState } from 'react'
import { UpcomingMovie } from '../../core/entities/upcoming.entity'
import * as UseCases from '../../core/use-cases'
import { UpcomingAdapter } from '../../config/adapters/http/upcoming.adapter'

const useUpcoming = () => {

    const [isLoading, setIsLoading] = useState(true)
    const [upcomingMovies, setUpcomingMovies] = useState<UpcomingMovie[] |
undefined>([])

    useEffect(()=>{
        initLoadUpcoming()
    }, [])


    const initLoadUpcoming = async ()=>{

        const upcomingMoviesResult = await
UseCases.moviesUpcomingUseCase(UpcomingAdapter)

        console.log({upcomingMoviesResult})

        setUpcomingMovies(upcomingMoviesResult)
    }


    return {
        isLoading,
        upcomingMovies
  }
}

export default useUpcoming
```

- Lo llamo en HomeScreen para ver si funciona
- Por algún motivo no muestra el array de genrelds, solo muestra Array
- Hago el resto de casos de uso
- Para subirlo a git **he quitado la API_KEY hasta que no configure las variables de entorno**

REFACTORING

- En lugar de usar un hook para cada caso de uso, usaremos el mimso useMovies y el Promise.all
- Usaremos la misma entidad Movie para todos.
- Le pongo un nombre genérico a la interfaz para usarla en todos los casos de uso
- Uso la misma entity en todos los casos y states
- En useMovie hago carpintería

```
import React, { useEffect, useState } from 'react'
import { Movie } from '../../core/entities/movie.entity'
import * as UseCases from '../../core/use-cases'
import { MovieDBFetcher } from '../../config/adapters/http/movieDB.adapter'

const useMovies = () => {

    const [isLoading, setIsLoading] = useState(true)
    const [nowPlaying, setNowPlaying] = useState<Movie[]>([])
    const [popularMovie, setpopularMovie] = useState<Movie[]>([])
    const [topRatedMovie, setTopRatedMovie] = useState<Movie[]>([])
    const [upcomingMovie, setupcomimgMovie] = useState<Movie[]>([])

    useEffect(()=>{
        initLoad()
    },[])

    const initLoad= async()=>{
        const nowPlayingPromise = UseCases.moviesNowPlayingUseCase(MovieDBFetcher)
        const popularPromise = UseCases.PopularUseCase(MovieDBFetcher)
        const topRatedPromise = UseCases.TopRatedUseCase(MovieDBFetcher)
        const upcomingPromise = UseCases.moviesUpcomingUseCase(MovieDBFetcher)

        const[
            nowPlayingMovies,
            popularMovies,
            topRatedMovies,
            upcomingMovies
        ] = await Promise.all([nowPlayingPromise, popularPromise, topRatedPromise,
upcomingPromise])


        setNowPlaying(nowPlayingMovies)
        setpopularMovie(popularMovies)
        setTopRatedMovie(topRatedMovies)
        setupcomimgMovie(upcomingMovies)
    }

    return {
        isLoading,
        nowPlaying
    }
}

export default useMovies
```

- **NOTA:** esta es una aproximación al DDD, faltarían los data sources y el repositorio que interactuaría con los casos de uso.
    - Los casos de uso no serían una función sino una clase o un Factory Function, y no le pasariamos el fetcher si no que le inyectaríamos el repositorio para poder consumir la info

---

REACT NATIVE - App Películas parte 2

- En esta sección veremos como aplicar estilos pero también hay más cosas que hacer, como
    - Otras peticiones para traer los actores, traer la info de unsa sola película, para mostrar las películas
    - Algún tipo de carrusel, infiniteScroll, configurar las variables de entorno

Carrusel de posters

- En HomeScreen coloco un ScrollView
- Uso usesafeAreaInsets para tener una distancia segura de la pantalla desde el top
- Coloco un loading provisional
- Creo la carpeta components/movies
- El ScrollView, a diferencia del FlatList, va a renderizar los elementos pese a que no estén en pantalla
    - Horizontal para que el scroll sea de manera horizontal
    - ShowHorizontalScrollIndicator en false para que no aparezca una barra de progreso
- Hago un map de las movies. Recuerda que para renderizar debo pasarle un key. Le paso movie.id

```
import React from 'react'
import { Text, View } from 'react-native'
import { Movie } from '../../../core/entities/movie.entity'
import { ScrollView } from 'react-native-gesture-handler'

interface Props{
    movies: Movie[]
    height?: number
}

const PosterCarousel = ({movies, height= 440}: Props) => {
  return (
    <View style={{height}} >
        <ScrollView
        horizontal
        showsHorizontalScrollIndicator ={false}>
            {movies.map(movie=><Text key={movie.id}>{movie.title}</Text>)}

        </ScrollView>
    </View>
  )
}

export default PosterCarousel
```

- En vez de mostrar un texto voy a crear un componente para renderizar el poster en components/movies
- En el View esparzo el styles porque quiero agregar el width de manera provisional

```
import React from 'react'
import { Image, View } from 'react-native'
import { Movie } from '../../../core/entities/movie.entity'
import {StyleSheet} from 'react-native'


interface Props{
    movie: Movie
}

const Poster = ({movie}: Props) => {
  return (
    <View style={{...styles.imageContainer, width: 300}} >
        <Image style={styles.image} source={{uri: movie.poster}} />
    </View>
  )
}



const styles = StyleSheet.create({
  image:{
    flex:1,
    borderRadius: 18
  },
  imageContainer:{
    flex:1,
    borderRadius: 18,
    shadowColor: "black",
    shadowOffset:{
        width:0,
        height: 10
    },
    shadowOpacity: 0.24,
    shadowRadius: 7,
    elevation: 9
  }
})

export default Poster
```

- Lo coloco en el Carrousel que coloco en el HomeScreen

```
import React from 'react'
import { Text, View } from 'react-native'
import { Movie } from '../../../core/entities/movie.entity'
```

```
import { ScrollView } from 'react-native-gesture-handler'
import Poster from './Poster'

interface Props{
    movies: Movie[]
    height?: number
}

const PosterCarousel = ({movies, height= 440}: Props) => {
  return (
    <View style={{height}} >
        <ScrollView
        horizontal
        showsHorizontalScrollIndicator ={false}>
            {movies.map(movie=> <Poster key={movie.id} movie={movie}  />)}
        </ScrollView>
    </View>
  )
}

export default PosterCarousel
```

- Le paso el nowPlaying en HomeScreen

```
import React from 'react'
import { Text, View } from 'react-native'
import useMovies from '../../hooks/useMovies'
import { ScrollView } from 'react-native-gesture-handler'
import { useSafeAreaInsets } from 'react-native-safe-area-context'
import PosterCarousel from '../../components/movies/PosterCarousel'


export const HomeScreen  = () => {

  const {top} = useSafeAreaInsets()

const {isLoading, nowPlaying} = useMovies()

if(isLoading){
  return (<Text>Is loading...</Text>)
}


  return (
    <ScrollView>
      <View style={{marginTop: top + 20, paddingBottom: 30}}>

      <PosterCarousel movies={nowPlaying} />
      </View>
    </ScrollView>
  )
}
```

Terminar Carrousel y Navegación

- Quiero poder navegar a DetailsScreen
- Añado un par de props a Poster para hacerlo más flexible. Las hago opcionales y les coloco valores por defecto
- Para poder navegar a Details coloco el View dentro de un Pressable
- Hago el tipado estricto del useNavigation
- El componente Details me pide el movie.id (como tipé en RootParams), se lo paso como segundo parametro dentro de un objeto al navigation

```
import React from 'react'
import { Image, Pressable, View } from 'react-native'
import { Movie } from '../../../core/entities/movie.entity'
import { NavigationProp, useNavigation } from '@react-navigation/native'
import { RootStackParams } from '../../navigation/Navigation'
import {StyleSheet} from 'react-native'


interface Props{
    movie: Movie
    height?: number
    width?: number
}

const Poster = ({movie, height=420, width=300}: Props) => {

    const navigation = useNavigation<NavigationProp<RootStackParams>>()

  return (
    <Pressable onPress={()=>{navigation.navigate('Details', {movieId: movie.id})}}
>
        <View style={{...styles.imageContainer, width:300}} >
            <Image style={styles.image} source={{uri: movie.poster}} />
        </View>
    </Pressable>
  )
}



const styles = StyleSheet.create({
  image:{
    flex:1,
    borderRadius: 18
  },
  imageContainer:{
    flex:1,
    borderRadius: 18,
```

```
    shadowColor: "black",
    shadowOffset:{
        width:0,
        height: 10
    },
    shadowOpacity: 0.24,
    shadowRadius: 7,
    elevation: 9
  }
})

export default Poster
```

- Le añado el width y el height al Pressable.
- Quiero añadirle opacidad, por lo que en lugar de doble llave dentro del style, coloco solo un par de llaves y dentro, con una funcion de retorno implícito coloco el width y el height.
    - Con esta función ahora puedo desestructurar el pressed para saber si se está presionando
    - Hago un ternario para añadir la opacidad

```
import React from 'react'
import { Image, Pressable, View } from 'react-native'
import { Movie } from '../../../core/entities/movie.entity'
import { NavigationProp, useNavigation } from '@react-navigation/native'
import { RootStackParams } from '../../navigation/Navigation'
import {StyleSheet} from 'react-native'


interface Props{
    movie: Movie
    height?: number
    width?: number
}

const Poster = ({movie, height=420, width=300}: Props) => {

    const navigation = useNavigation<NavigationProp<RootStackParams>>()

  return (
    <Pressable onPress={()=>{navigation.navigate('Details', {movieId: movie.id})}}
    style={({pressed})=>({
        width,
        height,
        marginHorizontal:4,
        paddingBottom: 20,
        paddingHorizontal:5,
        opacity: pressed? 0.9: 1
    })}
    >
        <View style={{...styles.imageContainer, width:300}} >
            <Image style={styles.image} source={{uri: movie.poster}} />
        </View>
```

```
      </Pressable>
    )
}



const styles = StyleSheet.create({
  image:{
    flex:1,
    borderRadius: 18
  },
  imageContainer:{
    flex:1,
    borderRadius: 18,
    shadowColor: "black",
    shadowOffset:{
        width:0,
        height: 10
    },
    shadowOpacity: 0.24,
    shadowRadius: 7,
    elevation: 9
  }
})

export default Poster
```

- Estoy usando un ScrollView porque me interesa que todas las imágenes estén ya creadas
- Con el FlatList se renderizarían de manera dinámica, y es lo que vamos a hacer con los otros listados

---

Carrousel de películas con FlatList

- Vamos a reutilizar mucho nuestro código
- Quiero mostrar en HomeScreen las péliculas populares
- Extraigo popularMovies del hook useMovies
- Creo components/HorizontalCarousel.tsx
- En un FlatList siempre le paso la data (que es un arreglo de movies), el renderItem (de la que desestructuro item y con un return implicito poniendo paréntesis renderizo el componente Poster), y también debo pasarle el keyExtractor, que siempre debe ser un string
- Quito el width y height que había colocado en duro en el container de la imagen de Poster, ya que se lo paso por props

```
import React from 'react'
import { Text, View } from 'react-native'
import { Movie } from '../../../core/entities/movie.entity'
import { FlatList } from 'react-native-gesture-handler'
import Poster from './Poster'

interface Props{
```

```
    movies: Movie[]
    title?: string
}

export const HorizontalCarousel  = ({movies, title}: Props) => {
  return (
    <View style={{height: title? 260: 220}} >
      {
        title && (
            <Text style={{fontSize: 30, fontWeight: '400', marginLeft: 10,
marginBottom: 10}} >
                {title}
            </Text>
        )
      }

      <FlatList
      data={movies}
      renderItem={({item})=>(<Poster movie={item} width={140} height={200} />)}
      keyExtractor={item=> item.id.toString()}
      horizontal
      showsHorizontalScrollIndicator={false}
      />
    </View>
  )
}
```

- Uso el mismo componente para las top-rated y las upcoming

```
import React from 'react'
import { Text, View } from 'react-native'
import useMovies from '../../hooks/useMovies'
import { ScrollView } from 'react-native-gesture-handler'
import { useSafeAreaInsets } from 'react-native-safe-area-context'
import PosterCarousel from '../../components/movies/PosterCarousel'
import { HorizontalCarousel } from '../../components/movies/HorizontalCarousel'


export const HomeScreen  = () => {

  const {top} = useSafeAreaInsets()

const {isLoading, nowPlaying, popularMovie, topRatedMovie, upcomingMovie} =
useMovies()

if(isLoading){
  return (<Text>Is loading...</Text>)
}


  return (
    <ScrollView>
```

```
      <View style={{marginTop: top + 20, paddingBottom: 30}}>

      <PosterCarousel movies={nowPlaying} />
      <HorizontalCarousel movies={popularMovie} title="Populars"/>
      <HorizontalCarousel movies={topRatedMovie} title="Top Rated"/>
      <HorizontalCarousel movies={upcomingMovie} title="Upcoming"/>
      </View>
    </ScrollView>
  )
}
```

- Enfoquémonos en customizar el infiniteScroll

Infinite Scroll Horizontal

- Para aplicar el scroll infinito necesito saber cuando mi FlatList está en el final o cerca del final
  - Para ello usaré la propiedad onScroll
  - Para sacar el tipado del evento pongo el cursor encima

```
const PosterCarousel = ({movies, height= 440}: Props) => {
  return (
    <View style={{height}} >
        <ScrollView
        horizontal
        showsHorizontalScrollIndicator ={false}
        onScroll={(event)=>} //coloco el cursor encima para saber de que tipo es
el evento
        >
            {movies.map(movie=> <Poster key={movie.id} movie={movie}  />)}
        </ScrollView>
    </View>
  )
}
```

- **NOTA:** El onScroll es en el **FlatList!!**
- Creo la función onScroll
- Extraigo de event.nativeEvent los valores
- Los imprimo en consola

```
import React from 'react'
import { NativeScrollEvent, NativeSyntheticEvent, Text, View } from 'react-native'
import { Movie } from '../../../core/entities/movie.entity'
import { FlatList } from 'react-native-gesture-handler'
import Poster from './Poster'
```

```
interface Props{
    movies: Movie[]
    title?: string
}

export const HorizontalCarousel  = ({movies, title}: Props) => {

  const onScroll = (event:NativeSyntheticEvent<NativeScrollEvent> )=>{
    const {contentOffset, layoutMeasurement, contentSize} = event.nativeEvent
    console.log({contentOffset, layoutMeasurement, contentSize})
  }

  return (
    <View style={{height: title? 260: 220}} >
      {
        title && (
            <Text style={{fontSize: 30, fontWeight: '400', marginLeft: 10,
marginBottom: 10}} >
                {title}
            </Text>
        )
      }

      <FlatList
      data={movies}
      renderItem={({item})=>(<Poster movie={item} width={140} height={200} />)}
      keyExtractor={item=> item.id.toString()}
      horizontal
      showsHorizontalScrollIndicator={false}
      onScroll={onScroll}
      />
    </View>
  )
}

~~~js
```

- Para cargar esas nuevas películas tengo que integrarlo en el flujo que ya tengo actualmente
- En **contentOffset** es el movimiento que muevo en x (y estará en 0 porque es scroll horizontal)
- El **contentSize** es el tamaño del elemento total
- El **LayoutMeasurement** es el tamaño del elemento
- Cuando hago scroll, estos dos últimos no varían
- Si llevo el carrito hasta el final tengo que en contentOffset.x 2480 y en contentSize.width 2960 (porque hay un padding)
- Le sumo 600 pixeles de gracia
- Si esto es mayor que el contentSize.width significará que llegamos al final
    - En caso contrario será false
- Con un if, si no he llegado al final mando un return para que no haga nada
- Pero si llega al final, debo cargar las siguientes películas
- Creo la funcion l**oadNextPage** en las Props

- Si loadNextPage tiene un valor, lo ejecuta

```
import React from 'react'
import { NativeScrollEvent, NativeSyntheticEvent, Text, View } from 'react-native'
import { Movie } from '../../../core/entities/movie.entity'
import { FlatList } from 'react-native-gesture-handler'
import Poster from './Poster'

interface Props{
    movies: Movie[]
    title?: string
    loadNextPage: ()=> void
}

export const HorizontalCarousel  = ({movies, title, loadNextPage}: Props) => {

  const onScroll = (event:NativeSyntheticEvent<NativeScrollEvent> )=>{
    const {contentOffset, layoutMeasurement, contentSize} = event.nativeEvent
    console.log({contentOffset, layoutMeasurement, contentSize})

    const isEndReached = (contentOffset.x + layoutMeasurement.width + 600) >=
contentSize.width

    if(!isEndReached) return

    loadNextPage && loadNextPage()

  }

  return (
    <View style={{height: title? 260: 220}} >
      {
        title && (
            <Text style={{fontSize: 30, fontWeight: '400', marginLeft: 10,
marginBottom: 10}} >
                {title}
            </Text>
        )
      }

      <FlatList
      data={movies}
      renderItem={({item})=>(<Poster movie={item} width={140} height={200} />)}
      keyExtractor={item=> item.id.toString()}
      horizontal
      showsHorizontalScrollIndicator={false}
      onScroll={onScroll}
      />
    </View>
  )
}
```

- En HomeScreen le coloco un console.log al loadNextPage del componente HorizontalCarousel

```
import React from 'react'
import { Text, View } from 'react-native'
import useMovies from '../../hooks/useMovies'
import { ScrollView } from 'react-native-gesture-handler'
import { useSafeAreaInsets } from 'react-native-safe-area-context'
import PosterCarousel from '../../components/movies/PosterCarousel'
import { HorizontalCarousel } from '../../components/movies/HorizontalCarousel'


export const HomeScreen  = () => {

  const {top} = useSafeAreaInsets()

const {isLoading, nowPlaying, popularMovie, topRatedMovie, upcomingMovie} =
useMovies()

if(isLoading){
  return (<Text>Is loading...</Text>)
}


  return (
    <ScrollView>
      <View style={{marginTop: top + 20, paddingBottom: 30}}>

      <PosterCarousel movies={nowPlaying} />
      <HorizontalCarousel movies={popularMovie} title="Populars" loadNextPage=
{()=>console.log("end reached!")} />
      <HorizontalCarousel movies={topRatedMovie} title="Top Rated" loadNextPage=
{()=>console.log("end reached!")}/>
      <HorizontalCarousel movies={upcomingMovie} title="Upcoming" loadNextPage=
{()=>console.log("end reached!")}/>
      </View>
    </ScrollView>
  )
}
```

- Hago uso del useRef para determinar cuando llamo a loadNextPage porque si no se va a disparar muchas veces
- Cuando se que voy a cargar algo pongo el .current a true
- Antes de ejecutar el código verifico que .current no esté en true, porque significa que estoy haciendo la petición de las sigueintes películas
- Para regresarlo a false podemos hacer uso del useEffect. Cuando las movies cambien pono el .current a false
- Uso setTimeOut para darle unas milésimas de segundo y no vaya tan rápido pues produce un efecto no deseado (opcional)

```
import React, { useEffect, useRef } from 'react'
import { NativeScrollEvent, NativeSyntheticEvent, Text, View } from 'react-native'
import { Movie } from '../../../core/entities/movie.entity'
import { FlatList } from 'react-native-gesture-handler'
import Poster from './Poster'

interface Props{
    movies: Movie[]
    title?: string
    loadNextPage?: ()=> void
}

export const HorizontalCarousel  = ({movies, title, loadNextPage}: Props) => {

  const isLoading = useRef(false)

  useEffect(()=>{

    setTimeout(()=>{
      isLoading.current = false
    }, 200)

  }, [movies])

  const onScroll = (event:NativeSyntheticEvent<NativeScrollEvent> )=>{

    if(isLoading.current) return

    const {contentOffset, layoutMeasurement, contentSize} = event.nativeEvent
    console.log({contentOffset, layoutMeasurement, contentSize})

    const isEndReached = (contentOffset.x + layoutMeasurement.width + 600) >=
contentSize.width

    if(!isEndReached) return

    isLoading.current = true

    loadNextPage && loadNextPage()

  }

  return (
    <View style={{height: title? 260: 220}} >
      {
        title && (
            <Text style={{fontSize: 30, fontWeight: '400', marginLeft: 10,
marginBottom: 10}} >
                {title}
            </Text>
        )
      }
```

```
        <FlatList
        data={movies}
        renderItem={({item})=>(<Poster movie={item} width={140} height={200} />)}
        keyExtractor={item=> item.id.toString()}
        horizontal
        showsHorizontalScrollIndicator={false}
        onScroll={onScroll}
        />
      </View>
    )
}
```

- Si quisiera cargar un spinner o algo para decir que está cargando si debería hacerlo con un state y no con un ref
- Ref no dispara rerenders cuando cambia su valor

Infinite Scroll parte 2

- HorizontalCarousel no está llamadno a las nuevas películas, solo a la función si es que la tiene
- Para cargar las siguientes películas, la lógica está en el custom hook useMovies
- Creo una variable popularPage y la inicializo en 1
- En la función incremento la página y llamo al caso de uso
- Debo añadir el argumento page al caso de uso

```
import React, { useEffect, useState } from 'react'
import { Movie } from '../../core/entities/movie.entity'
import * as UseCases from '../../core/use-cases'
import { MovieDBFetcher } from '../../config/adapters/http/movieDB.adapter'


let popularPage = 1

const useMovies = () => {



    const [isLoading, setIsLoading] = useState(true)
    const [nowPlaying, setNowPlaying] = useState<Movie[]>([])
    const [popularMovie, setpopularMovie] = useState<Movie[]>([])
    const [topRatedMovie, setTopRatedMovie] = useState<Movie[]>([])
    const [upcomingMovie, setUpcomingMovie] = useState<Movie[]>([])

    const popularNextPage = async () =>{
         popularPage ++;

         const popularMovies = await UseCases.PopularUseCase(MovieDBFetcher)
    }

    useEffect(()=>{
```

```
        initLoad()
    },[])

    const initLoad= async()=>{
        const nowPlayingPromise = UseCases.moviesNowPlayingUseCase(MovieDBFetcher)
        const popularPromise = UseCases.PopularUseCase(MovieDBFetcher)
        const topRatedPromise = UseCases.TopRatedUseCase(MovieDBFetcher)
        const upcomingPromise = UseCases.moviesUpcomingUseCase(MovieDBFetcher)

        const[
            nowPlayingMovies,
            popularMovies,
            topRatedMovies,
            upcomingMovies
        ] = await Promise.all([nowPlayingPromise, popularPromise, topRatedPromise,
upcomingPromise])


        setNowPlaying(nowPlayingMovies)
        setpopularMovie(popularMovies)
        setTopRatedMovie(topRatedMovies)
        setUpcomingMovie(upcomingMovies)
        setIsLoading(false)


    }

    return {
        isLoading,
        nowPlaying,
        popularMovie,
        topRatedMovie,
        upcomingMovie,
        popularNextPage
    }
}

export default useMovies
```

- En el caso de uso de popular, agrego una interfaz para el objeto options y poder pasarle la página y el limit

```
import { HttpAdapter } from "../../../config/adapters/http/http.adapter";
import { MovieDBResponse} from
"../../../infraestructure/interfaces/movieDBResponses";
import { MovieMapper } from "../../../infraestructure/mappers/movie.mapper";
import { Movie } from "../../entities/movie.entity";

interface Options{
    page?: number
    limit?: number
```

```
}


export const PopularUseCase = async (fetcher: HttpAdapter, options?: Options):
Promise<Movie[]>=>{

        const popularMoviesResult = await fetcher.get<MovieDBResponse>
('/3/tv/popular', {
            params:{
                page: options?.page ?? 1
            }
        })

    return
popularMoviesResult.results.map(MovieMapper.fromMovieDBResultToEntity)

}
```

- Ahora, en el hook, añado el argumento page al caso de uso
- Esparzo con el spread el state anterior y el nuevo

```
import React, { useEffect, useState } from 'react'
import { Movie } from '../../core/entities/movie.entity'
import * as UseCases from '../../core/use-cases'
import { MovieDBFetcher } from '../../config/adapters/http/movieDB.adapter'


let popularPage = 1

const useMovies = () => {



    const [isLoading, setIsLoading] = useState(true)
    const [nowPlaying, setNowPlaying] = useState<Movie[]>([])
    const [popularMovie, setpopularMovie] = useState<Movie[]>([])
    const [topRatedMovie, setTopRatedMovie] = useState<Movie[]>([])
    const [upcomingMovie, setUpcomingMovie] = useState<Movie[]>([])

    const popularNextPage = async () =>{
         popularPage ++;

         const popularMovies = await UseCases.PopularUseCase(MovieDBFetcher,
{page: popularPage})

         setpopularMovie(prev=> [...prev, ... popularMovies])
    }

    useEffect(()=>{
        initLoad()
    },[])
```

```
    const initLoad= async()=>{
        const nowPlayingPromise = UseCases.moviesNowPlayingUseCase(MovieDBFetcher)
        const popularPromise = UseCases.PopularUseCase(MovieDBFetcher)
        const topRatedPromise = UseCases.TopRatedUseCase(MovieDBFetcher)
        const upcomingPromise = UseCases.moviesUpcomingUseCase(MovieDBFetcher)

        const[
            nowPlayingMovies,
            popularMovies,
            topRatedMovies,
            upcomingMovies
        ] = await Promise.all([nowPlayingPromise, popularPromise, topRatedPromise,
upcomingPromise])


        setNowPlaying(nowPlayingMovies)
        setpopularMovie(popularMovies)
        setTopRatedMovie(topRatedMovies)
        setUpcomingMovie(upcomingMovies)
        setIsLoading(false)


    }

    return {
        isLoading,
        nowPlaying,
        popularMovie,
        topRatedMovie,
        upcomingMovie,
        popularNextPage
    }
}

export default useMovies
```

- Ahora puedo desestructurar este popularNextPage del hook y llamarlo

```
import React, { useEffect, useRef } from 'react'
import { NativeScrollEvent, NativeSyntheticEvent, Text, View } from 'react-native'
import { Movie } from '../../../core/entities/movie.entity'
import { FlatList } from 'react-native-gesture-handler'
import Poster from './Poster'

interface Props{
    movies: Movie[]
    title?: string
    loadNextPage?: ()=> void
}

export const HorizontalCarousel  = ({movies, title, loadNextPage}: Props) => {
```

```
  const isLoading = useRef(false)

  useEffect(()=>{

    setTimeout(()=>{
      isLoading.current = false
    }, 200)

  }, [movies])

  const onScroll = (event:NativeSyntheticEvent<NativeScrollEvent> )=>{

    if(isLoading.current) return

    const {contentOffset, layoutMeasurement, contentSize} = event.nativeEvent
    console.log({contentOffset, layoutMeasurement, contentSize})

    const isEndReached = (contentOffset.x + layoutMeasurement.width + 600) >=
contentSize.width

    if(!isEndReached) return

    isLoading.current = true

    loadNextPage && loadNextPage()

  }

  return (
    <View style={{height: title? 260: 220}} >
      {
        title && (
            <Text style={{fontSize: 30, fontWeight: '400', marginLeft: 10,
marginBottom: 10}} >
                {title}
            </Text>
        )
      }

      <FlatList
      data={movies}
      renderItem={({item})=>(<Poster movie={item} width={140} height={200} />)}
      keyExtractor={item=> item.id.toString()}
      horizontal
      showsHorizontalScrollIndicator={false}
      onScroll={onScroll}
      />
    </View>
  )
}
```

- Está dando problemas de id's duplicados

- Podemos verificar que en nuestro arreglo de pelis populares los id's sean únicos
- También puedo generar un id único con el index en el keyExtractor
- HorizontalCarousel.tsx

```
<FlatList
data={movies}
renderItem={({item})=>(<Poster movie={item} width={140} height={200} />)}
keyExtractor={(item, index)=> `${item.id}-${index}`}
horizontal
showsHorizontalScrollIndicator={false}
onScroll={onScroll}
/>
```

Información Película por ID

- Para hacer peticiones http de una película, como cada vez que entre a la misma película voy a ahcer la petición, es conveniente almacenar la info en caché. Eso es lo que hace React Query, se puede usar en React Native
- Necesitamos el id de la película
- En DetailsScreen tomo los params de useRoute

```
import { useRoute } from '@react-navigation/native'
import React from 'react'
import { Text, View } from 'react-native'

export const Detailsscreen  = () => {

  const {movieId} = useRoute().params

  console.log(movieId)

  return (
    <View>
      <Text>Hello World</Text>
    </View>
  )
}
```

- De esta manera funcion (aunque debo tipar el useRoute)
- Hay otra forma. Tipando las props, dispongo de navigation y route

```
import { useRoute } from '@react-navigation/native'
import { StackScreenProps } from '@react-navigation/stack'
import React from 'react'
import { Text, View } from 'react-native'
import { RootStackParams } from '../../navigation/Navigation'
```

```
interface Props extends StackScreenProps<RootStackParams, 'Details'>{}

export const Detailsscreen  = ({route}:Props) => {

  const {movieId} = route.params

  return (
    <View>
      <Text>Hello World</Text>
    </View>
  )
}
```

- Creo un custom hook useMovie
- Haremos algo parecido al useMovies. Apenas se carga lanzamos un efecto para realizar la petición http
    - Podriamos poner de dependencia el movieId o dejar el arreglo vacío del useEffect

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

const useMovie = (movieId: number) => {
  const [isLoading, setIsLoading] = useState(true)

  useEffect(()=>{
    loadMovie()

},[movieId])

  const loadMovie =()=>{

  }

    return (
    isLoading

  )
}

export default useMovie
```

- Si hago un llamado a un endpoint y detrás de movie coloco el id .../3/movie/*aqui_coloco_ID* me trae mucha info
- En la entidad creo una interfaz que extiende de Movie

```
export interface Movie{
    id: number
    title: string
    description: string
    releaseDate: Date
```

```
    rating: number
    poster: string
    backdrop: string
}

export interface FullMovie extends Movie{
    genres: string[]
    duration: number
    budget: number
    originalTitle: string
    productionCompanies: string[]
}
```

- Debo crear el caso de uso GetMovie y el mapper para mapear la data
- Del endpoint extraigo la interfaz para tipar la respuesta del fetcher en el caso de uso

```
// Generated by https://quicktype.io

export interface IFullMovie {
    adult:                 boolean;
    backdrop_path:         null;
    belongs_to_collection: null;
    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:  any[];
    production_countries:  ProductionCountry[];
    release_date:          string;
    revenue:               number;
    runtime:               number;
    spoken_languages:      SpokenLanguage[];
    status:                string;
    tagline:               string;
    title:                 string;
    video:                 boolean;
    vote_average:          number;
    vote_count:            number;
}

export interface Genre {
    id:   number;
    name: string;
}
```

```
export interface ProductionCountry {
    iso_3166_1: string;
    name:       string;
}

export interface SpokenLanguage {
    english_name: string;
    iso_639_1:    string;
    name:         string;
}
```

- Hago el mapper.
- Hago un .map de los genres

```
import { FullMovie } from "../../core/entities/movie.entity"
import { MovieDBMovie } from "../interfaces/full-movie.interface"

export class FullMovieMapper {

    static fromMovieDBToEntity(result: MovieDBMovie): FullMovie{

        return{id: result.id,
        title: result.title,
        description: result.overview,
        releaseDate: new Date(result.release_date),
        rating: result.vote_average,
        poster: `https://image.tmdb.org/t/p/w500${result.poster_path}`,
        backdrop: `https://image.tmdb.org/t/p/w500${result.backdrop_path}`,
        genres: result.genres.map(genre=>genre.name),
        duration: result.runtime,
        budget: result.budget,
        originalTitle: result.original_title,
        productionCompanies:
result.production_companies.map(company=>company.name)}
    }
}
```

- Voy al caso de uso, utilizo un try catch
- Le paso el id, se lo paso como argumento y lo coloco con un template string
- Uso el mapper y retorno el resultado

```
import { HttpAdapter } from "../../../config/adapters/http/http.adapter";
import { IFullMovie } from "../../../infraestructure/interfaces/full-
movie.interface";
import { FullMovieMapper } from "../../../infraestructure/mappers/full-
movie.mapper";
import { FullMovie } from "../../entities/movie.entity";

export const getMovieUseCase = async (fetcher: HttpAdapter, id: number):
```

```
Promise<FullMovie>=>{

    try {
        const getFullMovie = await fetcher.get<IFullMovie>(`/3/movie/${id}`)

        const fullMovie = FullMovieMapper.fromMovieDBToEntity(getFullMovie)

        return fullMovie

    } catch (error) {
        throw new Error("Can't get full movie")
    }

}
```

- Puedo ir al hook, antes debo crear el adaptador

```
import { AxiosAdapter } from "./axios.adapter";

const fullMovieAdapter = new AxiosAdapter({
    baseURL: "https://api.themoviedb.org",
    params: {
        api_key: "",
        language: 'es'
    }
})
```

- Exporto el caso de uso desde el archivo de barril y voy al hook useMovie
- Desde el useEffect disparo la función
- En la función cambio el isLoading a true
- Hago uso del caso de uso, le paso el adaptador y el id (que tendré en DetailsScreen)
- Pongo el isLoading en false y retorno en el objeto los dos estados

```
import React, { useEffect, useState } from 'react'
import * as UseCase from '../../core/use-cases'
import { FullMovie } from '../../core/entities/movie.entity'
import { fullMovieAdapter } from '../../config/adapters/http/fullMovie.adapter'

const useMovie = (movieId: number) => {
  const [isLoading, setIsLoading] = useState(true)
  const [fullMovie, setFullMovie] = useState<FullMovie>()

  useEffect(()=>{
    loadMovie()

},[movieId])

  const loadMovie = async()=>{
    setIsLoading(true)
```

```
    const fullMovieResult = await UseCase.getMovieUseCase(fullMovieAdapter,
movieId)
    setFullMovie(fullMovieResult)
    setIsLoading(false)
  }

    return {
      isLoading,
      fullMovie
    }
}

export default useMovie
```

- Ahora puedo ir a DetailsScreen y desestructurar el state del hook
- Renderizo el título en pantalla
- Ahora ya tengo la info, solo es hacer un poco de carpíntería

```
import { StackScreenProps } from '@react-navigation/stack'
import React from 'react'
import { Text, View } from 'react-native'
import { RootStackParams } from '../../navigation/Navigation'
import useMovie from '../../hooks/useMovie'

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

export const Detailsscreen  = ({route}:Props) => {

  const {movieId} = route.params

  const{fullMovie} = useMovie(movieId)

  return (
    <View>
      <Text>{fullMovie?.title}</Text>
    </View>
  )
}
```

Pantalla de detalles - Header

- Creo en components/movies/movie/movieHeader
- Utilizo useWindowsDimensions porque no sé que dispositivo va a usar la persona, para que sea adaptable
- Renombro el height a screenHeight
- Esparzo con spread los estilos, multiplico el screenHeight por 0.7 para obtener el 70% de la pantalla
- Uso el useNavigation

```
import React from 'react'
import { Image, StyleSheet, Text, View, useWindowDimensions } from 'react-native'
import { FullMovie } from '../../../../core/entities/movie.entity'
import { useNavigation } from '@react-navigation/native'

interface Props{
    movie: FullMovie
}

export const MovieHeader  = ({movie}: Props) => {


    const {height: screenHeight}= useWindowDimensions()
    const navigation = useNavigation()



  return (
    <>
        <View style={{...styles.imageContainer, height: screenHeight * 0.7}}>
            <View style={styles.imageBorder} >
                <Image
                    style={styles.posterImage}
                    source={{uri: movie.poster}}
                />
            </View>

        </View>
    </>
  )
}


const styles = StyleSheet.create({
    imageContainer: {
      width: '100%',
      shadowColor: '#000',
      shadowOffset: {
        width: 0,
        height: 10,
      },
      shadowOpacity: 0.24,
      shadowRadius: 7,

      elevation: 9,
      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',
    },
    backButton: {
      position: 'absolute',
      zIndex: 999,
      elevation: 9,
      top: 35,
      left: 10,
    },
    backButtonText: {
      color: 'white',
      fontSize: 25,
      fontWeight: 'bold',
      textShadowColor: 'rgba(0, 0, 0, 0.55)',
      textShadowOffset: {width: -1, height: 1},
      textShadowRadius: 10,
    },
  });
```

- Lo coloco en DetailsScreen y le paso la prop movie
- Me dice que movie puede ser undefined.
- Desestructuro el isLoading, lo uso para verificar que si está en true devuelva un texto "Loading..."
- Le indico a typescript que si o si vendrá una movie con !

```
import { StackScreenProps } from '@react-navigation/stack'
import React from 'react'
import { Text, View } from 'react-native'
import { RootStackParams } from '../../navigation/Navigation'
import useMovie from '../../hooks/useMovie'
import { MovieHeader } from '../../components/movies/movie/MovieHeader'

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

export const Detailsscreen  = ({route}:Props) => {
```

```
  const {movieId} = route.params

  const{isLoading, fullMovie} = useMovie(movieId)

  if(isLoading) {
    return <Text>Loading</Text>
  }
  return (
    <View>

      <MovieHeader movie={fullMovie!} />
    </View>
  )
}
```

- Sería una buena práctica pasarle por props una movie con solo lo que necesito: poster_image y pocas cosas más

```
interface Props{
    poster: string
    originalTitle: string
    title: string
}

export const MovieHeader  = ({poster, originalTitle, title}: Props) => {


    const {height: screenHeight}= useWindowDimensions()
    const navigation = useNavigation()

{...}
```

- Ahora en DetailsScreen le paso las props necesarias
- salto el error de TypeScript de que movie sea undefined chequeando el isLoading, movie siempre estará. Lo indico con !

```
import { StackScreenProps } from '@react-navigation/stack'
import React from 'react'
import { Text, View } from 'react-native'
import { RootStackParams } from '../../navigation/Navigation'
import useMovie from '../../hooks/useMovie'
import { MovieHeader } from '../../components/movies/movie/MovieHeader'

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

export const Detailsscreen  = ({route}:Props) => {

  const {movieId} = route.params
```

```
  const{isLoading, movie} = useMovie(movieId)


  if(isLoading) {
    return <Text>Loading</Text>
  }
  return (
    <View>
      <MovieHeader title={movie!.title} originalTitle={movie!.title} poster=
{movie!.poster} />
    </View>
  )
}
```

Detalles de la película

- Creo en movie/MovieDetails.tsx
- Mostramos el rating.
- Cómo ya se ha hecho un map de los generos, puedo hacer un join para unirlos con una coma

```
import React from 'react'
import { Text, View } from 'react-native'
import { FullMovie } from '../../../../core/entities/movie.entity'

interface Props{
    movie: FullMovie
}

export const MovieDetails= ({movie}: Props) => {
  return (
    <>
    <View style={{marginHorizontal: 20}}>
      <View style={{flexDirection: 'row'}}>
            <Text>{movie.rating}</Text>
            <Text style={{marginLeft: 5}}>
                - {movie.genres.join(', ')}
             </Text>

      </View>
    </View>

    <Text style={{fontSize: 23, marginTop:10, fontWeight:'bold'}} >
        Historia
    </Text>

    <Text style={{fontSize: 16}}>{movie.description}</Text>

    </>
```

```
  )
}
```

- Coloco un ScrollView para poder hacer scroll

```
import { StackScreenProps } from '@react-navigation/stack'
import React from 'react'
import { Text, ScrollView } from 'react-native'
import { RootStackParams } from '../../navigation/Navigation'
import useMovie from '../../hooks/useMovie'
import { MovieHeader } from '../../components/movies/movie/MovieHeader'
import { MovieDetails } from '../../components/movies/movie/MovieDetails'


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

export const Detailsscreen  = ({route}:Props) => {

  const {movieId} = route.params

  const{isLoading, movie} = useMovie(movieId)


  if(isLoading) {
    return <Text>Loading</Text>
  }
  return (
    <ScrollView>
      <MovieHeader title={movie!.title} originalTitle={movie!.title} poster=
{movie!.poster} />
      <MovieDetails movie={movie!} />
    </ScrollView>
  )
}
```

- Sigo con los detalles en MovieDetails
- Muestro la descripción
- Para mostrar el presupuesto voy a formatear la cifra con un helper

```
export class Formatter{
    public static currency(value: number): string | string[] | undefined {

        return new Intl.NumberFormat('en-US',{
            style: 'currency',
            currency: 'USD'
        }).format(value)
    }
}
```

- Lo aplico al presupuesto

```
import React from 'react'
import { Text, View } from 'react-native'
import { FullMovie } from '../../../../core/entities/movie.entity'
import { Formatter } from '../../../../config/helpers/formatter'

interface Props{
    movie: FullMovie
}

export const MovieDetails= ({movie}: Props) => {
  return (
    <>
    <View style={{marginHorizontal: 20}}>
      <View style={{flexDirection: 'row'}}>
            <Text>{movie.rating}</Text>
            <Text style={{marginLeft: 5}}>
                - {movie.genres.join(', ')}
             </Text>

      </View>
    </View>

    <Text style={{fontSize: 23, marginTop:10, marginBottom: 5, fontWeight:'bold'}}
>
        Historia
    </Text>
    <Text style={{fontSize: 16, marginBottom: 20}}>{movie.description}</Text>

    <Text style={{fontSize: 23, marginTop: 10, fontWeight:'bold'}} >
        Presupuesto
    </Text>
    <Text style={{fontSize: 16, marginBottom: 35}}>
{Formatter.currency(movie.budget)}</Text>
    </>
  )
}
```

- Ahora quiero colocar los actores

Estructura de datos para los actores

- Nuevo caso de uso get-cast
- Para obtener el casting de actores y actrices debo añadir credits después del id, antes del api_key
- Obtenemos el cast y el crew, nos interesa el cast
- Hago el mismo proceso: saco la interfaz, creo la entity, el método en el mapper, el caso de uso, lo llamo en el hook
- Interfaz

```
export interface MovieDBCastResponse {
    id:   number;
    cast: MovieDBCast[];
    crew: MovieDBCast[];
}

export interface MovieDBCast {
    adult:                boolean;
    gender:               number;
    id:                   number;
    known_for_department: string;
    name:                 string;
    original_name:        string;
    popularity:           number;
    profile_path:         null | string;
    cast_id?:             number;
    character?:           string;
    credit_id:            string;
    order?:               number;
    department?:          string;
    job?:                 string;
}
```

- La entidad

```
export interface Cast{
    id: number
    name: string
    character: string
    avatar: string
}
```

- Creo un mapper para el cast

```
import { Cast } from "../../core/entities/movie.entity";
import { MovieDBCast} from "../interfaces/full-movie.interface";

export class CastMapper {
    public static fromMovieDBToEntity(result: MovieDBCast): Cast{

            return{
                id: result.id,
                name: result.name,
                character: result.character ?? 'No character',
                avatar: result.profile_path
                ? `https://image.tmdb.org/t/p/w500${result.profile_path}`
                : 'https://i.stack.imgur.com/l60Hf.png'
            }
```

```
    }
}
```

- En el caso de uso

```
import { HttpAdapter } from "../../../config/adapters/http/http.adapter";
import { MovieDBCastResponse } from "../../../infraestructure/interfaces/full-
movie.interface";
import { CastMapper } from "../../../infraestructure/mappers/cast.mapper";
import { Cast } from "../../entities/movie.entity";

export const getMovieCastUseCase = async (fetcher: HttpAdapter, movieId: number):
Promise<Cast[]>=>{
    try {
        const {cast} = await fetcher.get<MovieDBCastResponse>
(`/3/movie/${movieId}/credits`)

        const actors = cast.map((actor)=> CastMapper.fromMovieDBToEntity(actor))
        return actors

    } catch (error) {
        throw new Error('Can´t get movie cast')
    }
}
```

- Voy al hook

```
import React, { useEffect, useState } from 'react'
import * as UseCase from '../../core/use-cases'
import { Cast, FullMovie } from '../../core/entities/movie.entity'
import { MovieDBFetcher } from '../../config/adapters/http/movieDB.adapter'

const useMovie = (movieId: number) => {
  const [isLoading, setIsLoading] = useState(true)
  const [movie, setMovie] = useState<FullMovie>()
  const [cast, setCast] = useState<Cast[]>()

  useEffect(()=>{
    loadMovie()
},[movieId])

  const loadMovie = async()=>{
    setIsLoading(true)

      const fullMoviePromise = UseCase.getMovieUseCase(MovieDBFetcher, movieId)
      const movieCastPromise = UseCase.getMovieCastUseCase(MovieDBFetcher,
movieId)
```

```
      const [fullMovie, movieCast] = await Promise.all([fullMoviePromise,
movieCastPromise])

      setMovie(fullMovie)
      setCast(movieCast)
      setIsLoading(false)

      console.log(cast)

  }

    return {
      isLoading,
      movie,
      cast
    }
}

export default useMovie
```

- Lo desestructuro del hook en DetailsScreen y se lo paso a MovieDetails

```
import React from 'react'
import { Text, View } from 'react-native'
import { Cast, FullMovie } from '../../../../core/entities/movie.entity'
import { Formatter } from '../../../../config/helpers/formatter'

interface Props{
    movie: FullMovie
    actors: Cast[]
}

export const MovieDetails= ({movie, actors}: Props) => {
  return (
    <>
    <View style={{marginHorizontal: 20}}>
      <View style={{flexDirection: 'row'}}>
            <Text>{movie.rating}</Text>
            <Text style={{marginLeft: 5}}>
                - {movie.genres.join(', ')}
             </Text>

      </View>
    </View>

    <Text style={{fontSize: 23, marginTop:10, marginBottom: 5, fontWeight:'bold',
marginHorizontal: 10}} >
        Historia
    </Text>
    <Text style={{fontSize: 16, marginBottom: 20, marginHorizontal: 10}}>
{movie.description}</Text>
```

```
    <Text style={{fontSize: 23, marginTop: 10, fontWeight:'bold',
marginHorizontal: 10}} >
        Presupuesto
    </Text>
    <Text style={{fontSize: 16, marginBottom: 35, marginHorizontal: 10}}>
{Formatter.currency(movie.budget)}</Text>

    <View>
        <Text style={{fontSize: 23, marginVertical: 10, fontWeight: 'bold',
marginHorizontal: 20}} >
            {actors.map(actor=> <Text key={actor.id}>{actor.name}</Text>)}
        </Text>
    </View>
    </>
  )
}
```

Mostrar actores en pantalla

- Usemos un FlatList en MovieDetails

```
    <FlatList
        data={actors}
        keyExtractor={(item)=>item.id.toString()}
        horizontal
        showsHorizontalScrollIndicator={false}
        renderItem={({item})=><Text>{item.name}</Text>}
        />
```

- En lugar de renderizar el texto, me creo un componente en components/actors/CastActor.tsx

```
import React from 'react'
import { Image, Text, View } from 'react-native'
import { Cast } from '../../../core/entities/movie.entity'

interface Props{
    actor: Cast
}

export const CastActors = ({actor}: Props) => {
  return (
    <View style={styles.container} >
      <Image source={{uri: actor.avatar}} style={{width:100, height: 100,
borderRadius: 10}} />

      <View style={styles.actorInfo}>
        <Text style={{fontSize: 15, fontWeight: 'bold'}} >
            {actor.name}
```

```
          </Text>
          <Text style={{fontSize: 12, opacity: 0.7}} >
              {actor.character}
          </Text>

        </View>
      </View>
    )
}



import {StyleSheet} from 'react-native'

const styles = StyleSheet.create({
  container:{
    marginRight:10,
    paddingLeft: 10,
    display: 'flex',
    flexDirection: 'column',
    width: 100
  },
  actorInfo:{
    marginLeft: 10,
    marginTop: 4
  }
})
```

- Lo uso en MovieDetails

```
import React from 'react'
import { FlatList, Text, View } from 'react-native'
import { Cast, FullMovie } from '../../../../core/entities/movie.entity'
import { Formatter } from '../../../../config/helpers/formatter'
import { CastActors } from '../../actors/CastActors'

interface Props{
    movie: FullMovie
    actors: Cast[]
}

export const MovieDetails= ({movie, actors}: Props) => {
  return (
    <>
    <View style={{marginHorizontal: 20}}>
      <View style={{flexDirection: 'row'}}>
            <Text>{movie.rating}</Text>
            <Text style={{marginLeft: 5}}>
                - {movie.genres.join(', ')}
             </Text>

      </View>
```

```
    </View>

    <Text style={{fontSize: 23, marginTop:10, marginBottom: 5, fontWeight:'bold',
marginHorizontal: 10}} >
        Historia
    </Text>
    <Text style={{fontSize: 16, marginBottom: 20, marginHorizontal: 10}}>
{movie.description}</Text>

    <Text style={{fontSize: 23, marginTop: 10, fontWeight:'bold',
marginHorizontal: 10}} >
        Presupuesto
    </Text>
    <Text style={{fontSize: 16, marginBottom: 35, marginHorizontal: 10}}>
{Formatter.currency(movie.budget)}</Text>

    <View>
        <Text style={{fontSize: 23, marginVertical: 10, fontWeight: 'bold',
marginHorizontal: 20}} >
           Actores
        </Text>

        <FlatList
        data={actors}
        keyExtractor={(item)=>item.id.toString()}
        horizontal
        showsHorizontalScrollIndicator={false}
        renderItem={({item})=><CastActors actor={item}  />}
        />
    </View>
    </>
  )
}
```

Configurar variables de entorno

- Usaremos react-native-dotenv

```
npm i -D react-native-dotenv
```

- En .babelrc o babel.config

```
{
  "plugins": [
    ["module: react-native-dotenv"]
  ]
}
```

- Quedando así

- **NOTA**: si usas un plugin de reanimated asegurate de quea el último

```
module.exports = {
  presets: ['module:@react-native/babel-preset'],
  plugins: ['module: react-native-dotenv']
};
```

- Se puede configurar

```
module.exports = {
  presets: ['module:@react-native/babel-preset'],
  plugins: [
    ['module:react-native-dotenv',{
    envName: 'APP_ENV',
    moduleName: '@env',
    path: ".env"
  }]
]
};
```

- En Typescript hay que crear una carpeta llamada Types y ahi definir un módulo
- En la raíz types/env.d.ts (.d son los archivos de definicion de TypeScript)
- Uso el nombre del módulo que especifiqué en babel.config

```
declare module '@env'{
    export const API_KEY: string
}
```

- Ahora para usarlo en el adaptador

```
import { API_KEY } from "@env";
import { AxiosAdapter } from "./axios.adapter";

export const fullMovieAdapter = new AxiosAdapter({
    baseURL: "https://api.themoviedb.org/3/movie",
    params: {
        api_key: API_KEY ?? 'no key',
        language: 'es'
    }
})
```

- Este paquete a veces DA PROBLEMAS. Prueba a reiniciar la computadora (aunque parezca inverosímil)

React Native - Más Componentes

- Están configurados los iconos de Ionicons
- Creo la estructura de directorios
- En src
- Presentation
    - Components
        - Ui
    - Assets
    - Hooks
    - Navigator
    - Icons
    - Screens
        - Home/HomeScreen
        - Alerts
        - Animations
        - Inputs
        - Switches
        - ThemeChanger
        - Ui
- Renombro App a MasComponentes
- Muevo MasComponentes dentro de src
- Creo el HomeScreen.tsx
- En la carpeta Icons creo Icons.tsx

```
import  Icon  from "react-native-vector-icons/Ionicons"

export const AirplaneIcon = ()=> <Icon  name='airplane-outline' size={30} />
```

- Para la navegación creo el Stack (la instalación ya está hecha)

```
import { createStackNavigator } from '@react-navigation/stack';
import { HomeScreen } from '../screens/home/HomeScreen';

const Stack = createStackNavigator();

export const StackNavigator =()=> {
  return (
    <Stack.Navigator>
      <Stack.Screen name="HomeScreen" component={HomeScreen} />

    </Stack.Navigator>
  );
}
```

- Coloco el StackNavigator dentro del NavigationContainer
- **NOTA:** Tengo configurado React Native Paper también

```
import 'react-native-gesture-handler'
import React from 'react';
import {PaperProvider} from 'react-native-paper'
import IonIcon from 'react-native-vector-icons/Ionicons'
import { HomeScreen } from './presentation/screens/home/HomeScreen';
import { NavigationContainer } from '@react-navigation/native';
import { StackNavigator } from './presentation/navigator/StackNavigator';

export const ComponentsApp=(): React.JSX.Element=>{

  return(
      <NavigationContainer>
      <PaperProvider
      settings={({
         icon: (props)=> <IonIcon {...props} />
      })}
      >
         <StackNavigator />

      </PaperProvider>
      </NavigationContainer>

   )
}
```

Menú Principal y Estilos

- Copio el menuItems del Gist adjunto con la lección en HomeScreen, luego le encontraremos un mejor lugar

> https://gist.github.com/Klerith/8cc5b908636c53ee91d2bdae7be0aa25

```
import React from 'react'
import { Text, View } from 'react-native'
import { AirplaneIcon } from '../../icons/Icons'

export const HomeScreen = () => {
  return (
    <View>
      <Text>HomeScreen</Text>

    </View>
  )
}

export const menuItems = [
```

```
    // 01-animationMenuItems
    {
      name: 'Animation 101',
      icon: 'cube-outline',
      component: 'Animation101Screen',
    },
    {
      name: 'Animation 102',
      icon: 'albums-outline',
      component: 'Animation102Screen',
    },


    // 02-menuItems
    {
      name: 'Pull to refresh',
      icon: 'refresh-outline',
      component: 'PullToRefreshScreen',
    },
    {
      name: 'Section List',
      icon: 'list-outline',
      component: 'CustomSectionListScreen',
    },
    {
      name: 'Modal',
      icon: 'copy-outline',
      component: 'ModalScreen',
    },
    {
      name: 'InfiniteScroll',
      icon: 'download-outline',
      component: 'InfiniteScrollScreen',
    },
    {
      name: 'Slides',
      icon: 'flower-outline',
      component: 'SlidesScreen',
    },
    {
      name: 'Themes',
      icon: 'flask-outline',
      component: 'ChangeThemeScreen',
    },

    // 03- uiMenuItems
    {
      name: 'Switches',
      icon: 'toggle-outline',
      component: 'SwitchScreen',
    },
    {
      name: 'Alerts',
      icon: 'alert-circle-outline',
```

```
    component: 'AlertScreen',
  },
  {
    name: 'TextInputs',
    icon: 'document-text-outline',
    component: 'TextInputScreen',
  },
];
```

- Creo config/theme/theme.tsx, pego lo que hay en el Gist
- Hay una interfaz para los colores
- He colocado los colores en duro en los objetos con *colors.text*

```
import { StyleSheet } from "react-native";

export interface ThemeColors {
  primary: string;
  text: string;
  background: string;
  cardBackground: string;
  buttonTextColor: string;
}

export const colors: ThemeColors = {
  primary: "#5856D6",
  text: "black",

  background: "#F3F2F7",
  cardBackground: "white",
  buttonTextColor: "white",
};

export const globalStyles = StyleSheet.create({
  title: {
    fontSize: 30,
    fontWeight: "bold",
    color: colors.text,
  },
  subTitle: {
    fontSize: 20,
    fontWeight: "bold",
    color: colors.text,
  },

  mainContainer: {
    flex: 1,
    backgroundColor: colors.background,
  },
  globalMargin: {
    paddingHorizontal: 20,
    flex: 1,
```

```
  },

  btnPrimary: {
    backgroundColor: colors.primary,
    borderRadius: 10,
    padding: 10,
    alignItems: "center",
  },
  btnPrimaryText: {
    color: colors.text,
    fontSize: 16,
  },
});
```

- En HomeScreen, aplico los estilos a los view
- Coloco un ScrollView. el ScrollView no es perezoso, si tienes muchos elementos es mejor usar un FlatList
- Creo en comoponents/ui/Title.tsx
- Pongo el white y el safe (que son opcionales) en false por defecto
- En los estilos esparzo los estilos globales y añado las propiedades que necesito
- Si viene el white, el texto será white. Si viene el safe, usaré el top que extraigo de useSafeAreaInsets para que no choque con la parte superior

```
import React from 'react'
import { Text, View } from 'react-native'
import { colors, globalStyles } from '../../../config/theme/theme'
import { useSafeAreaInsets } from 'react-native-safe-area-context'

interface Props{
    text: string
    safe?: boolean
    white?: boolean
}

export const Title = ({text, safe= false, white= false}: Props) => {

    const {top} = useSafeAreaInsets()
  return (
      <Text style={{
        ...globalStyles.title,
        marginTop: safe ? top : 0,
        marginBottom: 10,
        color: white ? 'white': colors.text

    }}>{text}</Text>
  )
}
```

- Lo coloco en el ScrollView con el safe en true (solo con colocar la prop es suficiente)

```
export const HomeScreen = () => {
  return (
    <View style={globalStyles.mainContainer}>
      <View style={globalStyles.globalMargin}>
        <ScrollView>
          <Title text="Opciones de Menú" safe />
        </ScrollView>
      </View>

    </View>
  )
}
```

Opciones del Menú

- Creo el components/ui/MenuItem.tsx
- Voy a usar el MenuItem para navegar, por lo que voy a usar un Pressable

```
import React from 'react'
import { Pressable, Text, View } from 'react-native'
import {StyleSheet} from 'react-native'
import { colors } from '../../../config/theme/theme'
import  Icon  from 'react-native-vector-icons/Ionicons'

interface Props{
    name: string
    icon: string
    component?: string
}

export const MenuItem = ({name, icon, component}: Props) => {
  return (
        <Pressable
        onPress={()=>console.log('click')}
        >
            <View style={{
                ...styles.container,
                backgroundColor: colors.cardBackground
            }}>
                <Icon name={icon} size={25} style={{marginRight:10, color:
colors.primary}} />
                <Text style={{color: colors.text}}>{name}</Text>
                <Icon name='chevron-forward-outline' size={25} style={{marginLeft:
'auto'}} color={colors.primary}/>

            </View>
        </Pressable>
  )
```

```
}


const styles = StyleSheet.create({
  container:{
    flexDirection: 'row',
    alignItems: 'center',
    paddingHorizontal: 10,
    paddingVertical: 5
  }
})
```

- Hago un .map en el ScrollView y renderizo el MenuItem
- Puedo hacerlo así

```
export const HomeScreen = () => {
  return (
    <View style={globalStyles.mainContainer}>
      <View style={globalStyles.globalMargin}>
        <ScrollView>
          <Title text="Opciones de Menú" safe />

          {
            menuItems.map(item=>(
              <MenuItem key={item.component} icon={item.icon} name={item.name} />
            ))
          }
        </ScrollView>
      </View>

    </View>
  )
```

- Pero también puedo hacerlo usando el spread

```
export const HomeScreen = () => {
  return (
    <View style={globalStyles.mainContainer}>
      <View style={globalStyles.globalMargin}>
        <ScrollView>
          <Title text="Opciones de Menú" safe />

          {
            menuItems.map(item=>(
              <MenuItem key={item.component} {...item} />
            ))
          }
        </ScrollView>
```

```
      </View>

    </View>
  )
```

- Añado nuevas props a MenuItem para estilizar y agrupar los elementos
- Si es el primero o es el último voy a poder aplicar estilos condicionales

```
import React from 'react'
import { Pressable, Text, View } from 'react-native'
import {StyleSheet} from 'react-native'
import { colors } from '../../../config/theme/theme'
import  Icon  from 'react-native-vector-icons/Ionicons'

interface Props{
    name: string
    icon: string
    component?: string
    isFirst?: boolean
    isLast?: boolean
}

export const MenuItem = ({name, icon, component, isFirst= false, isLast= false }:
Props) => {
  return (
        <Pressable
        onPress={()=>console.log('click')}
        >
            <View style={{
                ...styles.container,
                backgroundColor: colors.cardBackground,
                ...(isFirst && {borderTopLeftRadius: 10, borderTopRightRadius: 10,
paddingTop: 10}),
                ...(isLast && {borderBottomLeftRadius: 10,
borderBottomRightRadius: 10, paddingTop: 10})
            }}>
                <Icon name={icon} size={25} style={{marginRight:10, color:
colors.primary}} />
                <Text style={{color: colors.text}}>{name}</Text>
                <Icon name='chevron-forward-outline' size={25} style={{marginLeft:
'auto'}} color={colors.primary}/>

            </View>
        </Pressable>
  )
}

const styles = StyleSheet.create({
  container:{
    flexDirection: 'row',
    alignItems: 'center',
```

```
    paddingHorizontal: 10,
    paddingVertical: 5
  }
})
```

- Para que se apliquen los cambios tengo que indicar cual es el primero y cual es el último
- Para ello extraigo el index del map

```
export const HomeScreen = () => {
  return (
    <View style={globalStyles.mainContainer}>
      <View style={globalStyles.globalMargin}>
        <ScrollView>
          <Title text="Opciones de Menú" safe />

          {
            menuItems.map((item, index)=>(
              <MenuItem key={item.component} {...item}
              isFirst={index === 0 }
              isLast={index === menuItems.length -1}
              />
            ))
          }
        </ScrollView>
      </View>

    </View>
  )
}
```

- Vamos a separarlos por grupos

```
const animationMenuItems = [
  {
    name: 'Animation 101',
    icon: 'cube-outline',
    component: 'Animation101Screen',
  },
  {
    name: 'Animation 102',
    icon: 'albums-outline',
    component: 'Animation102Screen',
  }
]

const menuItems= [
  {
    name: 'Pull to refresh',
    icon: 'refresh-outline',
```

```
    component: 'PullToRefreshScreen',
  },
  {
    name: 'Section List',
    icon: 'list-outline',
    component: 'CustomSectionListScreen',
  },
  {
    name: 'Modal',
    icon: 'copy-outline',
    component: 'ModalScreen',
  },
  {
    name: 'InfiniteScroll',
    icon: 'download-outline',
    component: 'InfiniteScrollScreen',
  },
  {
    name: 'Slides',
    icon: 'flower-outline',
    component: 'SlidesScreen',
  },
  {
    name: 'Themes',
    icon: 'flask-outline',
    component: 'ChangeThemeScreen',
  }
]

const uiMenuItems =[
  {
    name: 'Switches',
    icon: 'toggle-outline',
    component: 'SwitchScreen',
  },
  {
    name: 'Alerts',
    icon: 'alert-circle-outline',
    component: 'AlertScreen',
  },
  {
    name: 'TextInputs',
    icon: 'document-text-outline',
    component: 'TextInputScreen',
  }
]
```

- Hago los .map de los tres grupos dentro del ScrollView
- Lo coloco dentro de diferentes Views con un marginTop

```
export const HomeScreen = () => {
  return (
    <View style={globalStyles.mainContainer}>
      <View style={globalStyles.globalMargin}>
        <ScrollView>
          <Title text="Opciones de Menú" safe />


          {
            menuItems.map((item, index)=>(
              <MenuItem key={item.component} {...item}
              isFirst={index === 0 }
              isLast={index === menuItems.length -1}
              />
              ))
            }
            <View style={{marginTop: 10}}>
            {
              animationMenuItems.map((item, index)=>(
                <MenuItem key={item.component} {...item}
                isFirst={index === 0 }
              isLast={index === animationMenuItems.length -1} />
              ))
            }

            </View>

            <View style={{marginTop: 10}} >
            {
              uiMenuItems.map((item, index)=>(
                <MenuItem key={item.component} {...item}
                isFirst={index === 0 }
                isLast={index === uiMenuItems.length -1} />
              ))
            }
            </View>
        </ScrollView>
      </View>

    </View>
  )
}
```

- En MenuItem uso el useNavigation. Hay que configurar el RootStackParams para tener el tipado
- Lo dejo temporalmente tipado como any (dará error porque no hay rutas definidas)
- Coloco el componente en navigation.navigate dentro de la prop onPress del Pressable

---

Animated API

- En las listas que he creado de menuItems en HomeScreen hay una de las propiedades que es component
- Uso el mismo nombre para nombrar a mis componentes en la carpeta components
- Vamos con las lista de animationMenuItems
- En animations/Animation101Screen.tsx creo un componente básico
- Lo añado en el StackNavigator

```
const Stack = createStackNavigator();

export const StackNavigator =()=> {
  return (
    <Stack.Navigator>
      <Stack.Screen name="HomeScreen" component={HomeScreen} />
      <Stack.Screen name="Animation101Screen" component={Animation101Screen} />

    </Stack.Navigator>
  );
}
```

- Cuando quiero aplicar un cambio visual lo debo colocar en un estado
- Si lo único que quiero es mantener el valor de una variable uso **useRef**
- Le paso una instancia de Animated donde el Value es 0.4 (algio translúcido, en este caso)
- Lo coloco en 0 porque quiero que la caja púrpura aparezca de la nada
- .current para tomar el valor actual
- este 0.4 no es de tipo number, es de tipo Animated.Value
- **No puedo** usar Animated properties **en un View** normal
- Trabajaré con Animated.View (hay otros, como Animated.Image, etc)
- Le paso en el arreglo de styles un objeto con la propiedad **opacity** y le paso la referencia animatedOpacity
- Para el fadeIn, primero lo haremos en crudo, luego haremos un custom hook
    - Uso el objeto **Animated**, en este caso con la propiedad **timing** y le paso la referencia y el objeto de configuración:
    - A que valor quiero que vaya
    - La duración en milésimas de segundo
    - El tipo de aceleración (driver)
- Para disparar la animación uso .start
- Puedo colocar un callback dentro de start para disparar otra animación cuando esta animación termine

```
import React, { useRef } from 'react'
import { Animated, Pressable, Text, View } from 'react-native'

export const Animation101Screen = () => {

  const animatedOpacity = useRef(new Animated.Value(0)).current

  const fadeIn = ()=>{
```

```
    Animated.timing(animatedOpacity,{
      toValue: 1,
      duration: 300, //ms
      useNativeDriver: true //acelerado por hardware
    }).start(()=>console.log("dispara esto cuando la animación termina"))
  }

  return (
    <View style={styles.container} >
        <Animated.View style={[
          styles.purpleBox, {
            opacity: animatedOpacity
          }]} />
        <Pressable onPress={fadeIn} style={{marginTop:10}} >
            <Text>FadeIn</Text>
        </Pressable>
        <Pressable onPress={()=> console.log('FadeOut')} style={{marginTop:10}} >
            <Text>FadeOut</Text>
        </Pressable>
    </View>
  )
}



import {StyleSheet} from 'react-native'
import { colors } from '../../../config/theme/theme'

const styles = StyleSheet.create({
  container:{
    flex:1,
    justifyContent: 'center',
    alignItems: 'center'
  },
  purpleBox:{
    backgroundColor: colors.primary,
    width: 150,
    height:150
  }
})
```

- El FadeOut es más de lo mismo!

```
const fadeOut =()=>{
  Animated.timing(animatedOpacity,{
    toValue: 0,
    duration: 500,
    useNativeDriver: true
  }).start()
}
```

- Le paso la función al onPress
- Ahora quiero que la caja caiga y rebote

Easing - Bounce

- Para animarlo que caiga, le paso una referencia con valor -100 y en la animación lo llevo a 0 usando la propiedad transform
- Uso un arreglo con un objeto dentro para indicarle la propiedad **translateY**
- Uso Easing para agregarle la propiedad elastic (hay varias). Con bounce rebota más
- En fadeOut uso .resetAnimation para devolverle el estado inicial
- La coloco en el callback para que lo haga cuando la animación terminó

```
import React, { useRef } from 'react'
import { Animated, Easing, Pressable, Text, View } from 'react-native'

export const Animation101Screen = () => {

  const animatedOpacity = useRef(new Animated.Value(0)).current

  const animatedTop = useRef(new Animated.Value(-100)).current

  const fadeIn = ()=>{

    Animated.timing(animatedTop, {
      toValue: 0,
      duration: 700,
      useNativeDriver: true,
      //easing: Easing.elastic(1)
      easing: Easing.bounce
    }).start()

    Animated.timing(animatedOpacity,{
      toValue: 1,
      duration: 300, //ms
      useNativeDriver: true //acelerado por hardware
    }).start()
  }

  const fadeOut =()=>{
    Animated.timing(animatedOpacity,{
      toValue: 0,
      duration: 500,
      useNativeDriver: true
    }).start(()=>animatedTop.resetAnimation())



  }

  return (
```

```
      <View style={styles.container} >
          <Animated.View style={[
            styles.purpleBox, {
              opacity: animatedOpacity,
              transform:[{
                translateY: animatedTop,
              }]
            }]} />
          <Pressable onPress={fadeIn} style={{marginTop:10}} >
              <Text>FadeIn</Text>
          </Pressable>
          <Pressable onPress={fadeOut} style={{marginTop:10}} >
              <Text>FadeOut</Text>
          </Pressable>
      </View>
    )
}



import {StyleSheet} from 'react-native'
import { colors } from '../../../config/theme/theme'

const styles = StyleSheet.create({
  container:{
    flex:1,
    justifyContent: 'center',
    alignItems: 'center'
  },
  purpleBox:{
    backgroundColor: colors.primary,
    width: 150,
    height:150
  }
})
```

- Hay mucha lógica aquí, hay sobrecarga. Vamos a crear un hook para customizar las animaciones y quede más limpio

---

CustomHook useAnimation

- En hooks/useAnimation.tsx
- Empiezo por copiar todo el código relacionado con las animaciones al hook
- Cambio el valor de animatedTop a 0 porque la cambiaré manualmente
- Quedaría algo así

```
import React, { useRef } from 'react'
import { Animated, Easing } from 'react-native'

export const useAnimation = () => {
```

```
    const animatedOpacity = useRef(new Animated.Value(0)).current

    const animatedTop = useRef(new Animated.Value(0)).current

    const fadeIn = ()=>{

      Animated.timing(animatedTop, {
        toValue: 0,
        duration: 700,
        useNativeDriver: true,
        //easing: Easing.elastic(1)
        easing: Easing.bounce
      }).start()

      Animated.timing(animatedOpacity,{
        toValue: 1,
        duration: 300, //ms
        useNativeDriver: true //acelerado por hardware
      }).start()
    }

    const fadeOut =()=>{
      Animated.timing(animatedOpacity,{
        toValue: 0,
        duration: 500,
        useNativeDriver: true
      }).start(()=>animatedTop.resetAnimation())
    }

    return {
        fadeIn,
        fadeOut
    }
}
```

- El fadeIn solo debería encargarse del fadeIn por lo que el animatedTop lo comento momentáneamente
- Para hacer los valores reutilizables le indico a fadeIn que recibirá un objeto con la duración.
    - Le coloco un valor por defecto de 400
    - Hago lo mismo con toValue

```
import React, { useRef } from 'react'
import { Animated, Easing } from 'react-native'

export const useAnimation = () => {

    const animatedOpacity = useRef(new Animated.Value(0)).current

    const animatedTop = useRef(new Animated.Value(0)).current

    const fadeIn = ({duration= 400, toValue=1, callback= ()=>{}})=>{
```

```
      /*Animated.timing(animatedTop, {
        toValue: 0,
        duration: 700,
        useNativeDriver: true,
        //easing: Easing.elastic(1)
        easing: Easing.bounce
      }).start()*/

      Animated.timing(animatedOpacity,{
        toValue,
        duration,
        useNativeDriver: true
      }).start(callback)
    }

    const fadeOut =({toValue=0, duration=400, callback=()=>{}})=>{
      Animated.timing(animatedOpacity,{
        toValue,
        duration,
        useNativeDriver: true
      }).start(callback)
    }

    return {
        animatedTop,
        animatedOpacity,
        fadeIn,
        fadeOut
    }
}
```

- Desestructuro del hook las funciones y valores
- Ahora en el onPress me pide que los llame con un callback y debo pasarle aunque sea un objeto vacío

```
import React, { useRef } from 'react'
import { Animated, Pressable, Text, View } from 'react-native'

export const Animation101Screen = () => {


  const {fadeIn, fadeOut, animatedTop, animatedOpacity}= useAnimation()




  return (
    <View style={styles.container} >
        <Animated.View style={[
          styles.purpleBox, {
            opacity: animatedOpacity,
```

```
            transform:[{
              translateY: animatedTop,
            }]
          }]} />
        <Pressable onPress={()=>fadeIn({})} style={{marginTop:10}} >
            <Text>FadeIn</Text>
        </Pressable>
        <Pressable onPress={()=>fadeOut({})} style={{marginTop:10}} >
            <Text>FadeOut</Text>
        </Pressable>
    </View>
  )
}
```

- Falta el movimiento

```
const startMovingTopPosition =({initialPosition= 0, toValue=0, duration=700,
easing= Easing.linear, callback=()=>{}})=>{
    animatedTop.setValue(initialPosition)
    Animated.timing(animatedTop, {
        toValue,
        duration,
        useNativeDriver: true,
        //easing: Easing.elastic(1)
        easing
    }).start(callback)
}
```

- Lo incluyo en el return del hook y lo desestructuro en Animation101Screen
- Ahora debo hacer una combinación
- En el cuerpo de la función del onPress (donde el fadeIn) pongo las dos animaciones (podría hacerlo en una función y pasarle la función)

```
import React, { useRef } from 'react'
import { Animated, Easing, Pressable, Text, View } from 'react-native'

export const Animation101Screen = () => {


  const {fadeIn, fadeOut, animatedTop, animatedOpacity, startMovingTopPosition}=
useAnimation()




  return (
    <View style={styles.container} >
        <Animated.View style={[
          styles.purpleBox, {
```

```
            opacity: animatedOpacity,
            transform:[{
              translateY: animatedTop,
            }]
          }]} />

        <Pressable onPress={()=>{
          fadeIn({})
          startMovingTopPosition({initialPosition:-100, easing: Easing.bounce,
duration:750})
        }} style={{marginTop:10}}>
            <Text>FadeIn</Text>
        </Pressable>

        <Pressable onPress={()=>fadeOut({})} style={{marginTop:10}} >
            <Text>FadeOut</Text>
        </Pressable>
    </View>
  )
}
```

Animated ValueXY

- Creo animations/Animation102Screen
- Lo coloco en el StackNavigator
- Uso la documentación oficial para recrear este botón arrastrable
- La propiedad useNativeDriver hay que ponerla en false
- Dice que necesita un segundo argumento, en Animated.event

```
import React, {useRef} from 'react';
import {Animated, PanResponder, StyleSheet, View} from 'react-native';

const DraggableView = () => {
  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}),  //aqui coloco el segundo argumento
    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: '#61dafb',
    width: 80,
    height: 80,
    borderRadius: 4,
  },
});

export default DraggableView;
```

- Lo coloco en Animation102Screen. Le añado unos estilos al View para que me quede en el centro

```
import React from 'react'
import { Text, View } from 'react-native'
import DraggableView from '../../components/Draggable'

export const Animation102Screen = () => {
  return (
    <View style={{flex: 1, justifyContent: 'center'}} >
      <DraggableView />
    </View>
  )
}
```

Componentes Personalizados

- Creo switches/SwitchesScreen.tsx, lo coloco en el StackNavigator
- Creo components/ui/CustomView.tsx
- En el styles del View coloco un arreglo dentro del objeto
- **NOTA**: para hacer la interfaz y saber el tipo de style, puedo escribirla (backgroundColor: 'black') y colocar el cursor encima

- Como le voy a colocar elementos dentro debo definir el children en las Props
- Uso el CustomView en la pantalla de switchesScreen

```
import React, { ReactNode } from 'react'
import { StyleProp, Text, View, ViewStyle } from 'react-native'
import { globalStyles } from '../../../config/theme/theme'

interface Props{
    style?: StyleProp<ViewStyle>
    children?: ReactNode
}


export const CustomView = ({style, children}: Props) => {
  return (
    <View style={[globalStyles.mainContainer, style]} >
      {children}
    </View>
  )
}
```

- Puedo heredar de la inetrfaz PropsWithChildren cuando voy a colocar children

```
import React, { PropsWithChildren, ReactNode } from 'react'
import { ImageBackgroundComponent, StyleProp, Text, View, ViewStyle } from 'react-
native'
import { colors } from '../../../config/theme/theme'

interface Props extends PropsWithChildren{
    style?: StyleProp<ViewStyle>
}

export const Card  = ({style, children}: Props) => {
  return (
    <View style={[
      {backgroundColor: colors.cardBackground,
      borderRadius: 10,
    padding: 10},
    style
    ]}>
      {children}
    </View>
  )
}
```

- La coloco en SwitchScreen

```
import React from 'react'
import { Text} from 'react-native'
import { CustomView } from '../../components/ui/CustomView'
import { Card } from '../../components/ui/Card'

export const SwitchScreen = () => {
  return (
    <CustomView style={{marginTop: 10, paddingHorizontal: 10}} >
        <Card>
        <Text>Component</Text>
        </Card>
    </CustomView>

  )
}
```

- Creo un Button customizable también
- En el style del Pressable utilizo una función para saber cuando está presionado.
    - Abro paréntesis para hacer implícito el return, y llaves cuadradas

```
import React from 'react'
import { Pressable, StyleProp, Text, View, ViewStyle } from 'react-native'
import { colors, globalStyles } from '../../../config/theme/theme'

interface Props{
    text: string
    styles?: StyleProp<ViewStyle>
    onPress: ()=>void
}

export const Button = ({text, styles, onPress}: Props) => {
  return (
    <Pressable
        onPress={onPress}
        style={(pressed)=>([
            globalStyles.btnPrimary,
            {
                opacity: pressed? 0.8 : 1,
                backgroundColor: colors.primary
            },
            styles
        ])}
    >
        <Text style={[
            globalStyles.btnPrimaryText,
            {
                color: colors.buttonTextColor
            }
        ]} >
          {text}
```

```
        </Text>
    </Pressable>
  )
}
```

- Lo coloco en SwitchScreen

```
import React from 'react'
import { Text} from 'react-native'
import { CustomView } from '../../components/ui/CustomView'
import { Card } from '../../components/ui/Card'
import { Button } from '../../components/ui/Button'

export const SwitchScreen = () => {
  return (
    <CustomView style={{marginTop: 10, paddingHorizontal: 10}} >
        <Card>
        <Text>Component</Text>
        </Card>
        <Button text="Click me!" onPress={()=>{}} />
    </CustomView>

  )
}
```

- Ahora que todo funciona, dejo el CustomView vacío en el SwitchScreen

---

Componente Switch

- Copio el código de la documentación, con los states incluidos

```
import React, { useState } from 'react'
import { Switch, Text} from 'react-native'
import { CustomView } from '../../components/ui/CustomView'
import { Card } from '../../components/ui/Card'


export const SwitchScreen = () => {
  const [isEnabled, setIsEnabled] = useState(false);
  const toggleSwitch = () => setIsEnabled(previousState => !previousState)
  return (
    <CustomView style={{marginTop: 10, paddingHorizontal: 10}} >
        <Card>
        <Switch
        trackColor={{false: '#767577', true: '#81b0ff'}}
        thumbColor={isEnabled ? '#f5dd4b' : '#f4f3f4'}
        ios_backgroundColor="#3e3e3e"
        onValueChange={toggleSwitch}
```

```
          value={isEnabled}
        />
          </Card>
      </CustomView>
    )
}
```

- En ios aparece a la izquierda, no como en Android
- Luego lo arreglaremos
- Este switch no está aplicando el tema de material 3. Después usaremos un componente especializado.
- Creo un custom switch

```
import React from 'react'
import { Switch } from 'react-native-gesture-handler'
import { Platform, Text, View } from 'react-native'

interface Props{
    isOn: boolean
    text?: string
    onChange: (value: boolean) => void
}


export const CustomSwitch = ({isOn, text, onChange}: Props) => {
  return (
    <View style={styles.switchRow}>
        {
            text && <Text style={{color: colors.text}}>{text}</Text>
        }

        <Switch
            thumbColor={Platform.OS === 'android'? colors.primary: ""}
            ios_backgroundColor="#3e3e3e"
            onValueChange={onChange}
            value={isOn}
            />
    </View>
  )
}


import {StyleSheet} from 'react-native'
import { colors } from '../../../config/theme/theme'

const styles = StyleSheet.create({
    switchRow: {
        flexDirection: 'row',
        justifyContent: 'space-between',
        alignItems: 'center',
        marginVertical: 5
```

```
    }
})
```

- Lo renderizo en SwitchScreen pasándole las props

```
import React, { useState } from 'react'
import { Switch, Text} from 'react-native'
import { CustomView } from '../../components/ui/CustomView'
import { Card } from '../../components/ui/Card'
import { CustomSwitch } from '../../components/ui/Switch'


export const SwitchScreen = () => {
  const [isEnabled, setIsEnabled] = useState(false);
  const toggleSwitch = () => setIsEnabled(previousState => !previousState)
  return (
    <CustomView style={{marginTop: 10, paddingHorizontal: 10}} >
        <Card>
          <CustomSwitch isOn={isEnabled} onChange={toggleSwitch} text="Encendido"
/>

        </Card>
    </CustomView>

  )
}
```

- Podemos implementar que en cualquier lado donde presione de la area de la card active el toggle
- O que si no viene el texto los elementos estén al final
- Hago una refactorización en SwitchScreen

```
import React, { useState } from 'react'
import { Switch, Text} from 'react-native'
import { CustomView } from '../../components/ui/CustomView'
import { Card } from '../../components/ui/Card'
import { CustomSwitch } from '../../components/ui/Switch'


export const SwitchScreen = () => {

   const [state, setState]= useState({
    isActive: true,
    isHungry: false,
    isHappy: true
   })


  return (
    <CustomView style={{marginTop: 10, paddingHorizontal: 10}} >
```

```
        <Card>
          <CustomSwitch isOn={state.isActive}
          onChange={(value)=> setState({...state, isActive: value})}
          text="Active" />
          <CustomSwitch isOn={state.isHungry}
          onChange={(value)=> setState({...state, isHungry: value})}
          text="Humgry" />
          <CustomSwitch isOn={state.isHappy}
          onChange={(value)=> setState({...state, isHappy: value})}
          text="Happy" />

        </Card>
    </CustomView>
  )
}
```

Componente Separador

- Creo components/ui/Separator.tsx

```
import React from 'react'
import { StyleProp, Text, View, ViewStyle } from 'react-native'
import { colors, globalStyles } from '../../../config/theme/theme'

interface Props{
    style?: StyleProp<ViewStyle>
}

export const Separator = ({style}: Props) => {

  return (
    <View style={[
        {
            alignSelf: 'center',
            width: '80%',
            height: 1,
            backgroundColor: colors.text,
            opacity: 0.1,
            marginVertical: 8
        },
        style
    ]}>
      <Text>Separator</Text>
    </View>
  )
}
```

- Coloco uno entre cada switch

```
return (
    <CustomView style={{marginTop: 10, paddingHorizontal: 10}} >
        <Card>
          <CustomSwitch isOn={state.isActive}
          onChange={(value)=> setState({...state, isActive: value})}
          text="Active" />
          <Separator />
          <CustomSwitch isOn={state.isHungry}
          onChange={(value)=> setState({...state, isHungry: value})}
          text="Humgry" />
          <Separator />
          <CustomSwitch isOn={state.isHappy}
          onChange={(value)=> setState({...state, isHappy: value})}
          text="Happy" />
        </Card>
    </CustomView>
  )
```

- Coloquémonos enmedio de los items en HomeScreen
- En MenuItem coloco el return en un Fragment para colocar el Separator después del Pressable
- Pregunto si es el último para no colocarlo después

```
    <>
        <Pressable
        onPress={()=>navigation.navigate(component)}
        >
            <View style={{
                ...styles.container,
                backgroundColor: colors.cardBackground,
                ...(isFirst && {borderTopLeftRadius: 10, borderTopRightRadius: 10,
paddingTop: 10}),
                ...(isLast && {borderBottomLeftRadius: 10,
borderBottomRightRadius: 10, paddingTop: 10})
            }}>
                <Icon name={icon} size={25} style={{marginRight:10, color:
colors.primary}} />
                <Text style={{color: colors.text}}>{name}</Text>
                <Icon name='chevron-forward-outline' size={25} style={{marginLeft:
'auto'}} color={colors.primary}/>

            </View>
        </Pressable>

            {!isLast && <Separator />}
    </>
```

- de esta manera no queda muy bien la presentación en el Home, refactorizo el separador, meto el view en otro view y le añado el backgroundColor de la card

```
import React from 'react'
import { StyleProp, Text, View, ViewStyle } from 'react-native'
import { colors, globalStyles } from '../../../config/theme/theme'

interface Props{
    style?: StyleProp<ViewStyle>
}

export const Separator = ({style}: Props) => {

  return (
    <View style={{backgroundColor: colors.cardBackground}} >
        <View style={[
            {
                alignSelf: 'center',
                width: '80%',
                height: 1,
                backgroundColor: colors.text,
                opacity: 0.1,
                marginVertical: 8
            },
            style
        ]}>

        </View>

    </View>
  )
}
```

Component Alert

- Creo AlertScreen y lo coloco en el StackNavigator con ese nombre, igual que en el arreglo de MenuItems
- Copio la documentación dentro de mi CustomView
- Creo el cascarón con los botones (mi CustomButton)
- Coloco un View con un high para poner distancia entre ellos

```
import React from 'react'
import { Text, View } from 'react-native'
import { CustomView } from '../../components/ui/CustomView'
import { Title } from '../../components/ui/Title'
import { globalStyles } from '../../../config/theme/theme'
import { Button } from '../../components/ui/Button'

export const AlertScreen = () => {
  return (
    <CustomView style={globalStyles.globalMargin}>
      <Title safe text="Alertas" />
```

```
      <Button text="Alerta - 2 botones" onPress={()=>{}} />

      <View style={{height: 10}} />

      <Button text="Alerta - 3 botones" onPress={()=>{}} />

      <View style={{height: 10}} />

      <Button text="Prompt - Input" onPress={()=>{}} />
    </CustomView>
  )
}
```

- Copio las funciones de la documentación y las coloco en el cuerpo del Functional Component
- Las uso en el onPress

```
import React from 'react'
import { Alert, Text, View } from 'react-native'
import { CustomView } from '../../components/ui/CustomView'
import { Title } from '../../components/ui/Title'
import { globalStyles } from '../../../config/theme/theme'
import { Button } from '../../components/ui/Button'

export const AlertScreen = () => {
    const createTwoButtonAlert = () =>
    Alert.alert('Alert Title', 'My Alert Msg', [
      {
        text: 'Cancel',
        onPress: () => console.log('Cancel Pressed'),
        style: 'cancel',
      },
      {text: 'OK', onPress: () => console.log('OK Pressed')},
    ]);

  const createThreeButtonAlert = () =>
    Alert.alert('Alert Title', 'My Alert Msg', [
      {
        text: 'Ask me later',
        onPress: () => console.log('Ask me later pressed'),
      },
      {
        text: 'Cancel',
        onPress: () => console.log('Cancel Pressed'),
        style: 'cancel',
      },
      {text: 'OK', onPress: () => console.log('OK Pressed')},
    ]);

  return (
    <CustomView style={globalStyles.globalMargin}>
```

```
      <Title safe text="Alertas" />

      <Button text="Alerta - 2 botones" onPress={createTwoButtonAlert} />

      <View style={{height: 10}} />

      <Button text="Alerta - 3 botones" onPress={createThreeButtonAlert} />

      <View style={{height: 10}} />

      <Button text="Prompt - Input" onPress={()=>{}} />
    </CustomView>
  )
}
```

- Las alertas se ven en plan Material 2
- No hay mucha personalización estética
- En el onPress de la Alert puedo disparar una acción
- Algunas de las opciones de configuración que colocaría en un segundo objeto funcionan en android y no en ios y al revés

```
const createTwoButtonAlert = () =>
Alert.alert('Alert Title', 'My Alert Msg', [
  {
    text: 'Cancel',
    onPress: () => console.log('Cancel Pressed'),
    style: 'cancel',
  },
  {text: 'OK', onPress: () => console.log('OK Pressed')},

],
{cancelable: true,
onDismiss(){
    console.log('onDismiss')
}}
);
```

Component Alert Prompt

```
const showPrompt = () =>{
    Alert.prompt('¿Cual es tu correo electrónico?','Texto blablablabla',
    (value: string)=> console.log({value})
    )
}
```

- Uso el showPrompt en el onPress

- En Android no funciona (en ios si)
- En ios puedo hacer que el input aparezca encriptado, añadirle un valor por defecto, y que saque el teclado numérico de esta manera

```
const showPrompt = () =>{
    Alert.prompt(
        '¿Cual es tu correo electrónico?',
        'Texto blablablabla',
        (value: string)=> console.log({value}),
        'secure-text',
        'Soy el valor por defecto',
        'number-pad'
    )
}
```

Prompt ios y Android

- Instalo el paquete (no es el paquete más actualizado del mundo)

> react-native-prompt-android

- Ya no hace falta hacer el link manual
- Copio el ejemplo de la documentación

```
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'
        }
    );
   }
```

- Con paquetes de terceros debo usar el patrón adaptador
- En config/adapters/prompt.adapter.ts

```
import prompt from "react-native-prompt-android";


interface Options{
    title: string
    subTitle?: string
    buttons: PromptButton[]
    promptType?: 'default' | 'plain-text' | 'secure-text'
    placeholder?: string
    defaultValue?: string
}

interface PromptButton{
    text: string
    onPress: ()=> void
    style?: "cancel" | "default" | "destructive"
}

export  const showPrompt = ({title, subTitle, buttons, promptType, placeholder,
defaultValue}:Options) =>{
    prompt(
      title,
      subTitle,
      buttons,

      {
          type: promptType= 'plain-text', //plain-text por defecto para que se
muestre también en ios
          cancelable: false,
          defaultValue,
          placeholder
      }
  );
 }
```

- Cambio el nombre del método que tenía por onShowPrompt
- En AlertScreen

```
const onShowPrompt = () =>{
  showPrompt({title:"Titulo",
  subTitle:"Subtitulo",
  buttons: [
    {text: 'OK', onPress:()=>console.log('button OK')},
    {text: 'Cancel', onPress:()=>console.log('button Cancel')}
  ],
  placeholder: 'PlaceHolder'
  })
}
```

- De esta manera solo hay que cambiar la implementación y no todas las pantallas donde se consuma el prompt
-

Componente TextInput

- Creo screens/inputs/textInputScreen. Lo coloco en el StackNavigator
- Como todo el rato estoy colocando el globalStyles.marginCntainer en el CustomView lo añado como una prop al componente
- Si el margin está en true añada el estilo

```
import React, { ReactNode } from 'react'
import { StyleProp, Text, View, ViewStyle } from 'react-native'
import { globalStyles } from '../../../config/theme/theme'

interface Props{
    style?: StyleProp<ViewStyle>
    children?: ReactNode
    margin?: boolean
}


export const CustomView = ({margin = false, style, children}: Props) => {
  return (
    <View style={[
      margin ? globalStyles.mainContainer: null,
      style]} >
      {children}
    </View>
  )
}
```

- Al poner el TextInput dentro de mi custom Card hace que se vea bonito

```
import React from 'react'
import { Text, TextInput, View } from 'react-native'
import { CustomView } from '../../components/ui/CustomView'
import { Title } from '../../components/ui/Title'
import { Card } from '../../components/ui/Card'

export const TextInputScreen = () => {
  return (
    <CustomView margin>
      <Title text="text Inputs" safe />
      <Card>
        <TextInput/>
      </Card>
    </CustomView>
```

```
  )
}
```

- Defino un estilo de CSS desde el theme global
- El color del texto al cambiarlo de manera dinámica me dará un inconveniente más adelante. Lo solucionaremos

```
  input:{
    height: 40,
    margin: 12,
    borderWidth: 1,
    padding: 10,
    borderColor: 'rgba(0,0,0,0.3)',
    borderRadius: 10,
    color: colors.text
  }
```

- Hay varias properties en TextInput
  - autoCapitalize no es un valor booleano, indico si quiero capitalizar por palabras, sentencias...

```
import React, { useState } from 'react'
import { Text, TextInput, View } from 'react-native'
import { CustomView } from '../../components/ui/CustomView'
import { Title } from '../../components/ui/Title'
import { Card } from '../../components/ui/Card'
import { globalStyles } from '../../../config/theme/theme'

export const TextInputScreen = () => {

    const [form, setForm] = useState({
        name: '',
        email:'',
        phone: ''
    })

  return (
    <CustomView margin>
      <Title text="Text Input" safe />
      <Card>
        <TextInput style={globalStyles.input}
            placeholder="Nombre completo"
            autoCapitalize='words'
            autoCorrect={false}
            onChangeText={value=>setForm({...form, name: value})}
        />
      </Card>

      <View style={{height: 10}} />
      <Card>
```

```
          <TextInput style={globalStyles.input}
              placeholder="Email"
              autoCapitalize='words'
              autoCorrect={false}
              onChangeText={value=>setForm({...form, email: value})}
          />
        </Card>
        <View style={{height: 10}} />
        <Card>
          <TextInput style={globalStyles.input}
              placeholder="Teléfono"
              autoCapitalize='words'
              keyboardType='phone-pad'
              autoCorrect={false}
              onChangeText={value=>setForm({...form, phone: value})}
          />
        </Card>
        <View style={{height: 10}} />

          <Card>
              <Text>{JSON.stringify(form, null, 2)}</Text>
          </Card>
      </CustomView>
    )
}
```

- Ahora hay un problema y es que si tuviera varios objetos que leer en pantalla, no podría hacer scroll
- Meto todo el contenido dentro de un ScrollView

Scroll y teclado

- Si coloco un TextInput al final, al aparecer el teclado en Android si lo desplaza hacia arriba pero en ios no
- En ios es necesario envolver el componente dentro de un KeyboradAvoidingView~para que el teclado en pantalla no de problemas
- También es recomendable poner un View con un margin de gracia al final

```
import React, { useState } from 'react'
import { KeyboardAvoidingView, Platform, ScrollView, Text, TextInput, View } from
'react-native'
import { CustomView } from '../../components/ui/CustomView'
import { Title } from '../../components/ui/Title'
import { Card } from '../../components/ui/Card'
import { globalStyles } from '../../../config/theme/theme'

export const TextInputScreen = () => {

    const [form, setForm] = useState({
        name: '',
        email:'',
```

```
        phone: ''
    })

  return (

    <KeyboardAvoidingView behavior={Platform.OS === 'ios'? 'padding': undefined}>
    <ScrollView>

    <CustomView margin>
      <Title text="Text Input" safe />
      <Card>
        <TextInput style={globalStyles.input}
            placeholder="Nombre completo"
            autoCapitalize='words'
            autoCorrect={false}
            onChangeText={value=>setForm({...form, name: value})}
        />
      </Card>

      <View style={{height: 10}} />
      <Card>
        <TextInput style={globalStyles.input}
            placeholder="Email"
            autoCapitalize='words'
            autoCorrect={false}
            onChangeText={value=>setForm({...form, email: value})}
        />
      </Card>
      <View style={{height: 10}} />
      <Card>
        <TextInput style={globalStyles.input}
            placeholder="Teléfono"
            autoCapitalize='words'
            keyboardType='phone-pad'
            autoCorrect={false}
            onChangeText={value=>setForm({...form, phone: value})}
        />
      </Card>
      <View style={{height: 10}} />

        <Card>
            <Text>{JSON.stringify(form, null, 2)}</Text>
        </Card>
        <View style={{height: 20}} />
    </CustomView>
    </ScrollView>
    </KeyboardAvoidingView>
  )
}
```

Pull To Refresh

- Creo screens/ui/PullToRefreshScreen. Lo coloco en el StackNavigator
- RefreshControl pide ciertas properties obligatorias
- Con el refreshing en true, en ios el notch cubre el spinning
- Para arreglarlo uso el safeAreaInsets y extraigo el top, lo coloco en progressViewOffset
- Cambio el true en duro por un state
- En onRefresh llamo a la función que he creado
- Le puedo pasar un arreglo de colores en hexadecimal
- Para que se vea bien en ios le paso el globalStyles al ScrollView

```
import React, { useState } from 'react'
import { RefreshControl, ScrollView, Text, View } from 'react-native'
import { Title } from '../../components/ui/Title'
import { CustomView } from '../../components/ui/CustomView'
import { useSafeAreaInsets } from 'react-native-safe-area-context'
import { colors, globalStyles } from '../../../config/theme/theme'

export const PullToRefreshScreen = () => {

    const [isRefreshing, setIsRefreshing] = useState(false)

    const {top} = useSafeAreaInsets()

    const onRefresh =()=>{
        setIsRefreshing(true)

        setTimeout(()=>{
            setIsRefreshing(false)
        }, 2000)
    }

  return (
    <ScrollView refreshControl={
    <RefreshControl
        refreshing={isRefreshing}
        progressViewOffset={top}
        onRefresh={onRefresh}
        colors={[colors.primary, 'red', 'orange', 'green']} />}
        style={[globalStyles.mainContainer, globalStyles.globalMargin]}
        >
        <CustomView margin>
            <Title text="Pull To Refresh" safe  />
        </CustomView>
    </ScrollView>
  )
}
```

- Ahora si tiro de la pantalla aparece un spinning
- Podríamos crear un CustomScrollView para que quedara esta configuración siempre y no estar colocando esto una y otra vez

Componente - SectionList

- Copio la data del Gist para mostrar en el SectionList

> https://gist.githubusercontent.com/Klerith/62fc1759bbb686446caf22a04956d47e/raw/fdb7dc53c1385a5eeaf92ce2a7c53ddd1ba3c27a/characters.ts

```
interface Houses {
  title: string;
  data: string[];
}

const houses: Houses[] = [
  {
    title: 'DC Comics',
    data: [
      'Superman',
      'Batman',
      'Wonder Woman (Mujer Maravilla)',
      'The Flash (Flash)',
      'Aquaman',
      'Green Lantern (Linterna Verde)',
      'Cyborg',
      'Shazam',
      'Green Arrow (Flecha Verde)',
      'Batgirl (Batichica)',
      'Nightwing (Ala Nocturna)',
      'Supergirl',
      'Martian Manhunter (Detective Marciano)',
      'Harley Quinn',
      'Joker',
      'Catwoman (Gata Salvaje)',
      'Lex Luthor',
      'Poison Ivy (Hiedra Venenosa)',
      'Robin',
      'Deathstroke (Deathstroke el Terminator)',
    ],
  },
  {
    title: 'Marvel Comics',
    data: [
      'Spider-Man (Hombre Araña)',
      'Iron Man (Hombre de Hierro)',
      'Captain America (Capitán América)',
      'Thor',
      'Black Widow (Viuda Negra)',
      'Hulk',
      'Doctor Strange (Doctor Extraño)',
      'Black Panther (Pantera Negra)',
      'Captain Marvel (Capitana Marvel)',
      'Wolverine',
      'Deadpool',
```

```
      'Scarlet Witch (Bruja Escarlata)',
      'Ant-Man (Hombre Hormiga)',
      'Wasp (Avispa)',
      'Groot',
      'Rocket Raccoon (Mapache Cohete)',
      'Gamora',
      'Drax the Destroyer (Drax el Destructor)',
    ],
  },
  {
    title: 'Anime',
    data: [
      'Son Goku (Dragon Ball)',
      'Naruto Uzumaki (Naruto)',
      'Monkey D. Luffy (One Piece)',
      'Sailor Moon (Sailor Moon)',
      'Kenshin Himura (Rurouni Kenshin)',
      'Edward Elric (Fullmetal Alchemist)',
      'Inuyasha (Inuyasha)',
      'Sakura Kinomoto (Cardcaptor Sakura)',
      'Light Yagami (Death Note)',
      'Eren Yeager (Attack on Titan)',
      'Lelouch Lamperouge (Code Geass)',
      'Vegeta (Dragon Ball)',
      'Ichigo Kurosaki (Bleach)',
      'Kaneki Ken (Tokyo Ghoul)',
      'Gon Freecss (Hunter x Hunter)',
      'Asuka Langley Soryu (Neon Genesis Evangelion)',
      'Saitama (One Punch Man)',
      'Mikasa Ackerman (Attack on Titan)',
      'Natsu Dragneel (Fairy Tail)',
      'Usagi Tsukino (Sailor Moon)',
      'Sasuke Uchiha (Naruto)',
    ],
  },
];
```

- Creo en screens/ui/CustomSectionListScreen, lo coloco en el StackNavigator
- Coloco mi CustomVIew, coloco un Title y el SectionList que tiene autocierre
    - Me pidelas props sections, keyExtractor y uso renderItem para renderizar cada elemento (puedo extraer tambien el index si lo necesito)

```
import React from 'react'
import { SectionList, Text, View } from 'react-native'
import { CustomView } from '../../components/ui/CustomView'
import { Title } from '../../components/ui/Title'
import { Card } from '../../components/ui/Card'

interface Houses {
    title: string;
    data: string[];
```

```
}

const houses: Houses[] = [
  {
    title: 'DC Comics',
    data: [
      'Superman',
      'Batman',
      'Wonder Woman (Mujer Maravilla)',
      'The Flash (Flash)',
      'Aquaman',
      'Green Lantern (Linterna Verde)',
      'Cyborg',
      'Shazam',
      'Green Arrow (Flecha Verde)',
      'Batgirl (Batichica)',
      'Nightwing (Ala Nocturna)',
      'Supergirl',
      'Martian Manhunter (Detective Marciano)',
      'Harley Quinn',
      'Joker',
      'Catwoman (Gata Salvaje)',
      'Lex Luthor',
      'Poison Ivy (Hiedra Venenosa)',
      'Robin',
      'Deathstroke (Deathstroke el Terminator)',
    ],
  },
  {
    title: 'Marvel Comics',
    data: [
      'Spider-Man (Hombre Araña)',
      'Iron Man (Hombre de Hierro)',
      'Captain America (Capitán América)',
      'Thor',
      'Black Widow (Viuda Negra)',
      'Hulk',
      'Doctor Strange (Doctor Extraño)',
      'Black Panther (Pantera Negra)',
      'Captain Marvel (Capitana Marvel)',
      'Wolverine',
      'Deadpool',
      'Scarlet Witch (Bruja Escarlata)',
      'Ant-Man (Hombre Hormiga)',
      'Wasp (Avispa)',
      'Groot',
      'Rocket Raccoon (Mapache Cohete)',
      'Gamora',
      'Drax the Destroyer (Drax el Destructor)',
    ],
  },
  {
    title: 'Anime',
    data: [
```

```
        'Son Goku (Dragon Ball)',
        'Naruto Uzumaki (Naruto)',
        'Monkey D. Luffy (One Piece)',
        'Sailor Moon (Sailor Moon)',
        'Kenshin Himura (Rurouni Kenshin)',
        'Edward Elric (Fullmetal Alchemist)',
        'Inuyasha (Inuyasha)',
        'Sakura Kinomoto (Cardcaptor Sakura)',
        'Light Yagami (Death Note)',
        'Eren Yeager (Attack on Titan)',
        'Lelouch Lamperouge (Code Geass)',
        'Vegeta (Dragon Ball)',
        'Ichigo Kurosaki (Bleach)',
        'Kaneki Ken (Tokyo Ghoul)',
        'Gon Freecss (Hunter x Hunter)',
        'Asuka Langley Soryu (Neon Genesis Evangelion)',
        'Saitama (One Punch Man)',
        'Mikasa Ackerman (Attack on Titan)',
        'Natsu Dragneel (Fairy Tail)',
        'Usagi Tsukino (Sailor Moon)',
        'Sasuke Uchiha (Naruto)',
      ],
    },
  ];

export const CustomSectionListScreen = () => {
  return (
    <CustomView>
        <Title text="Lista de personajes" />

        <Card>
            <SectionList
                sections={houses}
                keyExtractor={item=> item}
                renderItem={({item})=> <Text style={{marginVertical: 2}} >{item}
</Text>}
                showsVerticalScrollIndicator={false}
                renderSectionHeader={({section})=><Text style={{fontWeight: '500',
fontSize: 20, marginVertical: 10}} >{section.title} </Text>}
            />
        </Card>
    </CustomView>
  )
}
```

- Me creo un componente para renderizar en el renderSectionHeader

```
import React from 'react'
import { Text, View } from 'react-native'
import { colors } from '../../../config/theme/theme'
import { useSafeAreaInsets } from 'react-native-safe-area-context'
import { globalStyles } from '../../../config/theme/theme'
```

```
interface Props{
    text: string
    safe?: boolean
    backgroundColor?: string
}

export const SubTitle = ({text, safe= false, backgroundColor= colors.background}:
Props) => {

    const {top} = useSafeAreaInsets()

    return (

      <Text style={{
            ...globalStyles.subTitle,
            marginTop: safe? top: 0,
            marginBottom: 10,
            backgroundColor
      }}>{text}</Text>

  )
}
```

- Renderizo Subtitle en el renderSectionHeader del SectionList

```
<Card>
    <SectionList
        sections={houses}
        keyExtractor={item=> item}
        renderItem={({item})=> <Text style={{marginVertical: 2}} >{item}</Text>}
        showsVerticalScrollIndicator={false}
        renderSectionHeader={({section})=> <SubTitle  safe text={section.title}
/>}
        stickySectionHeadersEnabled
        SectionSeparatorComponent={ Separator}
        ListHeaderComponent={()=> <Title text="Personajes" />}
        ListFooterComponent={()=> <Title text={`Secciones:  ${houses.length}`} />}
    />
</Card>
```

- Si no especifico un tamaño del dispositivo voy a disponer de todo el dispositivo
- En el style puedo especificar un height para que ocupe la mitad de la pantalla con un 500
- Ahora solo debería sacar las dimensiones del dispositivo con useDimensions, restar el top

```
import React from 'react'
import { SectionList, Text, View, useWindowDimensions } from 'react-native'
import { CustomView } from '../../components/ui/CustomView'
```

```
import { Title } from '../../components/ui/Title'
import { Card } from '../../components/ui/Card'
import { SubTitle } from '../../components/ui/Subtitle'
import { Separator } from '../../components/ui/Separator'
import { useSafeAreaInsets } from 'react-native-safe-area-context'

interface Houses {
    title: string;
    data: string[];
  }

  const houses: Houses[] = [
    {
      title: 'DC Comics',
      data: [
        'Superman',
        'Batman',
        'Wonder Woman (Mujer Maravilla)',
        'The Flash (Flash)',
        'Aquaman',
        'Green Lantern (Linterna Verde)',
        'Cyborg',
        'Shazam',
        'Green Arrow (Flecha Verde)',
        'Batgirl (Batichica)',
        'Nightwing (Ala Nocturna)',
        'Supergirl',
        'Martian Manhunter (Detective Marciano)',
        'Harley Quinn',
        'Joker',
        'Catwoman (Gata Salvaje)',
        'Lex Luthor',
        'Poison Ivy (Hiedra Venenosa)',
        'Robin',
        'Deathstroke (Deathstroke el Terminator)',
      ],
    },
    {
      title: 'Marvel Comics',
      data: [
        'Spider-Man (Hombre Araña)',
        'Iron Man (Hombre de Hierro)',
        'Captain America (Capitán América)',
        'Thor',
        'Black Widow (Viuda Negra)',
        'Hulk',
        'Doctor Strange (Doctor Extraño)',
        'Black Panther (Pantera Negra)',
        'Captain Marvel (Capitana Marvel)',
        'Wolverine',
        'Deadpool',
        'Scarlet Witch (Bruja Escarlata)',
        'Ant-Man (Hombre Hormiga)',
        'Wasp (Avispa)',
```

```
          'Groot',
          'Rocket Raccoon (Mapache Cohete)',
          'Gamora',
          'Drax the Destroyer (Drax el Destructor)',
        ],
      },
      {
        title: 'Anime',
        data: [
          'Son Goku (Dragon Ball)',
          'Naruto Uzumaki (Naruto)',
          'Monkey D. Luffy (One Piece)',
          'Sailor Moon (Sailor Moon)',
          'Kenshin Himura (Rurouni Kenshin)',
          'Edward Elric (Fullmetal Alchemist)',
          'Inuyasha (Inuyasha)',
          'Sakura Kinomoto (Cardcaptor Sakura)',
          'Light Yagami (Death Note)',
          'Eren Yeager (Attack on Titan)',
          'Lelouch Lamperouge (Code Geass)',
          'Vegeta (Dragon Ball)',
          'Ichigo Kurosaki (Bleach)',
          'Kaneki Ken (Tokyo Ghoul)',
          'Gon Freecss (Hunter x Hunter)',
          'Asuka Langley Soryu (Neon Genesis Evangelion)',
          'Saitama (One Punch Man)',
          'Mikasa Ackerman (Attack on Titan)',
          'Natsu Dragneel (Fairy Tail)',
          'Usagi Tsukino (Sailor Moon)',
          'Sasuke Uchiha (Naruto)',
        ],
      },
    ];

  export const CustomSectionListScreen = () => {

    const {height} = useWindowDimensions()

    const {top} = useSafeAreaInsets()

    return (
      <CustomView>
          <Title text="Lista de personajes" />

          <Card>
              <SectionList
                  sections={houses}
                  keyExtractor={item=> item}
                  renderItem={({item})=> <Text style={{marginVertical: 2}} >{item}
  </Text>}
                  showsVerticalScrollIndicator={false}
                  renderSectionHeader={({section})=> <SubTitle  safe text=
  {section.title} />}
                  stickySectionHeadersEnabled
```

```
                SectionSeparatorComponent={ Separator}
                ListHeaderComponent={()=> <Title text="Personajes" />}
                ListFooterComponent={()=> <Title text={`Secciones:
${houses.length}`} /> }
                style={{
                  height: height - top - 120
                }}
            />
        </Card>
    </CustomView>
  )
}
```

Modal

- Creo screens/ui/ModalScreen. Lo coloco en el StackNavigator
- Tenemos un Modal en componentes de React Native pero vamos a hacer algo distinto
- Si le coloco la propiedad visible en false se deja de ver el modal
- Manejarlo con un state es básicamente todo

```
import React from 'react'
import { Modal, Text, View } from 'react-native'
import { CustomView } from '../../components/ui/CustomView'
import { Title } from '../../components/ui/Title'

export const ModalScreen = () => {
  return (
    <CustomView>
      <Title text="Modal" />

      <Modal visible={false}>
        <View>
          <Title text="Modal Content" />
        </View>
      </Modal>
    </CustomView>
  )
}
```

InfiniteScroll

- Creo el componente, lo coloco en el StackNavigator
- Hagamos un ejercicio sencillo
- Creo un estado para guardar los números
- Usualmente vamos a querer usar un FlatList para hacer un InfiniteScroll, ya que carga de manera perezosa
- Con un ScrollView todos los elementos van a ser cargados desde el inicio

- EL FlatList tiene la propiedad onEndReached para disparar una función cuando llega al final
- Uso el setstate, esparzo el state anterior y el nuevo Array

```
import React, { useState } from 'react'
import { Text, View } from 'react-native'
import { CustomView } from '../../components/ui/CustomView'
import { Title } from '../../components/ui/Title'
import { colors } from '../../../config/theme/theme'
import { FlatList } from 'react-native-gesture-handler'

export const InfiniteScroll = () => {

    const [numbers, setNumbers] = useState([0,1,2,3,4,5])

    const loadMore =()=>{
        const newArray = Array.from({length: 5}, (_,i)=>numbers.length +i)
        setNumbers([...numbers, ...newArray])
    }


  return (
    <CustomView margin>
      <Title text="InfiniteScroll" />
      <FlatList
        onEndReached={loadMore}
        data={numbers}
        renderItem={({item}: any)=>(
            <Text style={{height: 300, backgroundColor: colors.primary, color:
'white', fontSize: 50}}>{item}</Text>
        )}

        />
    </CustomView>
  )
}
```

- Para evitar el trompicón y cargar los elementos antes de que llegue al final tengo la proponEndReachedTreshold, cuyo valor por defecto es 0.5
- Puedo ponerlo en 0.6 para que empiece a cargar antes de que llegue al final
- Debo colocar el keyExtractor, la key debe de ser un string

```
<CustomView margin>
  <Title text="InfiniteScroll" />
  <FlatList
    onEndReached={loadMore}
    data={numbers}
    renderItem={({item}: any)=>(
        <Text style={{height: 300, backgroundColor: colors.primary, color:
'white', fontSize: 50}}>{item}</Text>
    )}
```

```
    onEndReachedThreshold={0.6}
    keyExtractor={(item)=> item.toString()}
    />
</CustomView>
```

InfiniteScroll con imágenes

- Usaremos Lorem Picsum para las imágenes
- Creo un componente para renderizar los números

```
import React, { useState } from 'react'
import { Text, View, VirtualizedListWithoutRenderItemProps } from 'react-native'
import { CustomView } from '../../components/ui/CustomView'
import { Title } from '../../components/ui/Title'
import { colors } from '../../../config/theme/theme'
import { FlatList } from 'react-native-gesture-handler'

export const InfiniteScroll = () => {

    const [numbers, setNumbers] = useState([0,1,2,3,4,5])

    const loadMore =()=>{
        const newArray = Array.from({length: 5}, (_,i)=>numbers.length +i)
        setNumbers([...numbers, ...newArray])
    }


  return (
    <CustomView margin>
      <Title text="InfiniteScroll" />
      <FlatList
        onEndReached={loadMore}
        data={numbers}
        renderItem={({item}: any)=>(
            <ListItem number={item} />
        )}
        onEndReachedThreshold={0.6}
        keyExtractor={(item)=> item.toString()}
        />
    </CustomView>
  )
}

interface ListItemProps{
    number: number
}

const ListItem = ({number}: ListItemProps)=>{
    return(
```

```
        <Text style={{height: 300, backgroundColor: colors.primary, color:
'white', fontSize: 50}}>{number}</Text>
    )
}
```

- Ahora en lugar de renderizar el texto renderizamos un Image

```
const ListItem = ({number}: ListItemProps)=>{
    return(
        <Image  source={{uri:`https://picsum.photos/id/${number}/500/400`}}
        style={{
            height: 400,
            width: '100%'
        }}
        />
    )
}
```

- Cuando la imagen se carga en Android tiene un fadeIn pero en ios no
- Quiero mostrar algún tipo de indicador cuando estoy mostrando las imágenes
- Uso la prop ListFooterComponent de FlatList

```
<CustomView margin>
      <Title text="InfiniteScroll" />
      <FlatList
        onEndReached={loadMore}
        data={numbers}
        renderItem={({item}: any)=>(
            <ListItem number={item} />
        )}
        onEndReachedThreshold={0.6}
        keyExtractor={(item)=> item.toString()}
        ListFooterComponent={()=>(
            <View style={{height: 150, justifyContent: 'center'}} >
                <ActivityIndicator size={50} color={colors.primary} />
            </View>
        )}
        />
    </CustomView>
```

Animated Image

- Uso Animated.Image como componente
- Tengo una propiedad que es onLoadEnd que puedo usar para disparar la animación que extraigo del custom hook useAnimation
- El ActivityIndicator solo debe mostrarse si el isLoading está en true

```
import React, { useState } from 'react'
import { Animated, ImageStyle, StyleProp, View } from 'react-native'
import { useAnimation } from '../../hooks/useAnimation'
import { ActivityIndicator } from 'react-native-paper'

interface Props{
    uri: string,
    style?: StyleProp<ImageStyle>
}

export const FadeInImage = ({uri, style}: Props) => {
    const {fadeIn, animatedOpacity}= useAnimation()
    const [isLoading, setIsLoading] = useState(true)
  return (
    <View style={{justifyContent: 'center', alignItems: 'center'}}>

    {isLoading &&
    <ActivityIndicator
        style={{position: 'absolute'}}
        color= 'blue'
        size={30}
    />}

      <Animated.Image
        source={{uri}}
        onLoadEnd={()=>(
            fadeIn({duration:1000}),
            setIsLoading(false)
        )}
        style={[style, {opacity: animatedOpacity}]}

      />
    </View>
  )
}
```

- Se lo paso a ListItem para que lo renderice en el renderItem del FlatList
- Debo pasarle un height y un width para que muestre las imágenes

```
const ListItem = ({number}: ListItemProps)=>{
    return(
        <FadeInImage uri={`https://picsum.photos/id/${number}/500/400`}
            style={{
                height:400,
                width: '100%'
            }}
        />
```

```
        )
    }
```