




DESARROLLO DE APPS CON REACT NATIVE



Ing. Marco León Mora Méndez
SENA - CEET

Contenido

DESARROLLO DE APPS	4
REACT NATIVE	5
ENTORNO DE DESARROLLO	6
Configurar el entorno de desarrollo	6
CREAR UNA APP	11
Ejecutar la App en un emulador.....	11
Ejecutar la App en un dispositivo físico	12
COMPONENTES CENTRALES	13
Crear una ventana de inicio	13
Imagen de Fondo	13
Formulario de ingreso	14
Elementos del formulario	17
Crear elementos personalizados	20
Pasando parámetros al componente	22
Crear estilos globales	23
NAVEGACION ENTRE PANTALLAS	25
Separando las pantallas	25
Crear pantalla de registro	30
Agregar componentes a la pantalla de registro	32
Capturar valores de los inputs	36
ORGANIZANDO EL CÓDIGO CON EL PATRÓN DE DISEÑO MVVM	38
Reorganizar las carpetas según las convenciones de Google	38
Capa de Presentación	38
Capa de Dominio.....	38
Capa de Datos.....	38
Crear un componente personalizado para los inputs	40
Separar las hojas de estilos	47
MONTANDO UN SERVIDOR NODE JS	52
Prueba de Rutas del servidor.....	53
CONECTANDO A LA BASE DE DATOS (MySQL).....	56
Realizando la conexión desde Node.js	56
Creando un usuario	60
Creando usuarios desde la App.....	60

AJUSTANDO EL PROYECTO A LA ARQUITECTURA MVVM	62
Ajustar el dominio	62
Ajustes a las pantallas para mejorar la presentación.....	65
Ajustar la interfase	66
Estructura de carpetas del proyecto	68
Organización de los módulos de código	70
Validaciones en los formularios	71
Cambios al tipo de respuesta y validaciones	72
Encriptar el password	77
JSON WEB TOKEN	80
Configuración del back-end para generar el token	80
Enviando el login desde la App	87
Almacenamiento interno la de sesión del usuario.....	93
CUSTOM HOOKS	98
Implementación del Custom Hook	98
Navegar usando el Custom hook	100
Cerrar la sesión	105

DESARROLLO DE APPS

La creación de una aplicación (APP) se refiere al proceso de desarrollar software para dispositivos móviles o de escritorio que brinda funcionalidades específicas a los usuarios. Existen dos enfoques principales para desarrollar aplicaciones móviles: plataformas nativas y plataformas multiplataforma.

Las plataformas nativas, como Android o *iOS*, permiten desarrollar aplicaciones utilizando los lenguajes de programación y las herramientas específicas de cada plataforma. Esto significa que se debe escribir código separado para cada plataforma, lo que puede requerir más tiempo y recursos.

Por otro lado, los sistemas multiplataforma, como *React Native* y *Flutter*, permiten desarrollar aplicaciones utilizando un único código base que se puede ejecutar en múltiples plataformas. Estas plataformas utilizan *frameworks* y bibliotecas que traducen el código a un lenguaje nativo para cada plataforma.

Además de las funcionalidades proporcionadas por las plataformas nativas, las aplicaciones móviles a menudo hacen uso de bibliotecas de terceros para agregar características adicionales. Estas bibliotecas pueden incluir funcionalidades como la integración con servicios en la nube, análisis de datos, autenticación de usuarios, entre otros.

En resumen, al crear una aplicación, se debe considerar si se utilizará una plataforma nativa o una plataforma multiplataforma, si se requerirá almacenamiento de datos local utilizando SQLite¹ y si se utilizarán bibliotecas de terceros para agregar funcionalidades adicionales. A continuación, se presenta una tabla comparativa de las herramientas más utilizadas para proyectos móviles (tomado de

https://rua.ua.es/dspace/bitstream/10045/136523/1/Estudio_y_benchmarking_de_tecnologias_de_creacion_de_Gonzalez_Torres_Ricardo.pdf)

Tecnología	Año salida	Plataforma	Tipo	Lenguaje	Interfaz Gráfica	% de uso a 2023
XCode	2003	iOS	Nativo	Swift	Si	3
Android Studio	2014	Android	Nativo	JAVA/Kotlin	Si	31 java/ 8 Kotlin
Flutter	2017	Multi	Híbrido	Dart	No	3
React Native	2015	Multi	Híbrido	JavaScript	No	33
Ionic	2013	Multi	Híbrido	TypeScript	No	18
Xamarin	2011	Multi	Híbrido	C#	No	14

¹ SQLite es una biblioteca de base de datos relacional que se utiliza comúnmente en aplicaciones móviles para almacenar y administrar datos localmente en el dispositivo del usuario. Proporciona una forma eficiente y confiable de almacenar datos estructurados.

REACT NATIVE

React Native es una opción popular para el desarrollo de aplicaciones móviles debido a varias razones:

Eficiencia de desarrollo: React Native permite escribir código una vez y ejecutarlo en múltiples plataformas, como iOS y Android. Esto reduce el tiempo y los recursos necesarios para desarrollar y mantener aplicaciones móviles.

Rendimiento nativo: A diferencia de las aplicaciones híbridas, las aplicaciones desarrolladas con React Native se ejecutan directamente en el dispositivo, lo que proporciona un rendimiento similar al de las aplicaciones nativas.

Reutilización de código: React Native permite reutilizar componentes de interfaz de usuario entre diferentes plataformas, lo que facilita el desarrollo y la actualización de aplicaciones para múltiples sistemas operativos.

Comunidad activa: React Native cuenta con una gran comunidad de desarrolladores que comparten conocimientos, recursos y bibliotecas de código abierto. Esto facilita el aprendizaje y la resolución de problemas durante el desarrollo de aplicaciones.

Integración con tecnologías existentes: React Native se puede integrar fácilmente con código nativo existente, lo que permite aprovechar las funcionalidades y características específicas de cada plataforma.

Menor curva de aprendizaje: para aquellos desarrolladores Front-End, la curva de aprendizaje es más rápida, dado su previo conocimiento de JavaScript, HTML y CSS.

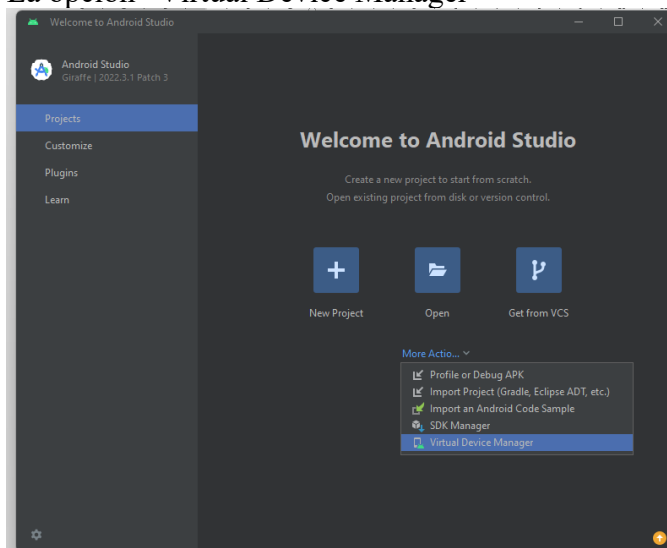
En resumen, utilizar React Native puede acelerar el desarrollo de aplicaciones móviles, ofrecer un rendimiento nativo y permitir la reutilización de código entre diferentes plataformas.

ENTORNO DE DESARROLLO

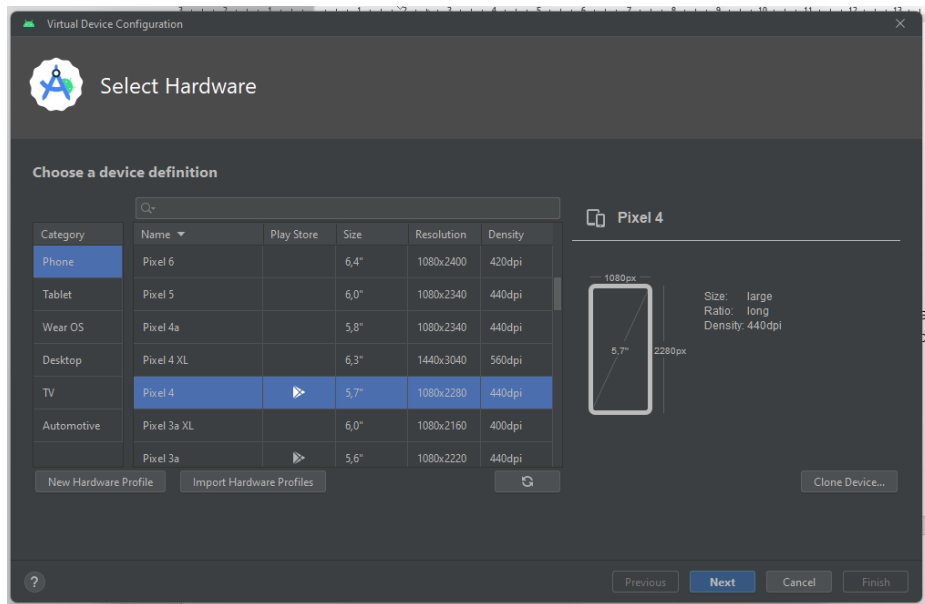
En este apartado se muestran los recursos necesarios para el desarrollo con React Native y su instalación, comprende el editor y sus dependencias necesarias, node.js como servidor, un emulador y el conversor de código para iOS y Android.

Configurar el entorno de desarrollo

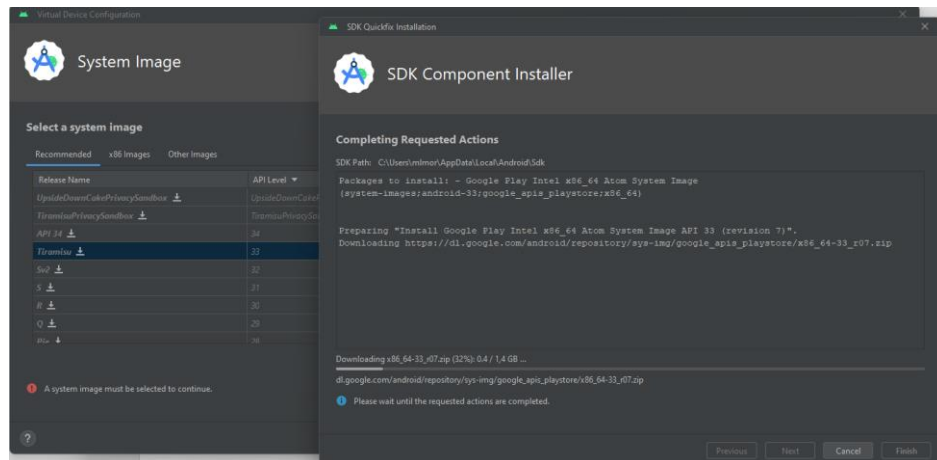
1. Instalar Visual Studio Code (VSC). En el asistente de instalación dejar marcada la opción “Agregar PATH...”
2. Ejecutar VSC e instalar dependencias, en la opción “Extensiones”:
 - a. “ES7 React/Redux/React-Native snippets”, suministra funciones nativas de React y Redux
 - b. “Simple React Snippets”, es un conjunto de ayudas
 - c. “Auto Close Tag”, ayuda para la creación de etiquetas (tags)
 - d. “Paste JSON as Code” para manipular JSON, es posible que ya esté instalado.
 - e. “TypeScript importer”
3. Instalar Android Studio. Desde la página oficial de Android Studio “developer.android.com/studio”.
 - a. Dejar habilitado “Android Virtual Device”
4. Instalar Node.js, de “node.org/es/download/”. Descargar la última versión. Dejar las opciones por defecto. En consola ejecutar “node --version” para verificar.
5. Instalar Postman (postman.com/downloads/), para realizar peticiones HTTP
6. Crear un emulador
 - a. Ejecutar Android Studio
 - b. La opción “Virtual Device Manager”



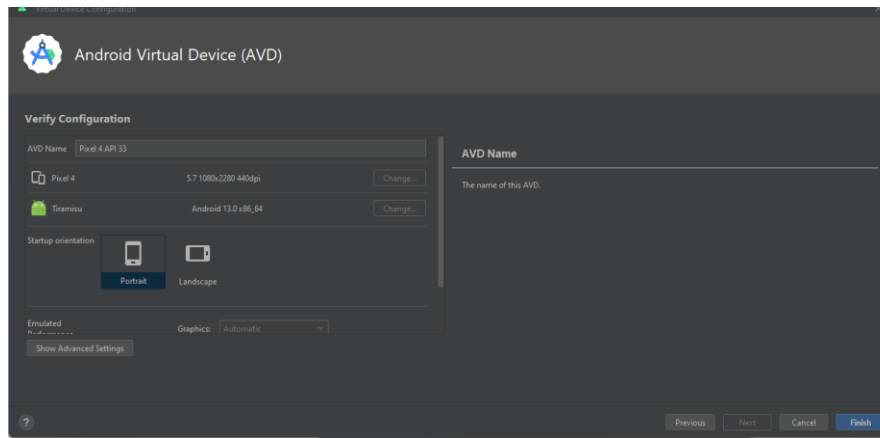
- c. “Create Device”, seleccionar “Phone” y buscar un emulador que tenga los servicios de Google Service para probar las aplicaciones sin restricciones (icono de Play Store)



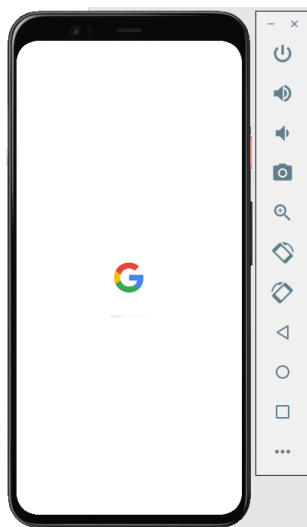
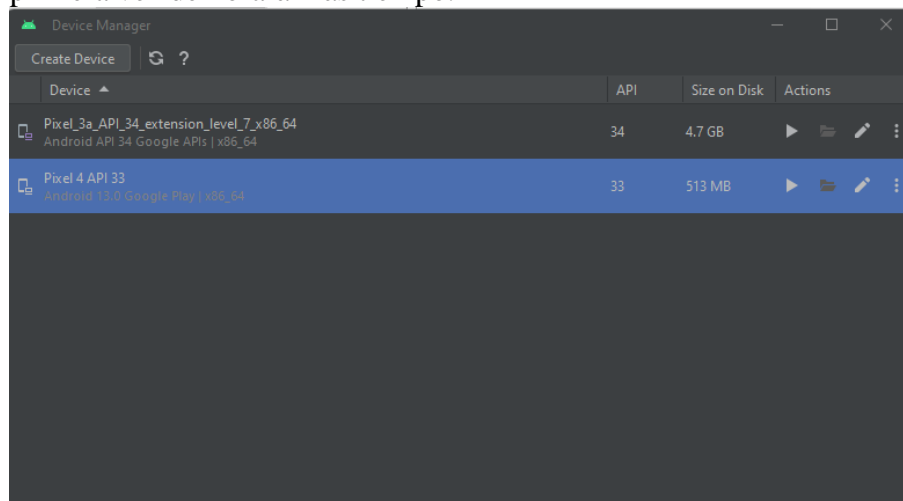
- d. Botón “Next” y descargar una versión, en este caso “Tiramisu”, descargar la API correspondiente:



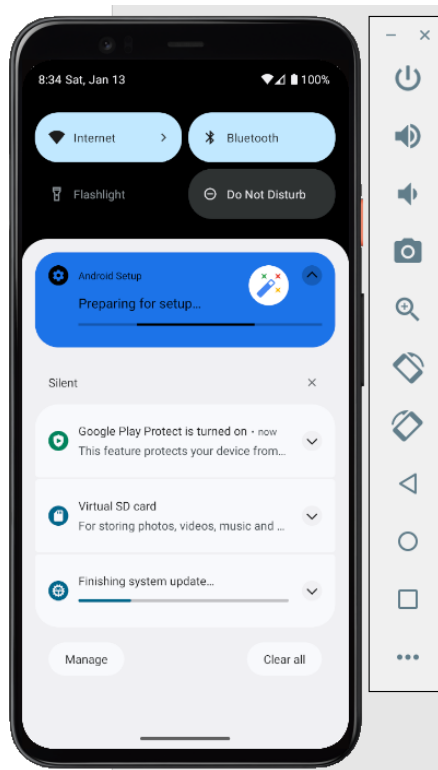
- e. Puede dejar las opciones por defecto y botón “Finish”



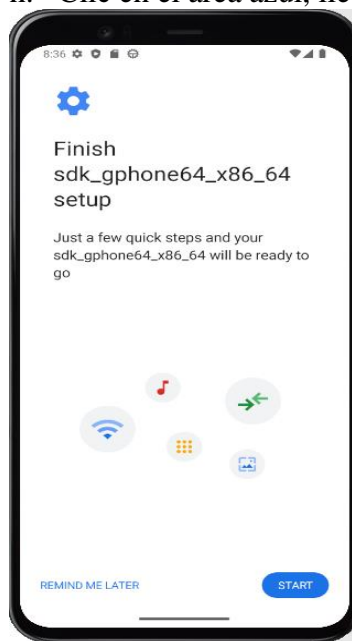
- f. Para ejecutar el emulador en la punta de flecha de la columna “Actions”, la primera vez demorará más tiempo.



- g. En la parte superior del emulador, deslizar para visualizar las notificaciones, esperar hasta que termine la configuración.



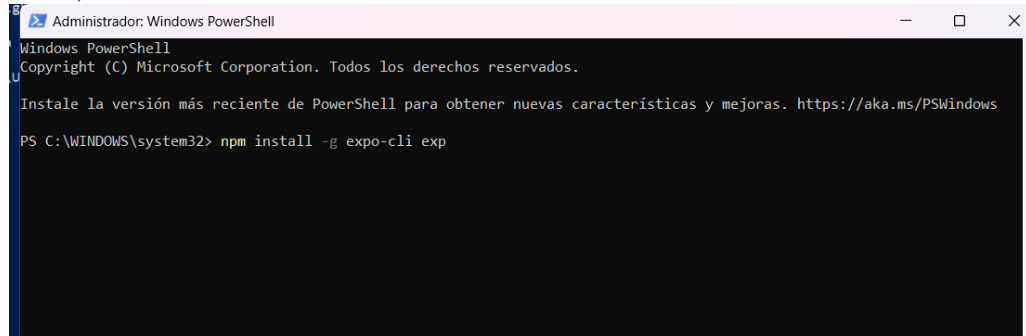
h. Clic en el área azul, lleva a la siguiente pantalla



i. Mas adelante, ingresar en el emulador, su correo Gmail

7. Instalar Expo CLI, Una interfaz de línea de comandos que permite montar un servidor para correr el proyecto en localmente, construir proyectos sencillos y publicarlo. En Google buscar “expo install” y el enlace “Como instalar Expo (React Native) – gists – GitHub”

- a. Copiar el comando “npm install -g expo-cli exp” y ejecutarlo en Power Shell, como administrador.



```
Administrador: Windows PowerShell
Windows PowerShell
Copyright (C) Microsoft Corporation. Todos los derechos reservados.

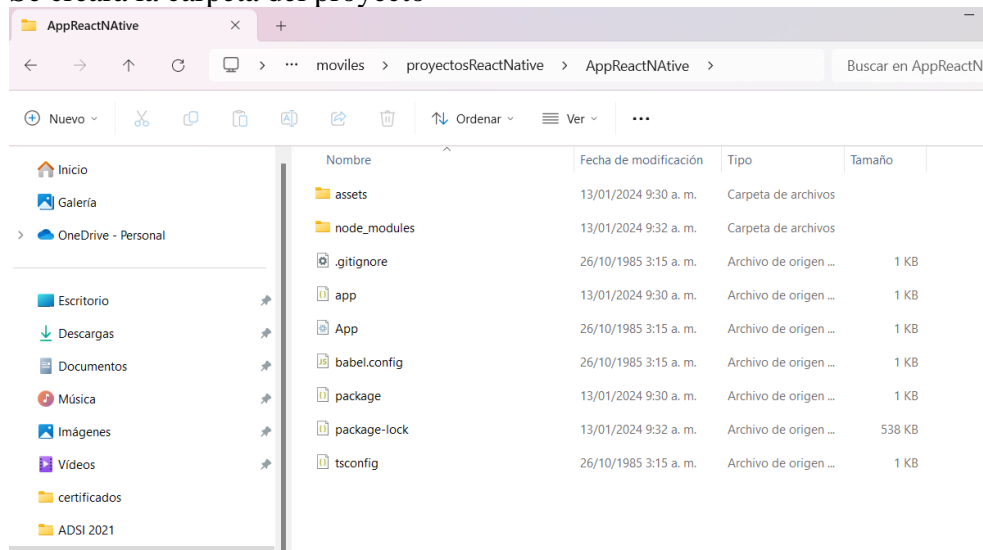
Instale la versión más reciente de PowerShell para obtener nuevas características y mejoras. https://aka.ms/PSWindows

PS C:\WINDOWS\system32> npm install -g expo-cli exp
```

- b. Verificar con “expo --version”.
- c. Si hay problemas con versiones desactualizadas, ejecutar:
- npm uninstall -g expo-cli
 - npm install --save-dev expo-cli
 - Si ya tiene un proyecto, reemplace en el archivo “package.json”:
“scripts”: { “start”: “expo start” } por “scripts”: { “start”: “npx expo start” }

CREAR UNA APP

1. Prepara una carpeta para el desarrollo del proyecto ejecutar VSC, abrir una consola y verificar que la ruta corresponda a esa carpeta.
2. En Google, buscar “react native install typescript” (ir a <https://reactnative.dev/docs/typescript>), para obtener los comandos a ejecutar.
3. En la consola de VSC, ejecutar el comando “npx create-expo-app --template” (se sugiere escribir el comando, no copiarlo)
 - a. Seleccionar con tab la opción “Blank (TypeScript)”
 - b. Darle un nombre a la aplicación y enter.
 - c. Se creará la carpeta del proyecto

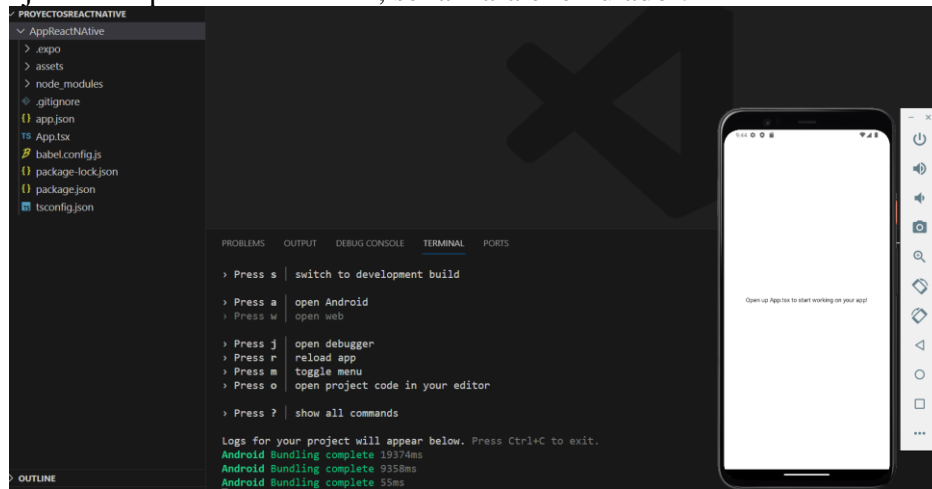


- i. En la carpeta “assets” se tienen los recursos del sistema (imágenes, etc.)
- ii. “App.ts” es el archivo inicial de la aplicación
- iii. En “package.json” se tienen las dependencias a utilizar en el desarrollo del proyecto.

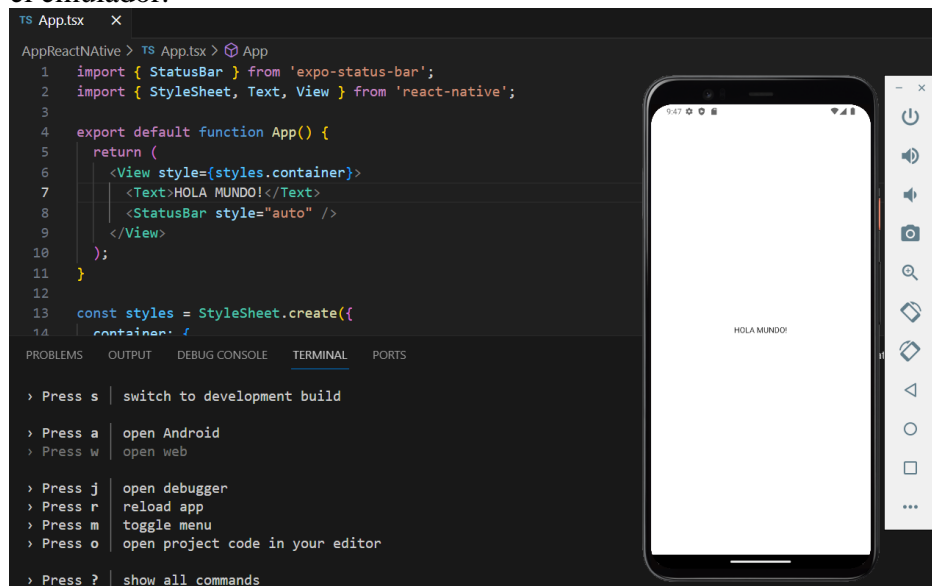
Ejecutar la App en un emulador

1. En la terminal de VSC
 - a. Cambiar a la carpeta de proyecto: “cd AppReactNative”

- b. ejecutar “npm run Android”, se lanzara el emulador.



- c. ¡En el archivo “App.tsx” cambiar el mensaje dentro de la etiqueta <Text> a “HOLA MUNDO!” y guardar (Ctl-S), inmediatamente cambiara el texto en el emulador.



Ejecutar la App en un dispositivo físico

1. En el dispositivo, ir a tienda, para Android es Play Store, buscar “expo go” e instalarla.
2. En la terminal de VSC, ejecutar “expo start”. Si le solicita dependencias adicionales, pulsar “Y” para instalarlas. Se generará un código QR.
3. Abrir la aplicación Expo Go en el dispositivo.
4. escanear el código QR con el dispositivo. Los dos equipos deben estar en la misma red WiFi.

COMPONENTES CENTRALES

El módulo principal, que se ejecuta al lanzar la aplicación es “App”, dentro de él se encuentran las “Views” que son las diferentes pantallas donde se realiza la navegación. Dentro de los Views hay componentes. Los views se relacionan con “styles”, que son los estilos aplicados a cada vista.

Crear una ventana de inicio

Buscar en Google “React native components”, en “Core Components” para tener una guía al construir el código.

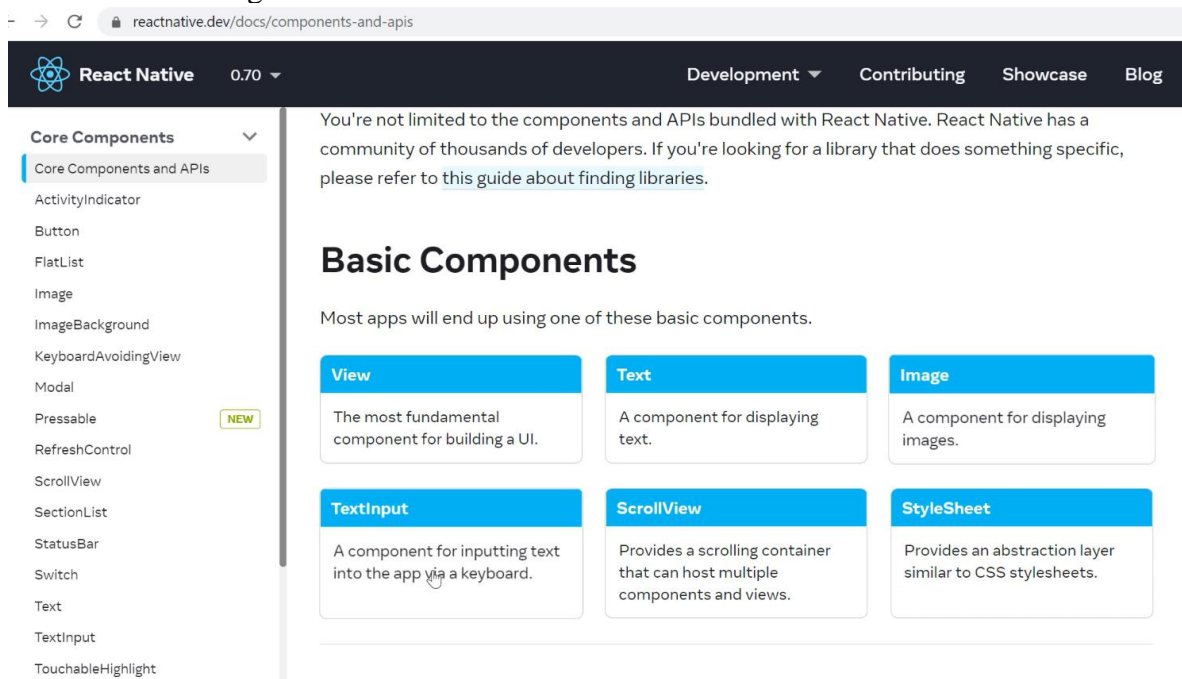


Imagen de Fondo

Copie los archivos de imágenes suministrados, en la carpeta “assets”

Modifique el código del archivo App.tsx como se muestra:

```
import { StatusBar } from 'expo-status-bar';
import { StyleSheet, Text, View, Image } from 'react-native';

export default function App() {
  return (
    <View style={styles.container}>
      <Image
        source={require('./assets/chef.jpg')}
        style={styles.imageBackground}
      />
    </View>
  );
}
```

```

    </View>
  );
}

const styles = StyleSheet.create({
  container: {
    flex: 1,
    backgroundColor: '#fff',
  },
  imageBackground: {
    width: '100%',
    height: '100%',
  },
});
});

```

Guarde los cambios y la aplicación cambiará así:



Formulario de ingreso

Agregue una vista interna, con sus estilos. Modifique el código así:

```

import { StatusBar } from 'expo-status-bar';
import { StyleSheet, Text, View, Image } from 'react-native';

export default function App() {
  return (
    <View style={styles.container}>
      <Image

```

```

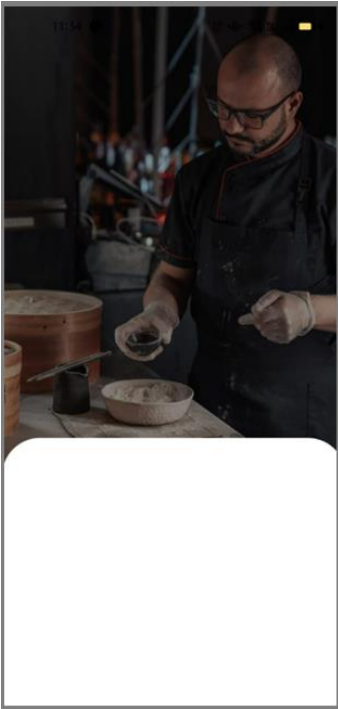
        source={require('./assets/chef.jpg')}
        style={styles.imageBackground}
      />
      <View style={styles.form}>
    </View>

    </View>
  );
}

const styles = StyleSheet.create({
  container: {
    flex: 1,
    backgroundColor: 'black',
  },
  imageBackground: {
    width: '100%',
    height: '100%',
    opacity: 0.7,
    bottom: '30%',
  },
  form:{
    width: '100%',
    height: '40%',
    backgroundColor: 'white',
    position: 'absolute',
    bottom: 0,
    borderTopLeftRadius: 40,
    borderTopRightRadius: 40,
  }
});

```

La App contendrá la caja del formulario:



Inserte la siguiente View justo después de la imagen de fondo:

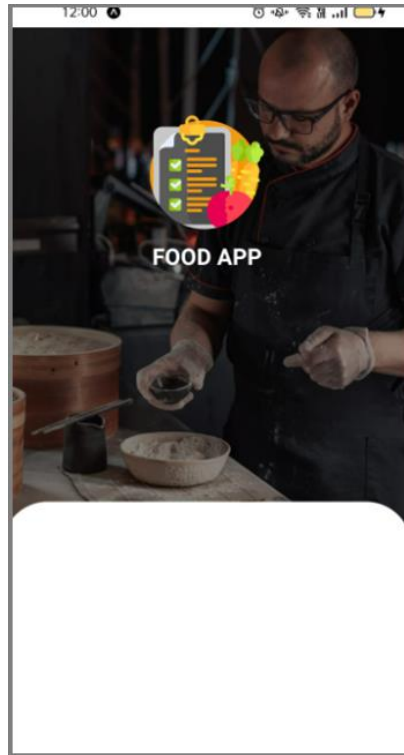
```
<View style={styles.logoContainer}>
  <Image
    source={require('./assets/logo.png')}
    style={styles.logoImage}
  />
  <Text style={styles.logoText}>FOOD APP</Text>
</View>
```

Y los siguientes estilos:

```
logoContainer:{
  position: 'absolute',
  alignSelf: 'center',
  top: '15%',
},
logoImage: {
  width: 100,
  height: 100,
},
logoText: {
  color: 'white',
  textAlign: 'center',
  fontSize: 20,
  marginTop: 10,
```

```
fontWeight: 'bold',  
}
```

Se obtendrá la siguiente pantalla:



Elementos del formulario

Modificar el archivo “App.tsx”, así:

```
import { StatusBar } from 'expo-status-bar';  
import { StyleSheet, Text, View, Image, TextInput, Button, ToastAndroid }  
from 'react-native';  
  
export default function App() {  
  return (  
    <View style={styles.container}>  
      <Image  
        source={require('./assets/chef.jpg')}  
        style={styles.imageBackground}  
      />  
      <View style={styles.logoContainer}>  
        <Image  
          source={require('./assets/logo.png')}  
          style={styles.logoImage}  
        />  
      </View>  
    </View>  
  );  
}
```

```

        <Text style={styles.logoText}>FOOD APP</Text>
    </View>

    <View style={styles.form}>
        <Text style={styles.formText}>INGRESAR</Text>
        <View style={styles.formInput}>
            <Image style={styles.formIcon}
                source={require('./assets/email.png')}
            />
            <TextInput
                style={styles.formTextInput}
                placeholder='Correo electrónico'
                keyboardType='email-address'
            />
        </View>
        <View style={styles.formInput}>
            <Image style={styles.formIcon}
                source={require('./assets/password.png')}
            />
            <TextInput
                style={styles.formTextInput}
                placeholder='Contraseña'
                keyboardType='default'
                secureTextEntry={true}
            />
        </View>

        <View style={{ marginTop: 30 }}>
            <Button
                title='ENTRAR'
                onPress={() => ToastAndroid.show('CLICK', ToastAndroid.LONG)}
                color={'orange'}
            />
        </View>

        <View style={styles.formRegister}>
            <Text>¿No tienes cuenta?</Text>
            <Text style={styles.formRegisterText}>Regístrate</Text>
        </View>

    </View>
</View>
);
}

```

```
const styles = StyleSheet.create({
  container: {
    flex: 1,
    backgroundColor: 'black',
  },
  imageBackground: {
    width: '100%',
    height: '100%',
    opacity: 0.7,
    bottom: '30%',
  },
  form: {
    width: '100%',
    height: '40%',
    backgroundColor: 'white',
    position: 'absolute',
    bottom: 0,
    borderTopLeftRadius: 40,
    borderTopRightRadius: 40,
    padding: 30,
  },
  formText: {
    fontWeight: 'bold',
    fontSize: 16,
  },
  formIcon: {
    width: 25,
    height: 25,
    marginTop: 5,
  },
  formInput: {
    flexDirection: 'row',
    marginTop: 30,
  },
  formTextInput: {
    flex: 1,
    borderBottomWidth: 1,
    borderBottomColor: '#AAAAAA',
    marginLeft: 15,
  },
  formRegister: {
    flexDirection: 'row',
```

```

      justifyContent: 'center',
      marginTop: 30,
    },

    formRegisterText: {
      fontStyle: 'italic',
      color: 'orange',
      borderBottomWidth: 1,
      borderBottomColor: 'orange',
      fontWeight: 'bold',
      marginLeft: 10,
    },
    logoContainer: {
      position: 'absolute',
      alignSelf: 'center',
      top: '15%',
    },
    logoImage: {
      width: 100,
      height: 100,
    },
    logoText: {
      color: 'white',
      textAlign: 'center',
      fontSize: 20,
      marginTop: 10,
      fontWeight: 'bold',
    },
  },
});

```

Crear elementos personalizados

Crear una carpeta “src” dentro de la carpeta principal

Dentro de “src” crear dos carpetas “components” y “views”

Dentro de “components” crear un archivo “RoundedButton.tsx”

En el archivo digitar “racfc” y aceptar el snippet:

```

import React from 'react'

export const RoundedButton = () => {
  return (
    <div>RoundedButton</div>
  )
}

```

```
}
```

Eliminar la línea con la etiqueta <div> y modificar:

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

export const RoundedButton = () => {
  return (
    <TouchableOpacity
      onPress={() => {}}

    >
      <Text>ENTRAR</Text>
    </TouchableOpacity>
  )
}
```

En el archivo “App.tsx”, eliminar la etiqueta Button y cambiar por el elemento personalizado:

```
<View style={{ marginTop: 30 }}>
  <RoundedButton />

</View>
```

Observe que se debe crear la línea del import:

```
import { RoundedButton } from './src/components/RoundedButton';
```

Si no, insertarla manualmente en la sección de imports.
Modificar “RoundedButton.tsx” así:

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

export const RoundedButton = () => {
  return (
    <TouchableOpacity
      style={styles.RoundedButton}
      onPress={() => { }}

    >
      <Text style={styles.textButton} >ENTRAR</Text>
    </TouchableOpacity>
  )
}
```

```

    )
  }

  const styles = StyleSheet.create({
    RoundedButton: {
      width: '100%',
      height: 50,
      backgroundColor: 'orange',
      alignItems: 'center',
      justifyContent: 'center',
      borderRadius: 15,
    },
    textButton: {
      color: 'white',
    }
  });

```

Pasando parámetros al componente

Se crea una interfase, por convención se llama “Props”, en “RoundedButton.tsx”, después de los imports:

```

interface Props {
  text: string;
  onPress: () => void,
}

```

Modificar el elemento RoundedButton:

```

export const RoundedButton = ({ text, onPress}: Props) => {
  return (
    <TouchableOpacity
      style={styles.RoundedButton}
      onPress={() => onPress()}
    >
      <Text style={styles.textButton} >{ text}</Text>
    </TouchableOpacity>
  )
}

```

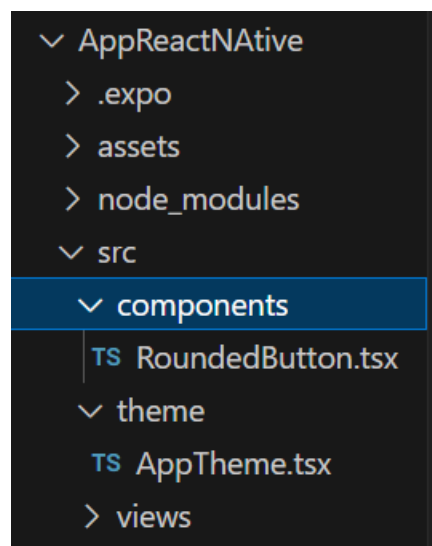
En el archivo “App.tsx” para pasar los parámetros, modificar la etiqueta <RoundedButton> así:

```
<View style={{ marginTop: 30 }}>
  <RoundedButton text='ENTRAR' onPress={ () => ToastAndroid.show('HOLA!',
ToastAndroid.SHORT)} />
</View>
```

Verifique el funcionamiento del código anterior, ¡debe aparecer una pequeña ventana emergente con el mensaje “HOLA!”

Crear estilos globales

Dentro de la carpeta “src” crear una carpeta “theme” y dentro de esta el archivo “AppTheme.tsx”



En “AppTheme.tsx” el siguiente código:

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

export const MyColors = {
  background: '#EEEEEE',
  primary: '#F4991A',
  secondary: '# E14D2A',
}
```

En “RoundedButton.tsx”, cambiar el color de fondo:


```
backgroundColor: MyColors.primary,
```

Se debe generar el import:

```
import { MyColors } from '../theme/AppTheme';
```

NAVEGACION ENTRE PANTALLAS

Detener el servidor (ctrl+c).

Instalar manualmente. Consultar en “reactnative.dev/docs/navigation” y ejecutar en la terminal el comando:

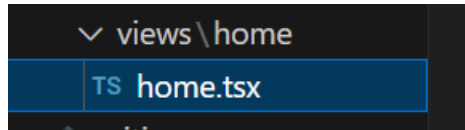
```
npm install @react-navigation/native @react-navigation/native-stack
```

Después, instalar las dependencias con “expo”:

```
npx expo install react-native-screens react-native-safe-area-context
```

Separando las pantallas

En la carpeta “views” crear otra carpeta “home” y en ella el archivo “home.tsx”:



En este archivo teclear “rafce” para generar el snippet, cambiar por HomeScreen:

```
import React from 'react'

export const HomeScreen = () => {
  return (
    <div>Screen</div>
  )
}
```

Cortar todo el código, desde el return con la pantalla inicial, desde “App.tsx” en “home.tsx”, quedará así:

```
import React from 'react'
import { View, Text, StyleSheet, Image, TextInput, ToastAndroid } from
'react-native'
import { RoundedButton } from '../components/RoundedButton';

export const HomeScreen = () => {

  return (
    <View style={styles.container}>
      <Image
        source={require('./assets/chef.jpg')}
        style={styles.imageBackground}
```

```

/>
<View style={styles.logoContainer}>
  <Image
    source={require('./assets/logo.png')}
    style={styles.logoImage}
  />
  <Text style={styles.logoText}>FOOD APP</Text>
</View>

<View style={styles.form}>
  <Text style={styles.formText}>INGRESAR</Text>
  <View style={styles.formInput}>
    <Image style={styles.formIcon}
      source={require('./assets/email.png')}
    />
    <TextInput
      style={styles.formTextInput}
      placeholder='Correo electrónico'
      keyboardType='email-address'
    />
  </View>
  <View style={styles.formInput}>
    <Image style={styles.formIcon}
      source={require('./assets/password.png')}
    />
    <TextInput
      style={styles.formTextInput}
      placeholder='Contraseña'
      keyboardType='default'
      secureTextEntry={true}
    />
  </View>

  <View style={{ marginTop: 30 }}>
    <RoundedButton text='ENTRAR' onPress={() =>
ToastAndroid.show('HOLA!', ToastAndroid.SHORT)} />
  </View>

  <View style={styles.formRegister}>
    <Text>¿No tienes cuenta?</Text>
    <Text style={styles.formRegisterText}>Regístrate</Text>
  </View>

</View>
</View>

```

```

    );
}

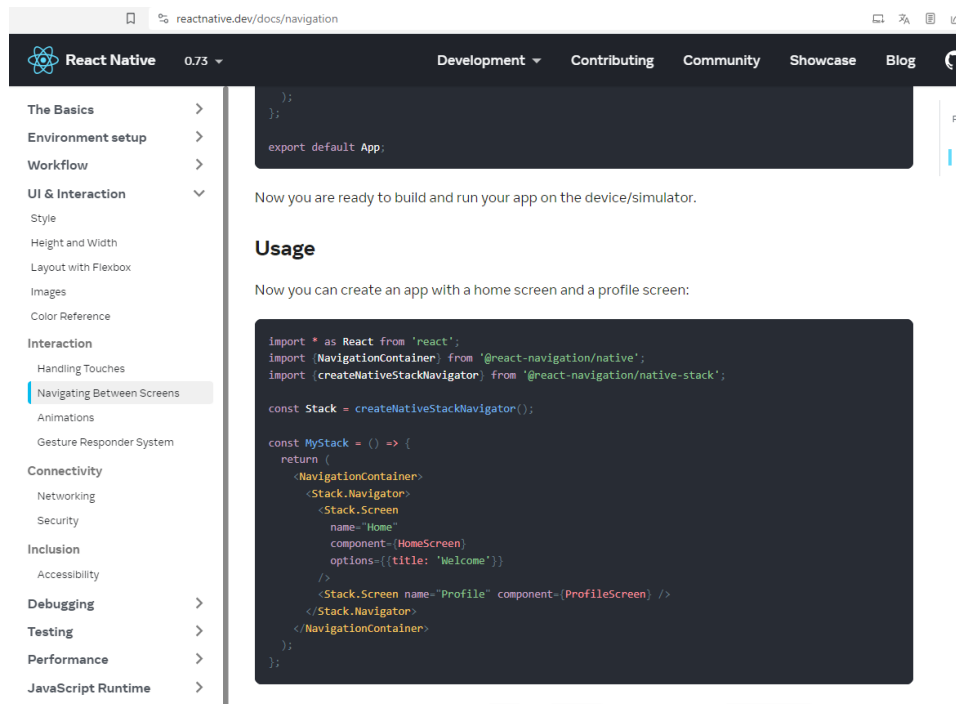
const styles = StyleSheet.create({
  container: {
    flex: 1,
    backgroundColor: 'black',
  },
  imageBackground: {
    width: '100%',
    height: '100%',
    opacity: 0.7,
    bottom: '30%',
  },
  form: {
    width: '100%',
    height: '40%',
    backgroundColor: 'white',
    position: 'absolute',
    bottom: 0,
    borderTopLeftRadius: 40,
    borderTopRightRadius: 40,
    padding: 30,
  },
  formText: {
    fontWeight: 'bold',
    fontSize: 16,
  },
  formIcon: {
    width: 25,
    height: 25,
    marginTop: 5,
  },
  formInput: {
    flexDirection: 'row',
    marginTop: 30,
  },
  formTextInput: {
    flex: 1,
    borderBottomWidth: 1,
    borderBottomColor: '#AAAAAA',
    marginLeft: 15,
  },
});

```

```
formRegister: {
  flexDirection: 'row',
  justifyContent: 'center',
  marginTop: 30,
},

formRegisterText: {
  fontStyle: 'italic',
  color: 'orange',
  borderBottomWidth: 1,
  borderBottomColor: 'orange',
  fontWeight: 'bold',
  marginLeft: 10,
},
logoContainer: {
  position: 'absolute',
  alignSelf: 'center',
  top: '15%',
},
logoImage: {
  width: 100,
  height: 100,
},
logoText: {
  color: 'white',
  textAlign: 'center',
  fontSize: 20,
  marginTop: 10,
  fontWeight: 'bold',
},
});
```

Desde la página de Real Native, copiar el código en el apartado “Usage”:



Y reemplazar el archivo “App.tsx”. Cambiar algunos nombres para que el código quede así:

```
import * as React from 'react';
import {NavigationContainer} from '@react-navigation/native';
import {createNativeStackNavigator} from '@react-navigation/native-stack';
import { HomeScreen } from './src/views/home/home';

const Stack = createNativeStackNavigator();

const App = () => {
  return (
    <NavigationContainer>
      <Stack.Navigator screenOptions={{
        headerShown: false
      }}>

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

        />
        { /*<Stack.Screen name="Profile" component={ProfileScreen} /> */}
      </Stack.Navigator>
    </NavigationContainer>
  );
};
```

```
export default App;
```

Al compilar y ejecutar expo, se genera un error debido a las rutas de las imágenes, cambiar así:

```
<Image
  source={require('../../assets/chef.jpg')}
  style={styles.imageBackground}
/>
<View style={styles.logoContainer}>
  <Image
    source={require('../../assets/logo.png')}
    style={styles.logoImage}
  />
```

Y lo mismo para los demás recursos.

Crear pantalla de registro

Dentro de la carpeta “View” crear una carpeta “register” y en ella un archivo “Register.tsx”:

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

export const RegisterScreen = () => {
  return (
    <View style={{flex:1, justifyContent: 'center', alignItems: 'center'}}>
      <Text>RegisterScreen</Text>
    </View>
  )
}
```

En “App.tsx” agregar la otra pantalla:

```
import * as React from 'react';
import { NavigationContainer } from '@react-navigation/native';
import { createNativeStackNavigator } from '@react-navigation/native-stack';
import { HomeScreen } from '../src/views/home/home';
import { RegisterScreen } from '../src/views/register/Register';

export type RootStackParamList = {
```

```

    HomeScreen: undefined;
    RegisterScreen: undefined;
  };

const Stack = createNativeStackNavigator <RootStackParamList>
();

const App = () => {
  return (
    <NavigationContainer>
      <Stack.Navigator screenOptions={{
        headerShown: false
      }}>
        <Stack.Screen
          name="HomeScreen"
          component={HomeScreen}
        />
        <Stack.Screen
          name="RegisterScreen"
          component={RegisterScreen}
          options={{
            headerShown: true,
            title: 'Registro',
          }}
        />
      </Stack.Navigator>
    </NavigationContainer>
  );
};

export default App;

```

En “home.tsx” agregar el import y la constante navigation:

```

import { StackNavigationProp } from '@react-navigation/stack';
import { RootStackParamList } from '../../App';

export const HomeScreen = () => {

  const navigation =
  useNavigation<StackNavigationProp<RootStackParamList>>();

  return (

```


Debe ejecutar en la terminal el siguiente comando para instalar el complemento “npm i @react-navigation/stack”. Debe verse registrado en el archivo “package.json”.

Agregar componentes a la pantalla de registro

Reutilizar el código de la pantalla de home (“home.tsx”) en “Register.tsx” haciendo los cambios necesarios en los componentes y los estilos:

```
import React from 'react'
import { useNavigation } from '@react-navigation/native';
import { View, Text, StyleSheet, Image, TextInput, ToastAndroid, Touchable,
TouchableOpacity } from 'react-native';
import { RoundedButton } from '../../components/RoundedButton';
import { StackNavigationProp } from '@react-navigation/stack';
import { RootStackParamList } from '../../App';
export const RegisterScreen = () => {

  return (
    <View style={styles.container}>
      <Image
        source={require('../../assets/chef.jpg')}
        style={styles.imageBackground}
      />
      <View style={styles.logoContainer}>
        <Image
          source={require('../../assets/user_image.png')}
          style={styles.logoImage}
        />
        <Text style={styles.logoText}>SELECCIONA UNA IMAGEN</Text>
      </View>

      <View style={styles.form}>
        <Text style={styles.formText}>REGISTRARSE</Text>

        <View style={styles.formInput}>
          <Image style={styles.formIcon}
            source={require('../../assets/user.png')}
          />
          <TextInput
            style={styles.formTextInput}
            placeholder='Nombres'
            keyboardType='default'
          />
        </View>
      </View>
    </View>
  )
}
```

```
<View style={styles.formInput}>
  <Image style={styles.formIcon}
    source={require('../assets/my_user.png')}
  />
  <TextInput
    style={styles.formTextInput}
    placeholder='Apellidos'
    keyboardType='default'
  />
</View>

<View style={styles.formInput}>
  <Image style={styles.formIcon}
    source={require('../assets/email.png')}
  />
  <TextInput
    style={styles.formTextInput}
    placeholder='Correo electrónico'
    keyboardType='email-address'
  />
</View>

<View style={styles.formInput}>
  <Image style={styles.formIcon}
    source={require('../assets/phone.png')}
  />
  <TextInput
    style={styles.formTextInput}
    placeholder='Teléfono'
    keyboardType='numeric'
  />
</View>

<View style={styles.formInput}>
  <Image style={styles.formIcon}
    source={require('../assets/password.png')}
  />
  <TextInput
    style={styles.formTextInput}
    placeholder='Contraseña'
    keyboardType='default'
    secureTextEntry={true}
  />
</View>
```

```

        <View style={styles.formInput}>
          <Image style={styles.formIcon}
            source={require('../assets/confirm_password.png')}
          />
          <TextInput
            style={styles.formTextInput}
            placeholder='Confirmar Contraseña'
            keyboardType='default'
            secureTextEntry={true}
          />
        </View>
        <View style={{ marginTop: 30 }}>
          <RoundedButton text='CONFIRMAR' onPress={() =>
            ToastAndroid.show('HOLA!', ToastAndroid.SHORT)} />
        </View>

      </View>
    </View>
  );
}

const styles = StyleSheet.create({
  container: {
    flex: 1,
    backgroundColor: 'black',
  },
  imageBackground: {
    width: '100%',
    height: '100%',
    opacity: 0.7,
    bottom: '30%',
  },
  form: {
    width: '100%',
    height: '70%',
    backgroundColor: 'white',
    position: 'absolute',
    bottom: 0,
    borderTopLeftRadius: 40,
    borderTopRightRadius: 40,
    padding: 30,
  },
});

```

```
formText: {
  fontWeight: 'bold',
  fontSize: 16,
},

formIcon: {
  width: 25,
  height: 25,
  marginTop: 5,
},

formInput: {
  flexDirection: 'row',
  marginTop: 25,
},

formTextInput: {
  flex: 1,
  borderBottomWidth: 1,
  borderBottomColor: '#AAAAAA',
  marginLeft: 15,
},

formRegister: {
  flexDirection: 'row',
  justifyContent: 'center',
  marginTop: 30,
},

formRegisterText: {
  fontStyle: 'italic',
  color: 'orange',
  borderBottomWidth: 1,
  borderBottomColor: 'orange',
  fontWeight: 'bold',
  marginLeft: 10,
},

logoContainer: {
  position: 'absolute',
  alignSelf: 'center',
  top: '5%',
  alignItems: 'center',
},

logoImage: {
```

```

    width: 100,
    height: 100,
  },
  logoText: {
    color: 'white',
    textAlign: 'center',
    fontSize: 20,
    marginTop: 10,
    fontWeight: 'bold',
  },
});

```

Obtendrá una pantalla como la siguiente:



Capturar valores de los inputs

Importar useState en “home.tsx”:

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

Agregar constantes para cada campo del formulario, antes de la constante navigation:

```
const [email, setEmail] = useState('');
const [password, setPassword] = useState('');
const navigation =
useNavigation<StackNavigationProp<RootStackParamList>>();
```

Relacionar las constantes con los inputs del formulario y controlar el evento:

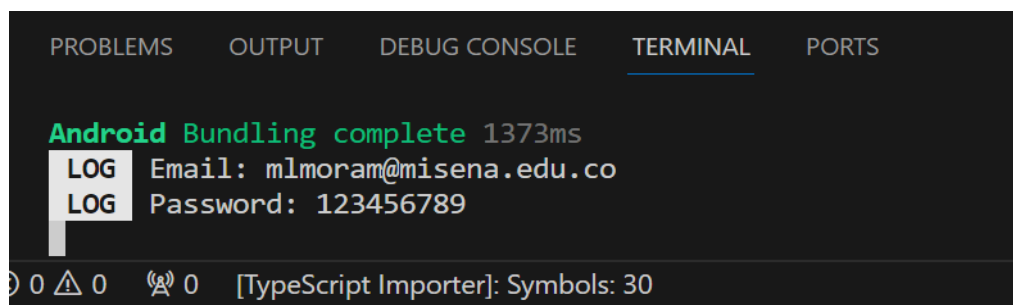
```
<TextInput
  style={styles.formTextInput}
  placeholder='Correo electrónico'
  keyboardType='email-address'
  value={email}
  onChangeText={ text => setEmail(text)}
/>
```

```
<TextInput
  style={styles.formTextInput}
  placeholder='Contraseña'
  keyboardType='default'
  secureTextEntry={true}
  value={password}
  onChangeText={ text => setPassword(text)}
/>
```

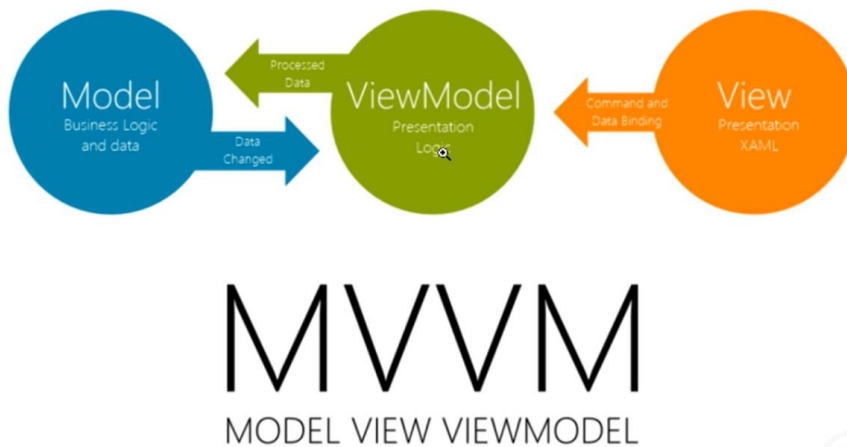
Para probar, cambiar el evento del botón para enviar los valores a la terminal:

```
<View style={{ marginTop: 30 }}>
  <RoundedButton text='ENTRAR' onPress={() =>{
    console.log('Email: ' + email);
    console.log('Password: ' + password);
  }} />
</View>
```

SI todo va bien, en la terminal se mostrarán los valores digitados, al dar clic en el botón:



ORGANIZANDO EL CÓDIGO CON EL PATRÓN DE DISEÑO MVVM



Reorganizar las carpetas según las convenciones de Google

Capa de Presentación

Contendrá todo lo relacionado con las vistas.

- Crear una carpeta “Presentation” dentro de “src”.
- Arrastrar a ella las carpetas “components”, “theme” y “views”, debe detener la aplicación y cerrar VSC para que permita el cambio manual.
- Se deben cambiar las rutas de los recursos, agregando “./”, en “App.tsx” a
-

```
import { HomeScreen } from './src/Presentation/views/home/home';
import { RegisterScreen } from
'./src/Presentation/views/register/Register';
```

Capa de Dominio

- Dentro de “src” cree una carpeta “Domain”.

Capa de Datos

- Dentro de “src” cree una carpeta “Data”.

Pasar los userState a cada una de las carpetas, creando un archivo ‘viewModel.tsx’ en cada carpeta de ‘views’, cortar los userState de “Home.tsx” y en ‘viewModel.tsx’ construir un objeto así:

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

const HomeViewModel = () => {
  const [values, setValues] = useState({
    email:'',
    password:'',
```

```

    });

    const onChange = (property: string, value: any) => {
      setValues({...values, [property]: value});
    }

    return {
      ...values,
      onChange
    }
  }
}

export default HomeViewModel;

```

Nota: los tres puntos en el código realizan una “desestructuración”, es decir, separa los elementos de la estructura original.

En “Home.tsx” modificar:

```

import useViewModel from './ViewModel';

export const HomeScreen = () => {
  const {email, password, onChange}
} = useViewModel();
  const navigation = useNavigation<StackNavigationProp<RootStackParamList>>();

  return (

```

En los bloques <TextInput> de “Home.tsx”, cambiar:

```

    <TextInput
      style={styles.formTextInput}
      placeholder='Contraseña'
      keyboardType='default'
      secureTextEntry={true}
      value={password}
      onChangeText={ text => onChange('email', text)}
    />

```

```

    <TextInput
      style={styles.formTextInput}
      placeholder='Contraseña'

```



```

        keyboardType='default'
        secureTextEntry={true}
        value={password}
        onChangeText={ text => onChange('password', text)}
    />

```

Hay que confirmar que todo esté correcto, ingresando los datos y dando clic en el botón. El código funciona lo mismo que anteriormente, pero está organizado de una manera más profesional.

Crear un componente personalizado para los inputs

En la carpeta “components” crear un archivo “CusatomTextInput.tsx”, y en él generar código con ‘rací’, modificar así:

```

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

interface Props {
  image: any;
  placeholder: string;
  value: string;
  keyboardType: KeyboardType,
  secureTextEntry?: boolean,
  property: string,
  onChangeText: (property: string, value: any) => void,
}

export const CustomTextInput = ({
  image,
  placeholder,
  value,
  keyboardType,
  secureTextEntry = false,
  property,
  onChangeText
}: Props) => {
  return (
    <View style={styles.formInput}>
      <Image style={styles.formIcon}
        source={image}
      />
      <TextInput
        style={styles.formTextInput}

```

```

        placeholder={placeholder}
        keyboardType={keyboardType}
        value={value}
        onChangeText={text => onChangeText(property, text)}
        secureTextEntry={secureTextEntry}
      />
    </View>
  )
}

const styles = StyleSheet.create({
  formIcon: {
    width: 25,
    height: 25,
    marginTop: 5,
  },
  formInput: {
    flexDirection: 'row',
    marginTop: 30,
  },
  formTextInput: {
    flex: 1,
    borderBottomWidth: 1,
    borderBottomColor: '#AAAAAA',
    marginLeft: 15,
  }
})

```

Modificar el archivo “home.tsx”:

```

import React, { useState } from 'react';
import { useNavigation } from '@react-navigation/native';
import { View, Text, StyleSheet, Image, TextInput, ToastAndroid, Touchable,
TouchableOpacity } from 'react-native';
import { RoundedButton } from
'../../Presentation/components/RoundedButton';
import { StackNavigationProp } from '@react-navigation/stack';
import { RootStackParamList } from '../../App';
import useViewModel from './ViewModel';
import { CustomTextInput } from '../../components/CustomTextInput';

export const HomeScreen = () => {

```

```

const {email, password, onChange} = useViewModel();
const navigation =
useNavigation<StackNavigationProp<RootStackParamList>>()>();

return (
  <View style={styles.container}>
    <Image
      source={require('../assets/chef.jpg')}
      style={styles.imageBackground}
    />
    <View style={styles.logoContainer}>
      <Image
        source={require('../assets/logo.png')}
        style={styles.logoImage}
      />
      <Text style={styles.logoText}>FOOD APP</Text>
    </View>

    <View style={styles.form}>
      <Text style={styles.formText}>INGRESAR</Text>

      <CustomTextInput
        image= {require('../assets/email.png')}
        placeholder='Correo electrónico'
        keyboardType='email-address'
        property='email'
        onChangeText={onChange}
        value={email}
      />
      <CustomTextInput
        image= {require('../assets/password.png')}
        placeholder='Contraseña'
        keyboardType='default'
        property='password'
        onChangeText={onChange}
        value={password}
        secureTextEntry={true}
      />

      <View style={{ marginTop: 30 }}>
        <RoundedButton text='ENTRAR' onPress={() =>{
          console.log('Email: ' + email);
          console.log('Password: ' + password);

```

```

        }} />
      </View>

      <View style={styles.formRegister}>
        <Text>¿No tienes cuenta?</Text>
        <TouchableOpacity onPress={() =>
navigation.navigate('RegisterScreen')}>
          <Text
style={styles.formRegisterText}>Regístrate</Text>
        </TouchableOpacity>
      </View>

    </View>
  </View>
);
}

const styles = StyleSheet.create({
  container: {
    flex: 1,
    backgroundColor: 'black',
  },
  imageBackground: {
    width: '100%',
    height: '100%',
    opacity: 0.7,
    bottom: '30%',
  },
  form: {
    width: '100%',
    height: '40%',
    backgroundColor: 'white',
    position: 'absolute',
    bottom: 0,
    borderTopLeftRadius: 40,
    borderTopRightRadius: 40,
    padding: 30,
  },
  formText: {
    fontWeight: 'bold',
    fontSize: 16,
  },
  formRegister: {
    flexDirection: 'row',
    justifyContent: 'center',

```

```

        marginTop: 30,
      },
      formRegisterText: {
        fontStyle: 'italic',
        color: 'orange',
        borderBottomWidth: 1,
        borderBottomColor: 'orange',
        fontWeight: 'bold',
        marginLeft: 10,
      },
      logoContainer: {
        position: 'absolute',
        alignSelf: 'center',
        top: '15%',
      },
      logoImage: {
        width: 100,
        height: 100,
      },
      logoText: {
        color: 'white',
        textAlign: 'center',
        fontSize: 20,
        marginTop: 10,
        fontWeight: 'bold',
      },
    },
  ));

```

Ahora se están usando componentes personalizados.

Crear un archivo “ViewModel.tsx” en la carpeta “register”. Se recomienda cerrar otros archivos para evitar confusiones debido a sus similitudes.

El archivo quedará así:

```

import React, { useState } from 'react'

const RegisterViewModel = () => {
  const [values, setValues] = useState({
    name: '',
    lastname: '',
    phone: '',
    email: '',
    password: '',
    confirmPassword: '',
  })

```

```

    });

    const onChange = (property: string, value: any) => {
      setValues({ ...values, [property]: value });
    }

    const register = () => {
      console.log(JSON.stringify(values));
    }

    return {
      ...values,
      onChange,
      register
    }
  }
}

export default RegisterViewModel;

```

Observe que se modifica el evento para mostrar en la terminal los valores ingresados. Modificar el archivo “register.tsx” así:

```

import React from 'react'
import { View, Text, StyleSheet, Image, TextInput, ToastAndroid, Touchable, TouchableOpacity } from 'react-native';
import { RoundedButton } from
'../../../../Presentation/components/RoundedButton';
import useViewModel from './ViewModel';
import { CustomTextInput } from
'../../../../Presentation/components/CustomTextInput';

export const RegisterScreen = () => {
  const { name, lastname, phone, email, password, confirmPassword, onChange,
register } = useViewModel();

  return (
    <View style={styles.container}>
      <Image
        source={require('../../../assets/chef.jpg')}
        style={styles.imageBackground}
      />
      <View style={styles.logoContainer}>
        <Image

```

```

        source={require('../assets/user_image.png')}
        style={styles.logoImage}
      />
      <Text style={styles.logoText}>SELECCIONA UNA IMAGEN</Text>
    </View>

    <View style={styles.form}>
      <Text style={styles.formText}>REGISTRARSE</Text>

      <CustomTextInput
        image={require('../assets/user.png')}
        placeholder='Nombres'
        keyboardType='default'
        property='name'
        onChangeText={onChange}
        value={name}
      />

      <CustomTextInput
        image={require('../assets/my_user.png')}
        placeholder='Apellidos'
        keyboardType='default'
        property='lastname'
        onChangeText={onChange}
        value={lastname}
      />

      <CustomTextInput
        image={require('../assets/email.png')}
        placeholder='Correo electrónico'
        keyboardType='email-address'
        property='email'
        onChangeText={onChange}
        value={email}
      />

      <CustomTextInput
        image={require('../assets/phone.png')}
        placeholder='Teléfono'
        keyboardType='numeric'
        property='phone'
        onChangeText={onChange}
        value={phone}
      />

```

```

    <CustomTextInput
      image={require('../assets/password.png')}
      placeholder='Contraseña'
      keyboardType='default'
      property='password'
      onChangeText={onChange}
      value={password}
      secureTextEntry={true}
    />

    <CustomTextInput
      image={require('../assets/confirm_password.png')}
      placeholder='Confirmar Contraseña'
      keyboardType='default'
      property='confirmPassword'
      onChangeText={onChange}
      value={confirmPassword}
      secureTextEntry={true}
    />

    <View style={{ marginTop: 30 }}>
      <RoundedButton text='CONFIRMAR' onPress={() => register()} />
    </View>

  </View>
</View>
);
}

```

La parte de estilos no se cambia.

Al ingresar valores en el formulario y dar clic en el botón se reciben los valores en la terminal, en formato JSON:

```

at TypeScriptParserMixin.parseExprOps (C:\Users\mlmor\OneDrive\Documents\SENA\SENA_CEEI\ADSO\moviles\proye
tosReactNative\AppReactNative\node_modules\@babel\parser\lib\index.js:10659:23)
Reloading apps
android Bundling complete 39ms
LOG { "name": "Marco", "lastname": "Mora", "phone": "3153800609", "email": "mlmoram@misena.edu.co", "password": "12345
", "confirmPassword": "12345" }

```

Separar las hojas de estilos

En la carpeta “home” crear un archivo “Styles.tsx”, cortar y pegar los estilos dentro de “home.tsx” y modificar así:


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

```
const HomeStyles = StyleSheet.create({
  container: {
    flex: 1,
    backgroundColor: 'black',
  },
  imageBackground: {
    width: '100%',
    height: '100%',
    opacity: 0.7,
    bottom: '30%',
  },
  form: {
    width: '100%',
    height: '40%',
    backgroundColor: 'white',
    position: 'absolute',
    bottom: 0,
    borderTopLeftRadius: 40,
    borderTopRightRadius: 40,
    padding: 30,
  },
  formText: {
    fontWeight: 'bold',
    fontSize: 16,
  },
  formRegister: {
    flexDirection: 'row',
    justifyContent: 'center',
    marginTop: 30,
  },
  formRegisterText: {
    fontStyle: 'italic',
    color: 'orange',
    borderBottomWidth: 1,
    borderBottomColor: 'orange',
    fontWeight: 'bold',
    marginLeft: 10,
  },
  logoContainer: {
    position: 'absolute',
    alignSelf: 'center',
    top: '15%',
  },
});
```

```

    },
    logoImage: {
      width: 100,
      height: 100,
    },
    logoText: {
      color: 'white',
      textAlign: 'center',
      fontSize: 20,
      marginTop: 10,
      fontWeight: 'bold',
    },
  },
});

export default HomeStyles;

```

Modificar el archivo “home.tsx”:

```

import React, { useState } from 'react';
import { useNavigation } from '@react-navigation/native';
import { View, Text, StyleSheet, Image, TextInput, ToastAndroid, Touchable,
TouchableOpacity } from 'react-native';
import { RoundedButton } from
'../../Presentation/components/RoundedButton';
import { StackNavigationProp } from '@react-navigation/stack';
import { RootStackParamList } from '../../App';
import useViewModel from './ViewModel';
import { CustomTextInput } from '../../components/CustomTextInput';
import styles from './Styles';

export const HomeScreen = () => {
  const {email, password, onChange} = useViewModel();
  const navigation =
useNavigation<StackNavigationProp<RootStackParamList>>();

  return (
    <View style={styles.container}>
      <Image
        source={require('../../assets/chef.jpg')}
        style={styles.imageBackground}
      />
      <View style={styles.logoContainer}>
        <Image
          source={require('../../assets/logo.png')}
          style={styles.logoImage}

```

```

        />
        <Text style={styles.logoText}>FOOD APP</Text>
    </View>

    <View style={styles.form}>
        <Text style={styles.formText}>INGRESAR</Text>

        <CustomTextInput
            image= {require('../assets/email.png')}
            placeholder='Correo electrónico'
            keyboardType='email-address'
            property='email'
            onChangeText={onChange}
            value={email}
        />
        <CustomTextInput
            image= {require('../assets/password.png')}
            placeholder='Contraseña'
            keyboardType='default'
            property='password'
            onChangeText={onChange}
            value={password}
            secureTextEntry={true}
        />

        <View style={{ marginTop: 30 }}>
            <RoundedButton text='ENTRAR' onPress={() =>{
                console.log('Email: ' + email);
                console.log('Password: ' + password);
            }} />
        </View>

        <View style={styles.formRegister}>
            <Text>¿No tienes cuenta?</Text>
            <TouchableOpacity onPress={() =>
navigation.navigate('RegisterScreen')}>
                <Text
style={styles.formRegisterText}>Regístrate</Text>
            </TouchableOpacity>
        </View>

    </View>
</View>
);

```

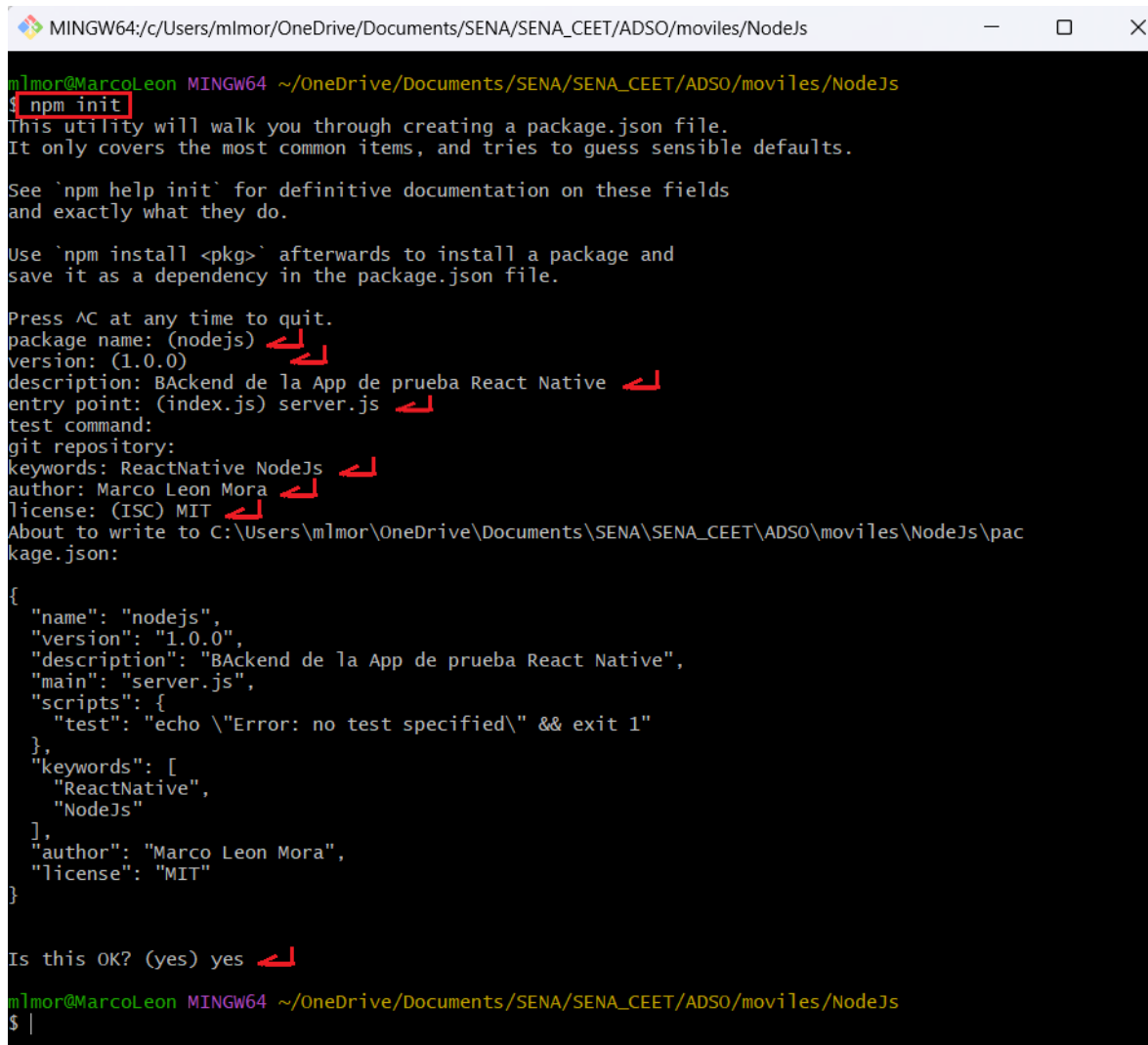
```
}
```

Realizar lo mismo en la carpeta “registro”.

MONTANDO UN SERVIDOR NODE JS

Asegúrese de tener instalado Node JS

Definir la carpeta que contendrá el proyecto, en ella abrir la consola Git Bash y ejecutar el comando “npm init”, ingrese la información suministrada, como se muestra en la siguiente imagen:



```
MINGW64/c:/Users/mlmor/OneDrive/Documents/SENA/SENA_CEET/ADSO/moviles/NodeJs
mlmor@MarcoLeon MINGW64 ~/OneDrive/Documents/SENA/SENA_CEET/ADSO/moviles/NodeJs
$ npm init
This utility will walk you through creating a package.json file.
It only covers the most common items, and tries to guess sensible defaults.

See 'npm help init' for definitive documentation on these fields
and exactly what they do.

Use 'npm install <pkg>' afterwards to install a package and
save it as a dependency in the package.json file.

Press ^C at any time to quit.
package name: (nodejs)
version: (1.0.0)
description: Backend de la App de prueba React Native
entry point: (index.js) server.js
test command:
git repository:
keywords: ReactNative NodeJs
author: Marco Leon Mora
license: (ISC) MIT
About to write to C:\Users\mlmor\OneDrive\Documents\SENA\SENA_CEET\ADSO\moviles\NodeJs\package.json:
{
  "name": "nodejs",
  "version": "1.0.0",
  "description": "Backend de la App de prueba React Native",
  "main": "server.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [
    "ReactNative",
    "NodeJs"
  ],
  "author": "Marco Leon Mora",
  "license": "MIT"
}

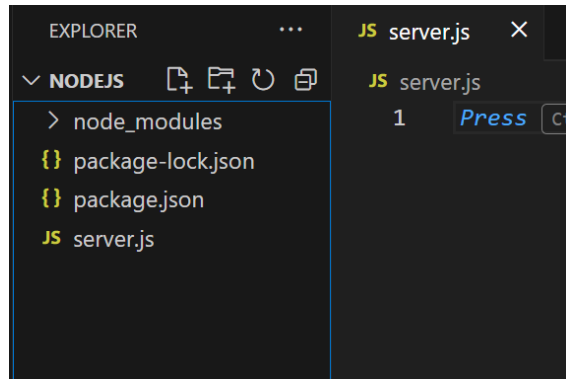
Is this OK? (yes) yes
mlmor@MarcoLeon MINGW64 ~/OneDrive/Documents/SENA/SENA_CEET/ADSO/moviles/NodeJs
$ |
```

Se creará un archivo “package.json”.

Abrir VSC en la carpeta del servidor, en el ejemplo es “NodeJs”

Para crear el nuevo servidor, ejecutar en la consola el comando “npm i express”

En la misma carpeta, crear un archivo “server.js”



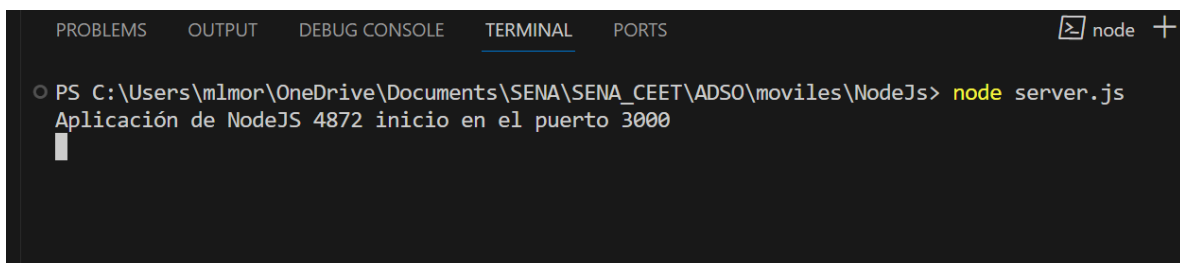
En VSC, abrir una terminal y ejecutar el comando “npm i http”
En el archivo “server.js”:

```
const express = require('express');
const app = express();
const http = require('http');
const server = http.createServer(app);

const port = process.env.PORT || 3000;

app.set('port', port);
//direccion ip V4 de la maquina, consultar con ipconfig
server.listen(3000, '192.168.1.96' || 'localhost', function() {
  console.log('Aplicación de NodeJS ' + process.pid + 'inicio en el puerto ' + port);
})
```

Para lanzar el servidor, en una terminal de VSC ejecutar “node server.js”.



Si al ejecutar el servidor marca error, verifique la ip.

Prueba de Rutas del servidor

Detener el servidor (Ctrl + c), ejecutar “npm i cors” y posteriormente “npm i morgan” este último para realizar depuración. En “package.json” se reflejarán las nuevas dependencias:

```
JS server.js {} package.json X
{} package.json > ...
10     "ReactNative",
11     "NodeJs"
12   ],
13   "author": "Marco Leon Mora",
14   "license": "MIT",
15   "dependencies": {
16     "cors": "^2.8.5",
17     "express": "^4.18.2",
18     "http": "^0.0.1-security",
19     "morgan": "^1.10.0"
20   }
21 }
```

Modificar “server.js” así:

```
const express = require('express');
const app = express();
const http = require('http');
const server = http.createServer(app);
const logger = require('morgan');
const cors = require('cors');

const port = process.env.PORT || 3000;
app.use(logger('dev'));
app.use(express.json());
app.use(express.urlencoded({ extended: true }));
app.use(cors());
app.disable('x-powered-by');

app.set('port', port);

//direccion ip V4 de la maquina, consultar con ipconfig
server.listen(3000, '192.168.1.96' || 'localhost', function() {
  console.log('Aplicación de NodeJS ' + process.pid + ' inicio en el puerto ' + port);
});

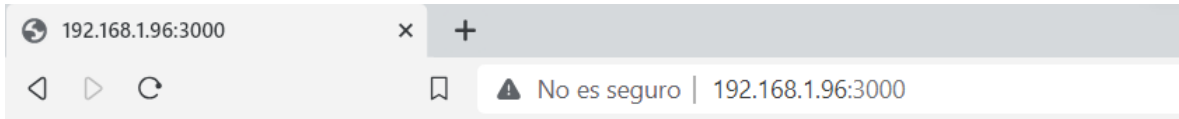
app.get('/', (req, res) => {
  res.send('Ruta raiz del Backend');
});

//Error handler
```

```
app.use((err, req, res, next) => {  
  console.log(err);  
  res.status(err.status || 500).send(err.stack);  
});
```

Lanzar el servidor en la terminal con “node server.js”.

En un navegador, digite la dirección del servidor y obtendrá la respuesta definida.
Opcionalmente puede utilizar Postman:



Rura raíz del Backend

CONECTANDO A LA BASE DE DATOS (MySQL)

Debe tener instalado MySQL, ejecutar el Workbench y crear un nuevo esquema (Base de Datos), en este ejemplo se denominará “nodejs_base1”.

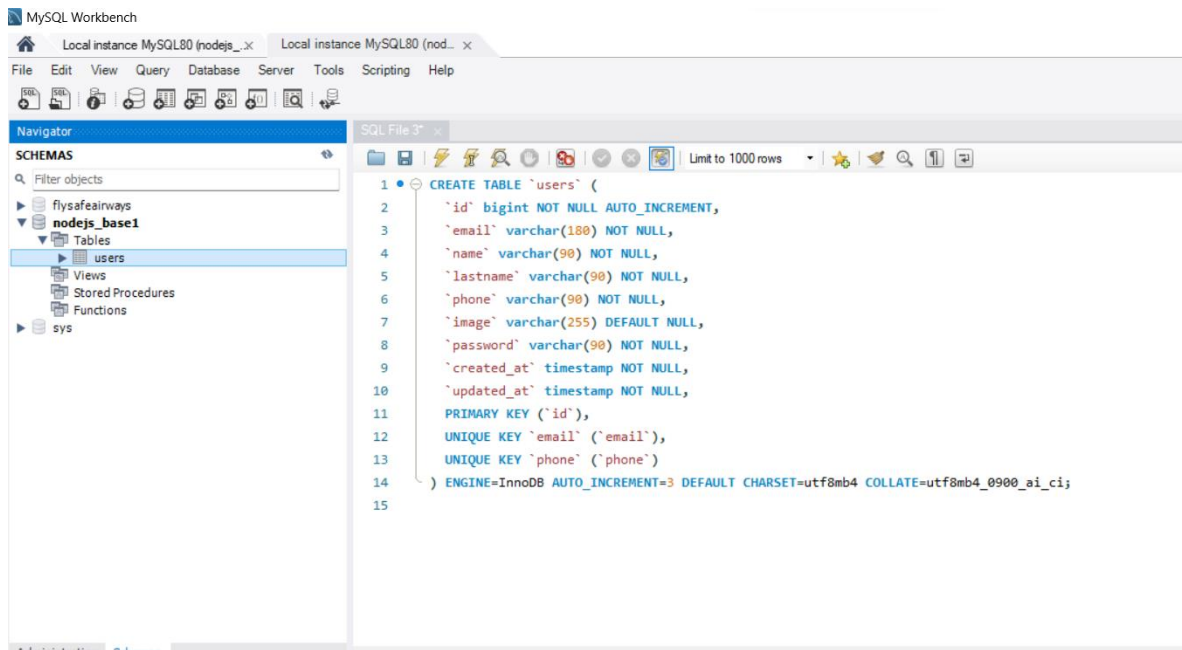
En la carpeta del servidor crear una carpeta “db” y dentro, un archivo “db.sql”.

Crear los scripts para creación de tablas, en primer lugar, la tabla “users”:

```
USE nodejs_base1;

CREATE TABLE users(
  id BIGINT PRIMARY KEY AUTO_INCREMENT,
  email VARCHAR(180) NOT NULL UNIQUE,
  name VARCHAR(90) NOT NULL,
  lastname VARCHAR(90) NOT NULL,
  phone VARCHAR(90) NOT NULL UNIQUE,
  image VARCHAR(255) NULL,
  password VARCHAR(90) NOT NULL,
  created_at TIMESTAMP(0) NOT NULL,
  updated_at TIMESTAMP(0) NOT NULL
);
```

En MYSQL Workbench cree la base de datos y en ella la tabla users.



Realizando la conexión desde Node.js

En el terminal de VSC ejecutar “npm i mysql” para instalar la dependencia a la BD.

Crear una carpeta “config” en la raíz y dentro un archivo “config.js”:

```
const mysql = require('mysql');
const db = mysql.createConnection({
  host: 'localhost',
  user: 'root',
  password: 'su_password_de_la_BD',
  database: 'nodejs_base1'
});

db.connect(function(err) {
  if (err) throw err;
  console.log('Base de datos conectada');
});

module.exports = db;
```

Crear una carpeta “models” en la raíz y en ella un archivo “user.js”:

```
const mysql = require('../config/config');
const User = {};
User.create = (user, result) => {
  const sql = `
    INSERT INTO users(
      email,
      name,
      lastname,
      phone,
      image,
      password,
      created_at,
      updated_at
    )
    VALUES (?, ?, ?, ?, ?, ?, ?, ?)`
  ;

  db.query(
    sql,
    [
      user.email,
      user.name,
      user.lastname,
      user.phone,
```

```

        user.image,
        user.password,
        new Date(),
        new Date()
    ],
    (err, res) => {
        if (err) {
            console.log('error: ', err);
            result(err, null);
        }
        else {
            console.log('Id del nuevo Usuario: ', res.insertId);
            result(null, res.insertId);
        }
    }
)
};

module.exports = User;

```

Crear una carpeta “controllers” en la raíz y en ella un archivo “usersController.js”:

```

const User = require('../models/user');

module.exports = {
    register(req, res) {
        const user = req.body; //Datos del cliente
        User.create(user, (err, data) => {
            if (err) {
                return res.status(501).json({
                    success: false,
                    message: 'Error al crear el usuario',
                    error: err
                });
            }
            return res.status(201).json({
                success: true,
                message: 'Creado el usuario',
                data: data //Id del usuario creado
            });
        });
    }
};

```

Crear una carpeta “routes” en la raíz y en ella un archivo “userRoutes.js”:

```
const usersController = require('../controllers/usersController');
module.exports = (app) => {
  app.post('/api/users/create', usersController.register);
}
```

En el archivo “server.js”, importar las rutas:

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

/**
 * Importar rutas
 */
const users = require('./routes/usersRoutes');

const port = process.env.PORT || 3000;
```

y llamar las rutas:

```
app.set('port', port);

/**
 * LLamando las rutas
 */
usersRoutes(app);
```

En la consola ejecute el servidor con “node server.js”,

Si muestra el error:

“sqlMessage: 'Client does not support authentication protocol requested by server; consider upgrading MySQL client',”

En MySQL Workbench ejecute los comandos:

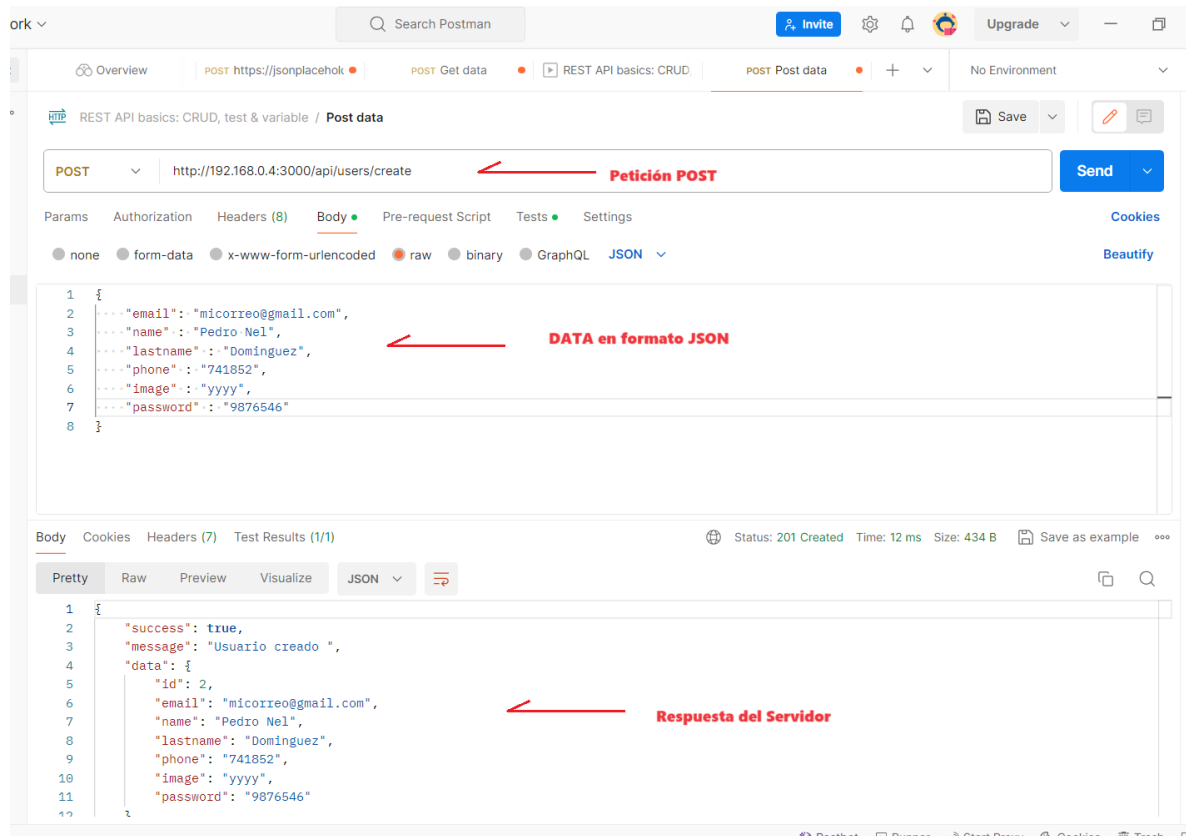
- “ALTER USER 'username'@'localhost' IDENTIFIED WITH mysql_native_password BY 'password';”c (debe reemplazar ‘username’ por su usuario y ‘password’ por su contraseña de la BD).
- “FLUSH PRIVILEGES;”

Al ejecutar de nuevo el servidor, aparecerá el mensaje indicando que todo va bien:

```
Node.js v21.5.0
PS C:\Users\mlmor\OneDrive\Documents\SENA\SENA_CEET\ADS0\moviles\NodeJs> node server.js
Aplicación de NodeJS 5672 inicio en el puerto 3000
Base de datos conectada
```

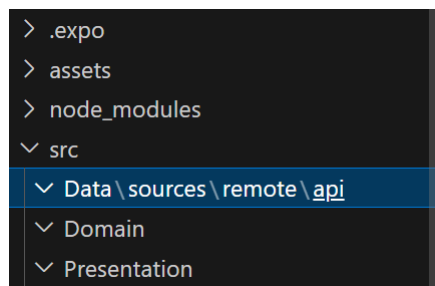
Creando un usuario

Lance el servidor y puede usar Postman para realizar una petición POST y crear un registro de usuario.



Creando usuarios desde la App

En la carpeta “Data” crear otra carpeta “sources” y dentro otra carpeta “remote” y otra “api”:



Desde la terminal, instalar Axios con el comando “npm i axios”

Dentro de “api” un archivo “ApiDelivery.tsx”:

```
import axios from 'axios';

const ApiDelivery = axios.create({
  baseURL: 'https://192.168.1.14:3000/api',
  headers: {
    'Content-Type': 'application/json'
  }
});
export {ApiDelivery};
```

En el archivo “ViewModel.tsx” importar ApiDelivery:

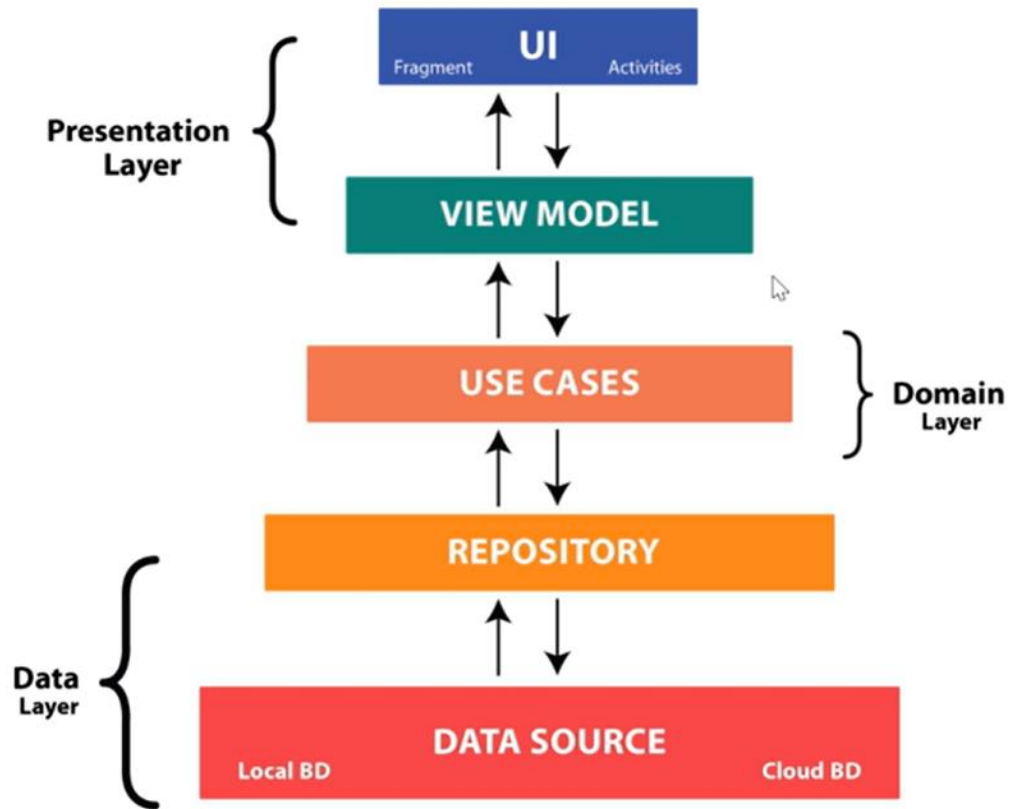
```
import React, { useState } from 'react'
import { ApiDelivery } from '../../Data/sources/remote/api/ApiDelivery';
```

Y modifique register, así:

```
const register = async () => {
  try {
    const response = await ApiDelivery.post('/users/create', values);
    console.log('RESPONSE: ' + JSON.stringify(response));
  } catch (error) {
    console.log('ERROR: ' + error);
  }
}
```

Lance el servidor, y en el emulador o el dispositivo físico, cree el nuevo usuario.

AJUSTANDO EL PROYECTO A LA ARQUITECTURA MVVM



Google define la arquitectura MVVM en las capas de la figura anterior, se crean entidades, repositorios, etc., según las reglas de desarrollo definidas por Google.

Ajustar el dominio

En la carpeta Domain cree dos carpetas 'entities' y 'useCases'. Dentro de 'entity' cree un archivo 'User.tsx':

```
export interface User {  
  id?: string;  
  name: string;  
  lastname: string;  
  phone: string;  
  email: string;  
  password: string;  
  confirmPassword: string;  
}
```

En la carpeta 'Data/sources/remote' crear una carpeta 'models' y en ella un archivo denominado 'ResponseApiDelivery.tsx':

```
export interface ResponseApiDelivery {
  success: boolean;
  message: string;
  data: any;
}
```

En a carpeta 'Domain' crear una carpeta 'repositories' y en ella un archivo 'AuthRepository.tsx':

```
import { User } from "../entities/User";

export interface AuthRepository {
  register(user: User): Promise<any>;
}
```

En la carpeta 'Data' crear otra carpeta 'repositories' y en ella otro archivo con el mismo nombre 'AuthRepository.tsx':

```
import { User } from "../../Domain/entities/User";
import { AuthRepository } from "../../Domain/repositories/AuthRepository";
import { ApiDelivery } from "../sources/remote/api/ApiDelivery";
import { ResponseApiDelivery } from
"../sources/remote/models/ResponseApiDelivery";

export class AuthRepositoryImpl implements AuthRepository {
  async register(user: User) {
    try {
      const response = await
ApiDelivery.post<ResponseApiDelivery>('/users/create', user);

      return Promise.resolve({ error: undefined, result: response.data
});
    } catch (error) {
      let e = (error as Error).message;
      console.log('error' + e);
      return Promise.resolve({ success: false, error: e, result:
undefined });
    }
  }
}
```


En la carpeta 'useCases' crear una carpeta 'auth' y en ella el archivo 'RegisterAuth.tsx':

```
import {AuthRepositoryImpl} from "../../Data/repositories/AuthRepository";
import { User } from "../../entities/User";

const { register} = new AuthRepositoryImpl();

export const RegisterAuthUseCase = async (user: User) =>{
    return await register(user);
}
```

Modificar el archivo 'viewModel.tsx' de la carpeta 'src/Presentation/views/register':

```
import React, { useState } from "react"
import { ApiDelivery } from "../../Data/sources/remote/api/ApiDelivery";
import { RegisterAuthUseCase } from
"../../Domain/useCases/auth/RegisterAuth";

const RegisterViewModel = () => {
    const [values, setValues] = useState({
        name: '',
        lastname: '',
        email: '',
        phone: '',
        password: '',
        confirmPassword: ''
    });

    const onChange = (property: string, value: any) => {
        setValues({ ...values, [property]: value });
    }

    const register = async() => {
        const { result, error } = await RegisterAuthUseCase(values);
        console.log('result' + JSON.stringify(result));
        console.log('error' + error);
    }

    return {
        ...values,
        onChange,
        register
    }
}
```

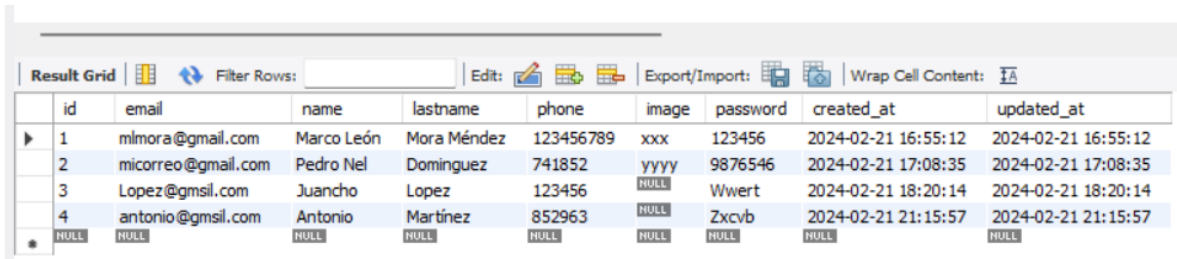
```
export default RegisterViewModel;
```

Después de realizadas las modificaciones anteriores recargue la aplicación y envíe los datos de un nuevo registro de usuario.

En la consola del servidor se muestra el resultado:

```
App Node.js 15808 ejecutando en 192.168.0.4:3000
Conectado a la base de datos MySQL
Usuario creado: {
  id: 4,
  name: 'Antonio',
  lastname: 'Martínez ',
  email: 'antonio@gmsil.com',
  phone: '852963',
  password: 'Zxcvb',
  confirmPassword: 'Zxcvb'
}
POST /api/users/create 201 47.211 ms - 191
```

que también podrá ser comprobado en la consola de MySQL Workbench:



The screenshot shows the MySQL Workbench interface with the 'Result Grid' tab selected. It displays a table with 10 columns: id, email, name, lastname, phone, image, password, created_at, and updated_at. There are 5 rows of data, with the last row (id 4) representing the newly created user 'Antonio Martínez'.

	id	email	name	lastname	phone	image	password	created_at	updated_at
▶	1	mimora@gmail.com	Marco León	Mora Méndez	123456789	xxx	123456	2024-02-21 16:55:12	2024-02-21 16:55:12
	2	micorreo@gmail.com	Pedro Nel	Dominguez	741852	yyyy	9876546	2024-02-21 17:08:35	2024-02-21 17:08:35
	3	Lopez@gmsil.com	Juancho	Lopez	123456	NULL	Wwert	2024-02-21 18:20:14	2024-02-21 18:20:14
	4	antonio@gmsil.com	Antonio	Martínez	852963	NULL	Zxcvb	2024-02-21 21:15:57	2024-02-21 21:15:57
*	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Ajustes a las pantallas para mejorar la presentación

En el archivo 'Register.tsx' (en views/register), agregar un contenedor scroll view para que se visualicen los inputs al momento de digitar.

1. Modifique el import para incluir el ScrollView, elimine lo que no se use.

```
src > Presentation > views > register > Register.tsx > RegisterScreen
1  import React from 'react'
2  import { View, Text, Image, ScrollView } from 'react-native'
3  import { RoundedButton } from '../components/RoundedButton';
4  import useViewModel from './ViewModel';
```

2. Inserte la etiqueta <ScrollView> justo después del view del form:

```

27     <View style={styles.form}>
28       <ScrollView>
29         <Text style={styles.formText}>REGISTRARSE</Text>
30
31         <CustomTextInput
32 > image={require('../../../../assets/user.png')} ...
37       onChangeText={onChange}
38     />

```

3. No olvide la etiqueta de cierre:

```

87 > <View style={{ marginTop: 30 }}> ...
89   </View>
90   </ScrollView>
91   </View>
92
93 </View>
94 );
95 }

```

Para visualizar mejor el botón, en el archivo ‘Styles.tsx’ de register, modificar un poco la altura del form:

```

16   form: {
17     width: '100%',
18     height: '75%',
19     backgroundColor: 'white',
20     position: 'absolute',
21     bottom: 0,
22     borderTopLeftRadius: 40,
23     borderTopRightRadius: 40,
24     padding: 30,
25   },

```

Ajustar la interfase

Modificar el archivo ResponseApiDelivery.tsx en Data/sources/remote/models:

```
src > Data > sources > remote > models > ResponseApiDelivery.tsx > ResponseApiDelivery
1  export interface ResponseApiDelivery {
2      success: boolean;
3      message: string;
4      data?: any;
5      error?: any;
6  }
```

Modificar AuthRepository.tsx en Domain/Repositories:

```
src > Domain > repositories > AuthRepository.tsx > ...
1  import { ResponseApiDelivery } from "../../Data/sources/remote/models/ResponseApiDelivery";
2  import { User } from "../entities/User";
3
4  export interface AuthRepository {
5      register(user: User): Promise<ResponseApiDelivery>;
6  }
7  }
```

Modificar AuthRepository.tsx en Data/repositories:

```
import { AxiosError } from "axios";
import { User } from "../../Domain/entities/User";
import { AuthRepository } from "../../Domain/repositories/AuthRepository";
import { ApiDelivery } from "../sources/remote/api/ApiDelivery";
import { ResponseApiDelivery } from "../sources/remote/models/ResponseApiDelivery";

export class AuthRepositoryImpl implements AuthRepository {
    async register(user: User): Promise<ResponseApiDelivery> {
        try {
            const response = await
ApiDelivery.post<ResponseApiDelivery>('/users/create', user);
            return Promise.resolve(response.data);
        } catch (error) {
            let e = (error as AxiosError);
            console.log('error' + JSON.stringify(e.response?.data));
            const apiError:ResponseApiDelivery =
JSON.parse(JSON.stringify(e.response?.data));
            return Promise.resolve(apiError);
        }
    }
}
```

Modificar en viewModel.tsx de Presentation/views/register:

```

18
19     const register = async() => {
20         const response = await RegisterAuthUseCase(values);
21         console.log('Result' + JSON.stringify(response));
22     }
23
24     return {
25         ...values,
26         onChange,

```

Cargue de nuevo la aplicación e inserte un nuevo registro para probar los cambios realizados.

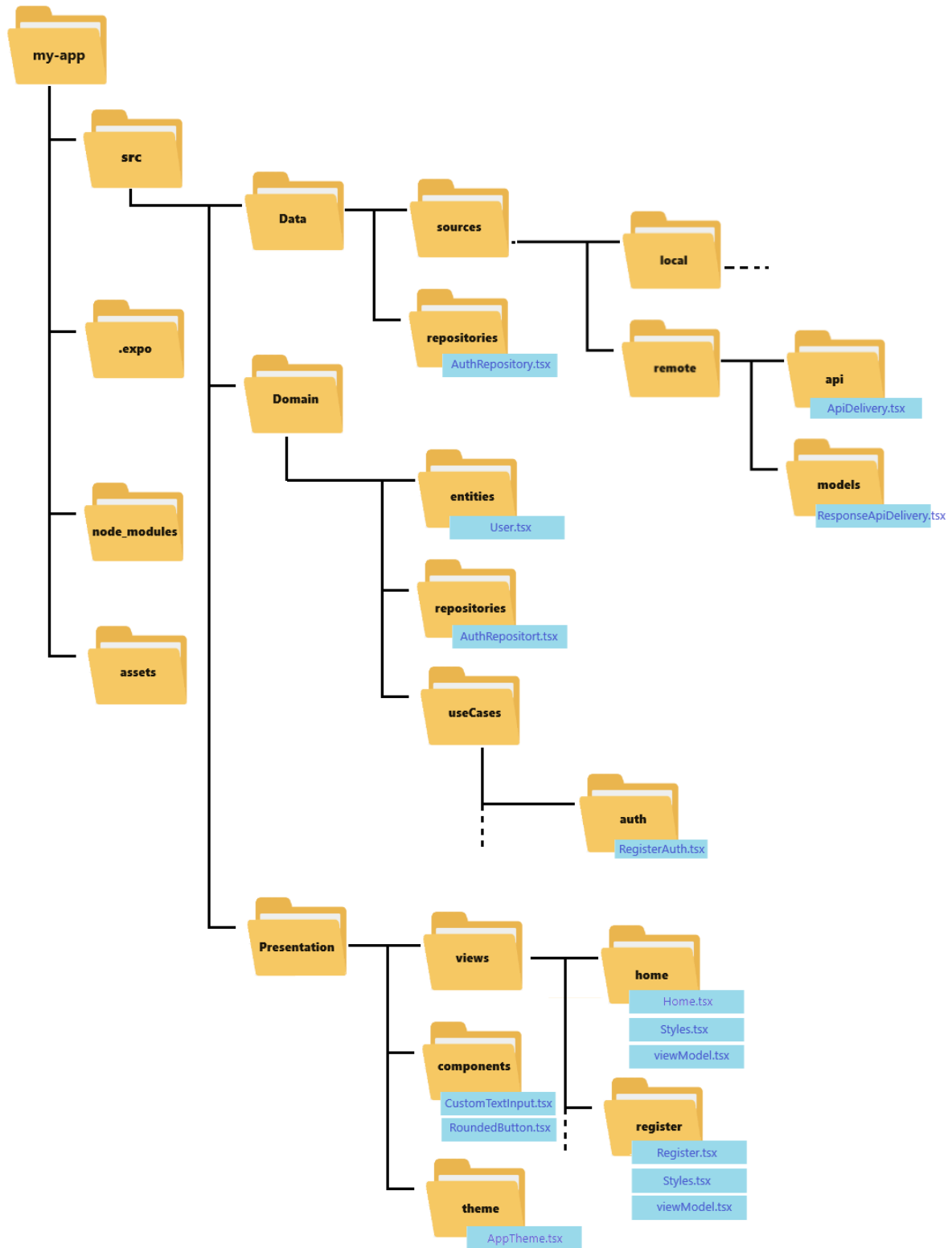
Estructura de carpetas del proyecto

La estructura de archivos en un proyecto de React Native puede variar dependiendo de las necesidades del proyecto, pero aquí hay una descripción general de una estructura de archivos común.

La arquitectura limpia (*Clean Architecture*) y el patrón de diseño Model-View-ViewModel (MVVM) son dos conceptos que se pueden usar juntos para estructurar una aplicación de manera eficiente y mantenible. En la figura siguiente se muestra la estructura general de cómo podrían organizarse los archivos en un proyecto que utiliza *Clean Architecture* y el patrón MVVM.

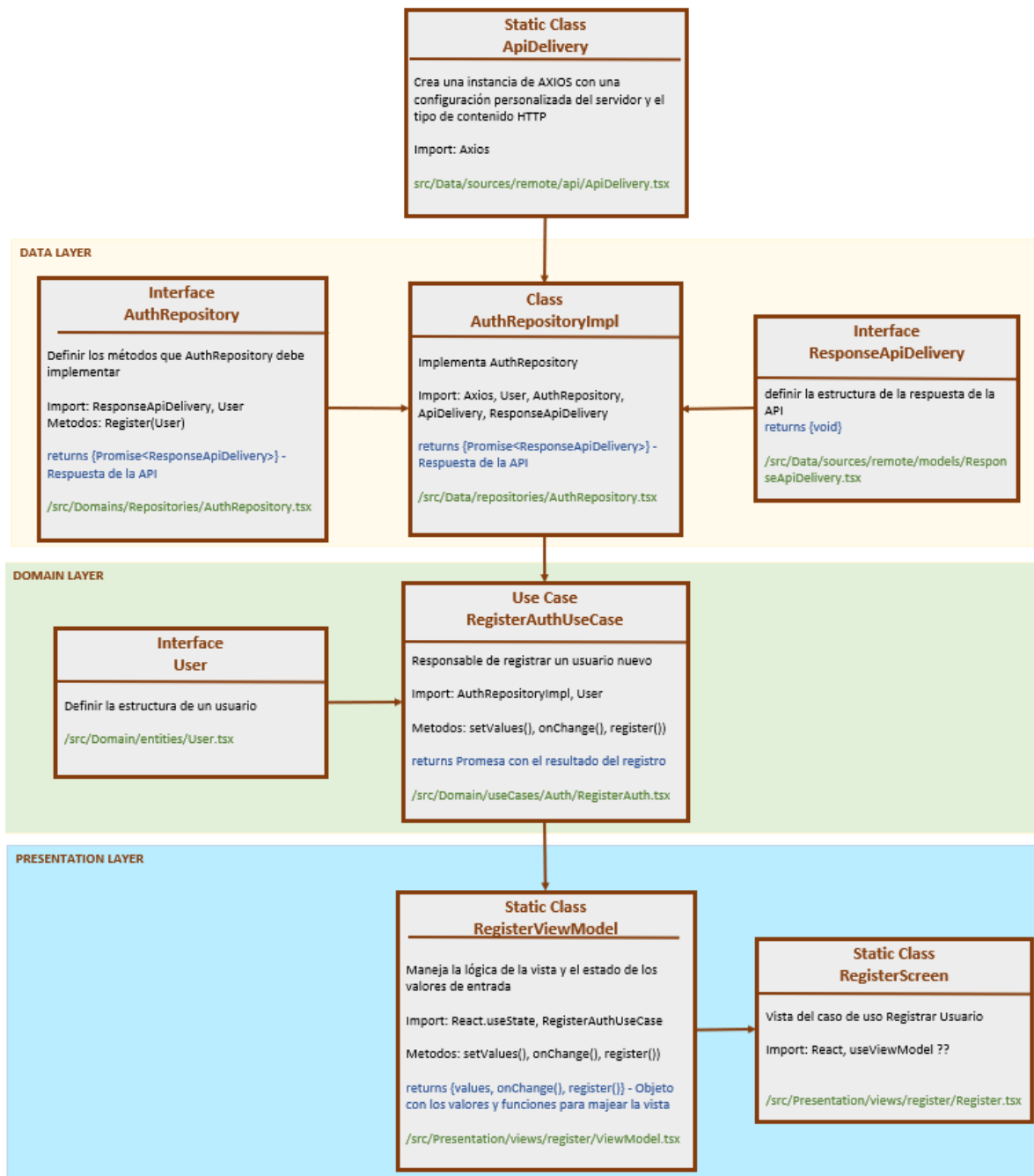
Para una explicación de Clean Architecture ver:

<https://www2.deloitte.com/es/es/pages/technology/articles/clean-architecture.html>



Organización de los módulos de código

La siguiente figura presenta la relación entre los diferentes objetos, no es estrictamente un diagrama de clases, sino que pretende dar información necesaria para comprender la funcionalidad de cada módulo, el archivo en el que se codifica y la capa a la que pertenece.



Validaciones en los formularios

Antes de agregar validaciones al formulario, agregar un componente del tipo “scrollView” para mejorar la presentación de la pantalla cuando se introducen datos. En el archivo `src/Presentation/views/register/Register.tsx`, agregar una etiqueta `<ScrollView>` justo antes del título “REGISTRATE”, todo el contenido de elementos estará dentro de esta etiqueta. Modifique el ‘import’ así:

```
/**
 * Register Screen
 * Is the view of the register screen,
 * it is the first file that is going to be rendered
 * when the user wants to register in the app.
 */
import React from 'react'
import { View, Text, Image, ScrollView } from 'react-native'
```

La etiqueta de apertura `<ScrollView>` así:

```
<View style={styles.form}>
  <ScrollView>
    <Text style={styles.formText}>REGISTRARSE</Text>

    <CustomTextInput
```

Y el cierre, justo antes del cierre del `<View>` del formulario:

```
    <View style={{ marginTop: 30 }}>
      <RoundedButton text='GUARDAR' onPress={() => register()} />
    </View>
  </ScrollView>
</View>
</View>
);
}
```

En el archivo de estilos del registro, modifique la propiedad ‘height’ del ‘form’, por ejemplo, a 75%, hasta que se visualice correctamente el botón.

Cambios al tipo de respuesta y validaciones

Puesto que, en el back-end ya se conoce el tipo de respuesta retornada (Archivo “Controllers/userController.js”), hacer los cambios necesarios, así:

En “../remote/models/ResponseApiDelivery.tsx”, modificar:>

```
export interface ResponseApiDelivery {
  success: boolean;
  message: string;
  data?: any;
  error?: any;
}
```

En “../Domain/repositories/AuthRepository.tsx”:

```
import { ResponseApiDelivery } from
"../../Data/sources/remote/models/ResponseApiDelivery";
import { User } from "../entities/User";

export interface AuthRepository {
  register(user: User): Promise<ResponseApiDelivery>;
}
```

En “../Data/repositories/AuthRepository.tsx”:

```
import { AxiosError } from "axios";
import { User } from "../../Domain/entities/User";
import { AuthRepository } from "../../Domain/repositories/AuthRepository";
import { ApiDelivery } from "../sources/remote/api/ApiDelivery";
import { ResponseApiDelivery } from
"../sources/remote/models/ResponseApiDelivery";

export class AuthRepositoryImpl implements AuthRepository {

  async register(user: User): Promise<ResponseApiDelivery> {
    try {
      const response = await
ApiDelivery.post<ResponseApiDelivery>('/users/create', user);
      return Promise.resolve(response.data);
    } catch (error) {
```

```

        let e = (error as AxiosError);
        console.log('error' + JSON.stringify(e.response?.data));
        const apiError:ResponseApiDelivery =
JSON.parse(JSON.stringify(e.response?.data));
        return Promise.resolve(apiError);
    }
}
}

```

En “../Presentation/views/register/ViewModel.tsx”:

```

const RegisterViewModel = () => {
    const [values, setValues] = useState({
        name: '',
        lastname: '',
        email: '',
        phone: '',
        password: '',
        confirmPassword: ''
    });

    const onChange = (property: string, value: any) => {
        setValues({ ...values, [property]: value });
    }

    const register = async() => {
        const response = await RegisterAuthUseCase(values);
        console.log('Result' + JSON.stringify(response));
    }

    return {
        ...values,
        onChange,
        register
    }
}

export default RegisterViewModel;

```

Probar insertando un nuevo Usuario y otro con el mis o correo, para apreciar los mensajes en los dos casos.

Enseguida implementar las validaciones al formulario de registro:
 Modificar el archivo “../views/register/Register.tsx”, así:

```

import React, { useEffect } from 'react'
import { View, Text, Image, ScrollView, ToastAndroid } from 'react-native'
import { RoundedButton } from '../../components/RoundedButton';
import useViewModel from './ViewModel';
import { CustomTextInput } from '../../components/CustomTextInput';
import styles from './Styles';

export const RegisterScreen = () => {

  const { name, lastname, email, phone, password, confirmPassword,
  errorMessage, onChange, register } = useViewModel();

  //Para saber si la variable ya tiene establecido un valor
  useEffect(() => {
    if (errorMessage !== '')
      ToastAndroid.show(errorMessage, ToastAndroid.LONG)
  }, [errorMessage]);

  return (
    <View style={styles.container}>
      <Image
        source={require('../../../../assets/chef.jpg')}
        style={styles.imageBackground}
      />
      <View style={styles.logoContainer}>
        <Image
          source={require('../../../../assets/user_image.png')}
          style={styles.logoImage}
        />
        <Text style={styles.logoText}>SELECCIONA UNA IMAGEN</Text>
      </View>

      <View style={styles.form}>
        <ScrollView>
          <Text style={styles.formText}>REGISTRARSE</Text>

          <CustomTextInput
            image={require('../../../../assets/user.png')}
            placeholder='Nombres'
            value={name}
            keyboardType='default'
            property='name'
            onChangeText={onChange}
          />
        </ScrollView>
      </View>
    </View>
  )
}

```

```
<CustomTextInput
  image={require('../assets/my_user.png')}
  placeholder='Apellidos'
  value={lastname}
  keyboardType='default'
  property='lastname'
  onChangeText={onChange}
/>

<CustomTextInput
  image={require('../assets/email.png')}
  placeholder='Correo electrónico'
  value={email}
  keyboardType='email-address'
  property='email'
  onChangeText={onChange}
/>

<CustomTextInput
  image={require('../assets/phone.png')}
  placeholder='Teléfono'
  value={phone}
  keyboardType='numeric'
  property='phone'
  onChangeText={onChange}
/>

<CustomTextInput
  image={require('../assets/password.png')}
  placeholder='Contraseña'
  value={password}
  keyboardType='default'
  secureTextEntry={true}
  property='password'
  onChangeText={onChange}
/>

<CustomTextInput
  image={require('../assets/confirm_password.png')}
  placeholder='Confirmar contraseña'
  value={confirmPassword}
  keyboardType='default'
  secureTextEntry={true}
  property='confirmPassword'
  onChangeText={onChange}
```

```

        />

        <View style={{ marginTop: 30 }}>
            <RoundedButton text='GUARDAR' onPress={() => register()} />
        </View>
    </ScrollView>
</View>
</View>
);
}

```

Y el archivo “ViewModel.tsx”:

```

import React, { useState } from "react"
import { RegisterAuthUseCase } from
"../../Domain/useCases/auth/RegisterAuth";

const RegisterViewModel = () => {
    const [errorMessage, setErrorMessage] = useState('');
    const [values, setValues] = useState({
        name: '',
        lastname: '',
        email: '',
        phone: '',
        password: '',
        confirmPassword: ''
    });

    const onChange = (property: string, value: any) => {
        setValues({ ...values, [property]: value });
    }

    const register = async () => {
        if (!isValidForm()) {
            const response = await RegisterAuthUseCase(values);
            console.log('Result' + JSON.stringify(response));
        }
    }

    const isValidForm = (): boolean => {
        if (values.name === '') {
            setErrorMessage('El nombre es requerido');
            return false;
        }
        if (values.lastname === '') {

```

```

        setErrorMessage('El apellido es requerido');
        return false;
    }
    if (values.email === '') {
        setErrorMessage('El correo es requerido');
        return false;
    }
    if (values.phone === '') {
        setErrorMessage('El teléfono es requerido');
        return false;
    }
    if (values.password === '') {
        setErrorMessage('La contraseña es requerida');
        return false;
    }
    if (values.confirmPassword === '') {
        setErrorMessage('La confirmación de contraseña es requerida');
        return false;
    }
    if (values.password !== values.confirmPassword) {
        setErrorMessage('Las contraseñas no coinciden');
        return false;
    }
    return true;
}

return {
    ...values,
    onChange,
    register,
    errorMessage
}
}

export default RegisterViewModel;

```

Encriptar el password

Bajar el servidor con 'CTRL+C'.
 En el servidor, Instalar el paquete npm I bcryptjs.
 Modificar el archivo "../models/user.js":

```
const db = require('../config/config');
```

```

const bcrypt = require('bcryptjs');

const User = {};
User.create = async (user, result) => {
  const hash = await bcrypt.hash(user.password, 10);
  const sql =
    `INSERT INTO users (
      email,
      name,
      lastname,
      phone,
      image,
      password,
      created_at,
      updated_at
    )
    VALUES (?, ?, ?, ?, ?, ?, ?, ?)`;

  db.query(sql,
    [
      user.email,
      user.name,
      user.lastname,
      user.phone,
      user.image,
      hash,
      new Date(),
      new Date()
    ], (err, res) => {
      if (err) {
        console.log('Error al crear el usuario: ', err);
        result(err, null);
      }
      else {
        console.log('Usuario creado: ', { id: res.insertId, ...user
      });
      result(null, { id: res.insertId, ...user });
    }
  );
}

module.exports = User;

```

Cree un nuevo Usuario y verifique que la contraseña ahora está encriptada:

[illegible]

JSON WEB TOKEN

Configuración del back-end para generar el token

Eliminar, en la BD los usuarios que no tengan el password encriptado.

En la página jwt.io se encuentra la documentación del paquete a utilizar.

En el servidor, instalar:

- npm i Passport
- npm i Passport-jwt
- npm jsonwebtoken

En Google buscar ‘secret key generator’, en alguno de las páginas, generar una clave de 256 bits y copiarla, crear el archivo “/config/keys.js”:

```
module.exports = {  
  secretOrKey: 'BC843E3A8652682481B5E59468627'  
}
```

En “/models/user.js” agregar los métodos ‘findById’ y ‘findByEmail’ justo antes del método ‘create’:

```
const db = require('../config/config');  
const bcrypt = require('bcryptjs');  
  
const User = {};  
  
User.findById = (id, result) => {  
  const sql = `SELECT id, email, name, lastname, image, password FROM  
users WHERE id = ?`;  
  
  db.query(sql,  
    [id], (err, user) => {  
      if (err) {  
        console.log('Error al consultar: ', err);  
        result(err, null);  
      }  
      else {  
        console.log('Usuario consultado: ', user[0] );  
        result(null, user[0]);  
      }  
    }  
  );  
}  
  
User.findByEmail = (email, result) => {
```

```
const sql = `SELECT id, email, name, lastname, image, phone, password
FROM users WHERE email = ?`;
```

```
db.query(
  sql,
  [email],
  (err, user) => {
    if (err) {
      console.log('Error al consultar: ', err);
      result(err, null);
    }
    else {
      console.log('Usuario consultado: ', user[0] );
      result(null, user[0]);
    }
  }
);
}
```

```
User.create = async (user, result) => {
  const hash = await bcrypt.hash(user.password, 10);
  const sql =
    `INSERT INTO users (
      email,
      name,
      lastname,
      phone,
      image,
      password,
      created_at,
      updated_at
    )
  VALUES (?, ?, ?, ?, ?, ?, ?, ?)`;

  db.query(sql,
    [
      user.email,
      user.name,
      user.lastname,
      user.phone,
      user.image,
      hash,
      new Date(),
    ]
  )
}
```

```

        new Date()
    ], (err, res) => {
        if (err) {
            console.log('Error al crear el usuario: ', err);
            result(err, null);
        }
        else {
            console.log('Usuario creado: ', { id: res.insertId, ...user
    });

            result(null, { id: res.insertId, ...user });
        }
    }
    );
}

module.exports = User;

```

Crear el archivo “/config/passport.js”:

```

const JwtStrategy = require('passport-jwt').Strategy;
const ExtractJwt = require('passport-jwt').ExtractJwt;
const Keys = require('./keys');
const User = require('../models/user');

module.exports = (passport) => {
    const opts = {};
    opts.jwtFromRequest = ExtractJwt.fromAuthHeaderWithScheme('jwt');
    opts.secretOrKey = Keys.secretOrKey;

    passport.use(new JwtStrategy(opts, (jwt_payload, done) => {
        User.findById(jwt_payload.id, (err, user) => {
            if (err) {
                return done(err, false);
            }
            if (user) {
                return done(null, user);
            }
            return done(null, false);
        });
    }));
});

```

En el archivo “/controllers/usersController.js” crear el método ‘login’:

```
const User = require('../models/user');
const bcrypt = require('bcryptjs');
const jwt = require('jsonwebtoken');
const keys = require('../config/keys');

module.exports = {
  login(req, res) {
    const email = req.body.email;
    const password = req.body.password;

    User.findByEmail(email, async (err, myUser) => {
      if (err) {
        return res.status(501).json({
          success: false,
          message: 'Error al consultar el usuario',
          error: err
        });
      }

      if (!myUser) { //Cliente sin autorización para realizar la
        petición
        return res.status(401).json({
          success: false,
          message: 'El email no existe en la base de datos'
        });
      }

      const isPasswordValid = await bcrypt.compare(password,
        myUser.password);
      if (isPasswordValid) {
        const token = jwt.sign({ id: myUser.id, email: myUser.email
        }, keys.secretOrKey, {});

        const data = {
          id: myUser.id,
          email: myUser.email,
          name: myUser.name,
          lastname: myUser.lastname,
          image: myUser.image,
          phone: myUser.phone,
          session_token: `JWT ${token}`
        }
      }
    });
  }
}
```

```

    }
    return res.status(201).json({
      success: true,
      message: 'Usuario autenticado ',
      data: data
    });

  }
  else {
    return res.status(401).json({
      success: false,
      message: 'Contraseña incorrecta'
    });
  }
});

},

register(req, res) {
  const user = req.body;
  User.create(user, (err, data) => {
    if (err) {
      return res.status(501).json({
        success: false,
        message: 'Error al crear el usuario',
        error: err
      });
    }

    return res.status(201).json({
      success: true,
      message: 'Usuario creado ',
      data: data //Id del nuevo usuario
    });
  });
}
}
}

```

Modificar el archivo “server.js”, así:

```

const express = require('express');
const passport = require('passport');

const app = express();

```

```

const http = require('http');
const server = http.createServer(app);
const logger = require('morgan');
const cors = require('cors');

/**
 * Importar rutas
 */
const usersRoutes = require('./routes/userRoutes');

const port = process.env.PORT || 3000;

app.use(logger('dev')); // log requests to the console DEBUG
app.use(express.json()); // support json encoded bodies
app.use(express.urlencoded({
  extended: true
})); // support encoded bodies
app.use(cors());
app.use(passport.initialize());
app.use(passport.session());

require('./config/passport')(passport);

app.disable('x-powered-by'); // disable the X-Powered-By header in responses

app.set('port', port);

/**
 * Llamar a las rutas
 */
usersRoutes(app);

// Iniciar el servidor
server.listen(port, '192.168.0.4' || 'localhost', function() {
  console.log('App Node.js ' + process.pid + ' ejecutando en ' +
server.address().address + ':' + server.address().port);
});

/** RUTAS *****/
app.get('/', (req, res) => {
  res.send('Estas en la ruta raiz del backend.');
```

```

});

app.get('/test', (req, res) => {
```

```

    res.send('Estas en la ruta TEST');
  });

//Manejo de errores *****
app.use((err, req, res, next) => {
  console.error(err);
  res.status(err.status || 500).send(err.stack);
});

//en package.json se cambio "passport": "^0.7.0", a "passport": "^0.4.1",

```

En el archivo “routes/userRoutes.js” crear la nueva ruta:

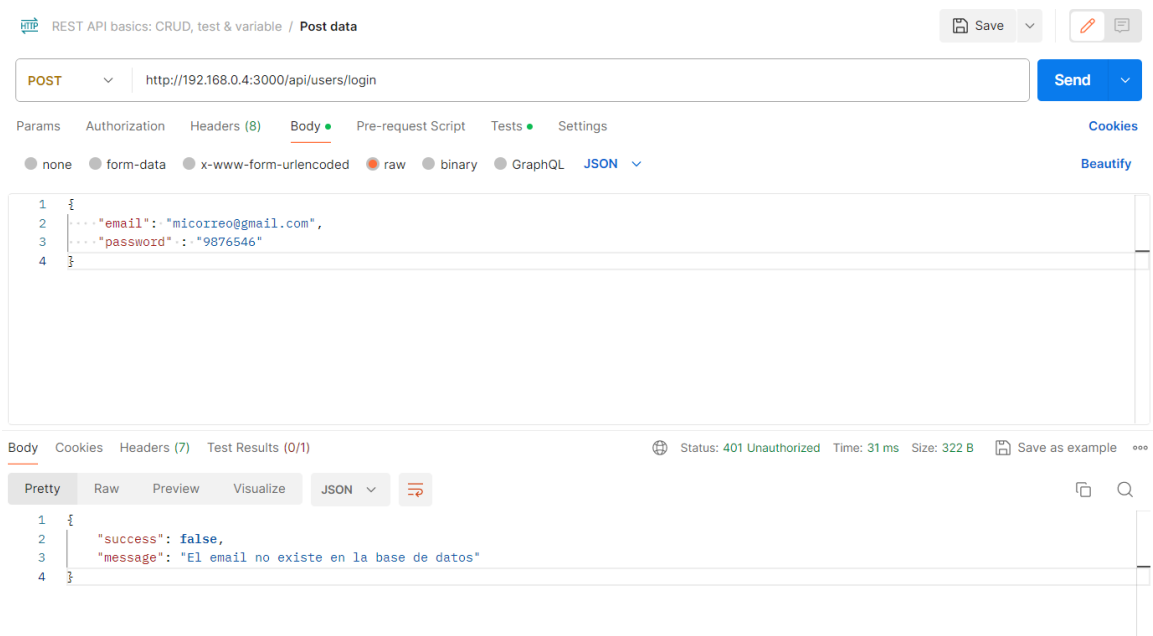
```

const userController = require('../controllers/usersController');

module.exports = (app) => {
  app.post('/api/users/create', userController.register);
  app.post('/api/users/login', userController.login);
}

```

Para probar las modificaciones implementadas, usar Postman:

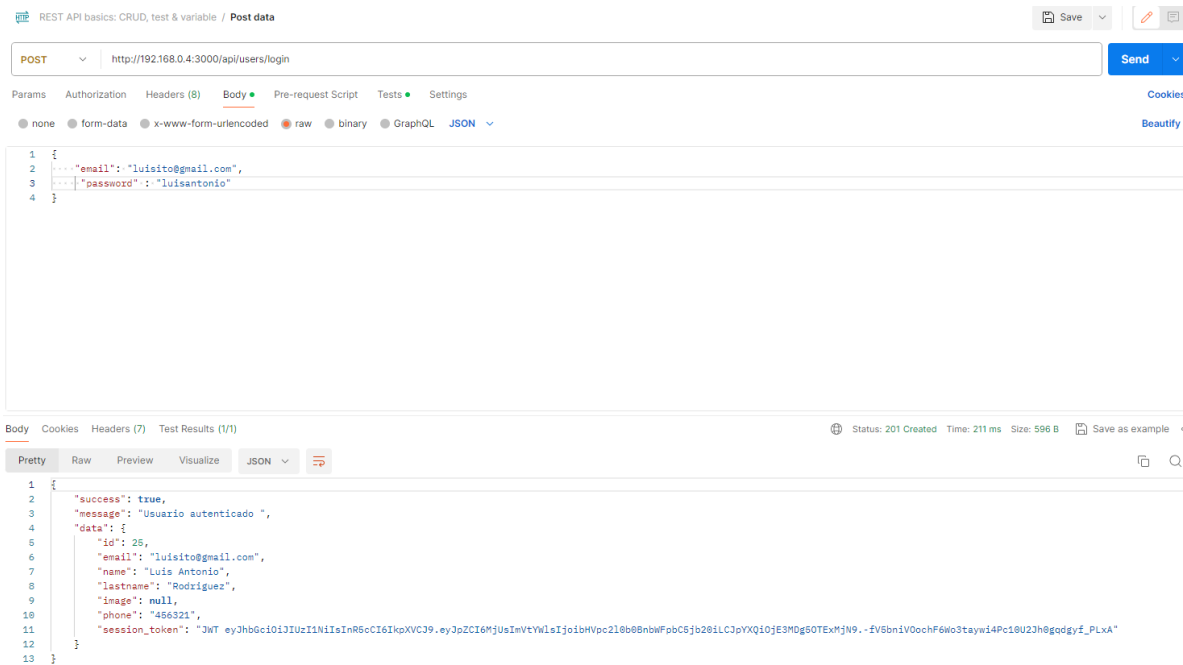


En caso de tener errores, compruebe en la consola del servidor las indicaciones del error. Es probable que deba cambiar la versión de la dependencia “password”, En el archivo “package.js” cambie la versión así:

```
"passport": "^0.4.1",
```

Detenga el servidor, ejecute el comando 'npm i' para realizar el cambio de versión. Pruebe de nuevo.

En el caso de un usuario correcto se tendrá una respuesta como (Observe que la respuesta contiene el token):



Enviando el login desde la App

En “../Domain/repositories/AuthRepository.tsx” agregar un método ‘login’.

```
import { ResponseApiDelivery } from
"../../Data/sources/remote/models/ResponseApiDelivery";
import { User } from "../entities/User";

export interface AuthRepository {
  login(email: string, password: string ): Promise<ResponseApiDelivery>;
  register(user: User): Promise<ResponseApiDelivery>;
}
```

En “Data/repositories/AuthRepository.tsx”, agregar al método ‘login’:


```

import { AxiosError } from "axios";
import { User } from "../../Domain/entities/User";
import { AuthRepository } from "../../Domain/repositories/AuthRepository";
import { ApiDelivery } from "../../sources/remote/api/ApiDelivery";
import { ResponseApiDelivery } from
"../../sources/remote/models/ResponseApiDelivery";

export class AuthRepositoryImpl implements AuthRepository {

    async register(user: User): Promise<ResponseApiDelivery> {
        try {
            const response = await
ApiDelivery.post<ResponseApiDelivery>('/users/create', user);
            return Promise.resolve(response.data);

        } catch (error) {
            let e = (error as AxiosError);
            console.log('error' + JSON.stringify(e.response?.data));
            const apiError:ResponseApiDelivery =
JSON.parse(JSON.stringify(e.response?.data));
            return Promise.resolve(apiError);
        }
    }

    async login(email: string, password: string):
Promise<ResponseApiDelivery> {
        try {
            const response = await
ApiDelivery.post<ResponseApiDelivery>('/users/login', {
                email: email,
                password: password
            });
            return Promise.resolve(response.data);

        } catch (error) {
            let e = (error as AxiosError);
            console.log('error' + JSON.stringify(e.response?.data));
            const apiError:ResponseApiDelivery =
JSON.parse(JSON.stringify(e.response?.data));
            return Promise.resolve(apiError);
        }
    }
}

```

En “Domain/useCases/auth” crear un archivo “Login.Auth.tsx”:

```
import { AuthRepositoryImpl } from
"../../../../../Data/repositories/AuthRepository";

const {login} = new AuthRepositoryImpl();

export const LoginAuthUseCase = async (email: string, password: string) => {
  return await login(email, password);
}
```

En ../Presentation/views/home/ViewModel.tsx” agregar:

```
import React, { useState } from 'react'
import { LoginAuthUseCase } from '../../../../../Domain/useCases/auth/LoginAuth';

const HomeViewModel = () => {
  const [errorMessage, setErrorMessage] = useState('');
  const [values, setValues] = useState(
    {
      email: '',
      password: ''
    }
  );

  const onChange = (property: string, value: any) => {
    setValues({...values, [property]: value});
  }

  const login = async () => {
    if (isValidForm()) {
      const response = await LoginAuthUseCase(values.email,
values.password);
      console.log('Respuesta: ' + JSON.stringify(response));
      if(!response.success) {
        setErrorMessage(response.message);
      }
    }
  };

  const isValidForm = () => {
    if (values.email === '') {
      setErrorMessage('El email es requerido');
      return false;
    }
  };
}
```

```

    }
    if (values.password === '') {
      setErrorMessage('La contraseña es requerida');
      return false;
    }

    return true;
  }

  return {
    ...values,
    onChange,
    login,
    errorMessage
  }
}

export default HomeViewModel;

```

En ../Presentation/views/home/Home.tsx”:

```

import React, { useState, useEffect } from 'react'
import { useNavigation } from '@react-navigation/native';

import { View, Text, Image, TextInput, StyleSheet, ToastAndroid,
TouchableOpacity } from 'react-native'
import { RoundedButton } from
'../../Presentation/components/RoundedButton';
import { StackNavigationProp } from '@react-navigation/stack';
import { RootStackParamList } from '../../../App';
import useViewModel from './ViewModel';
import { CustomTextInput } from '../../../components/CustomTextInput';
import styles from './Styles';

export const HomeScreen = () => {

  const { email, password, errorMessage, onChange, login } =
useViewModel();

  const navigation =
useNavigation<StackNavigationProp<RootStackParamList>>();

```

```

useEffect(() => {
    if (errorMessage !== '') {
        ToastAndroid.show(errorMessage, ToastAndroid.LONG);
    }
}, [errorMessage]);

return (
    <View style={styles.container}>
        <Image
            source={require('../../../../assets/chef.jpg')}
            style={styles.imageBackground}
        />
        <View style={styles.logoContainer}>
            <Image
                source={require('../../../../assets/logo.png')}
                style={styles.logoImage}
            />
            <Text style={styles.logoText}>FOOD APP</Text>
        </View>
        <View style={styles.form}>
            <Text style={styles.formText}>INGRESAR</Text>

            <CustomTextInput
                image={require('../../../../assets/email.png')}
                placeholder='Correo electrónico'
                value={email}
                keyboardType='email-address'
                property='email'
                onChangeText={onChange}
            />

            <CustomTextInput
                image={require('../../../../assets/password.png')}
                placeholder='Contraseña'
                value={password}
                keyboardType='default'
                secureTextEntry={true}
                property='password'
                onChangeText={onChange}
            />

            <View style={{ marginTop: 30 }}>
                <RoundedButton text='ENTRAR' onPress={() => login()} />
            </View>
        </View>
    </View>
);

```

```

        <View style={styles.formRegister}>
            <Text>¿No tienes cuenta?</Text>
            <TouchableOpacity onPress={() =>
navigation.navigate('RegisterScreen')}>
                <Text
style={styles.formRegisterText}>Registrate</Text>
            </TouchableOpacity>
        </View>
    </View>
</View>
);
}

```

Por último, en el archivo “../Domain/entities/user.tsx” agregar:

```

export interface User {
  id?: string;
  name: string;
  lastname: string;
  phone: string;
  email: string;
  password: string;
  confirmPassword: string;
  session_token?: string;
}

```

En aplicación, en el formulario de login ingrese los datos, observe los mensajes de validación y, si los datos son correctos, en la consola de la App se desplegará el token:

```

LOG error{"success":false,"message":"El email no existe en la base de datos"}
LOG Respuesta: {"success":false,"message":"El email no existe en la base de datos"}
LOG Respuesta: {"success":true,"message":"Usuario autenticado ","data":{"id":25,"email":"luisito@gmail.com","name":"Luis Antonio","lastname":"Rodriguez","image":null,"phone":"456321","session_token":"JWT_eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6ImJUsImVtYWlsIjoibHVpc2l0b0BnbWVpbC5jb20iLCJpYXQiOiJlbnR5cCI6IkpXVCJ9.XsPJRsi4zT4qGAqMBjIi4kZL6iFuCxftGG3eJHtsmw"}}

```

Y en la consola del servidor:

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

Usuario consultado: RowDataPacket {
  id: 25,
  email: 'luisito@gmail.com',
  name: 'Luis Antonio',
  lastname: 'Rodriguez',
  image: null,
  phone: '456321',
  password: '$2a$10$J3PLzRBaGqj0q1Jz1IbVc.KFB.VnPUZW3aN/oybW7BR2zn041RA8i'
}
POST /api/users/login 201 244.513 ms - 345
```

Almacenamiento interno la de sesión del usuario

Consultar en la instalación de “React Native Async Storage”. Copiar y ejecutaren la consola de la App, el comando: “npm i @react-native-async-storage/async-storage” y levantar de nuevo la App.

Dentro de la carpeta “../Data/repositories/sources” crear otra carpeta “remote” (al mismo nivel de la carpeta “local”. En ella el archivo “LocalStorage.tsx”:

```
import AsyncStorage from '@react-native-async-storage/async-storage';

export const LocalStorage = () => {
  const save = async (key: string, value: string) => {
    try {
      await AsyncStorage.setItem(key, value);
    } catch (error) {
      console.log('Error en Local Storage: ' + error);
    }
  }

  const getItem = async (key: string) => {
    try {
      const item = await AsyncStorage.getItem(key);
      return item;
    } catch (error) {
      console.log('Error en Local Storage: ' + error);
    }
  }

  const remove = async (key: string) => {
    try {
      await AsyncStorage.removeItem(key);
    } catch (error) {
      console.log('Error en Local Storage: ' + error);
    }
  }
}
```

```

    }
  }

  return {
    save,
    getItem,
    remove
  }
}

```

Crear el archivo `../Domain/repositories/UserLocalRepository.tsx`:

```

import { User } from "../entities/User";

export interface UserLocalRepository {
  save(user: User): void;
  getUser(): Promise<User>;
  remove(): Promise<void>;
}

```

Crear el archivo `../Data/repositories/UserLocalrepository.tsx`:

```

import { User } from "../../Domain/entities/User";
import { UserLocalRepository } from
"../../Domain/repositories/UserLocalRepository";
import { LocalStorage } from "../sources/local/LocalStorage";

export class UserLocalRepositoryImp implements UserLocalRepository{

  async save(user: User): Promise<void> {
    const { save } = LocalStorage();
    await save('user', JSON.stringify(user));
  }

  async getUser(): Promise<User> {
    const { getItem } = LocalStorage();
    const data = await getItem('user');
    const user: User = JSON.parse(data as any);
    return user;
  }
}

```

```

    async remove(): Promise<void> {
        const { remove } = LocalStorage();
        await remove('user');
    }
}

```

Crear una carpeta “../useCases/userLocal” y en ella el archivo “SaveUserLocal.tsx”:

```

import { UserLocalRepositoryImp } from
'../../Data/repositories/UserLocalRepository';
import { User } from '../../entities/User';

const { save } = new UserLocalRepositoryImp();

export const SaveUserLocalUseCase = async(user: User) => {
    return await save(user);
}

```

Crear el archivo “../useCases/userLocal/GetUserLocal.tsx”:

```

import { UserLocalRepositoryImp } from
'../../Data/repositories/UserLocalRepository';

const { getUser } = new UserLocalRepositoryImp();

export const GetUserLocalUseCase = async() => {
    return await getUser();
}

```

En el archivo “../Presentation/views/home/ViewModel.tsx”:

Al inicio, importar los casos de uso:

```

import { SaveUserLocalUseCase } from
'../../Domain/useCases/userLocal/SaveUserLocal';
import { GetUserLocalUseCase } from
'../../Domain/useCases/userLocal/GetUserLocal';}

```

Modificar el ‘login’:

```

const login = async () => {
    if (isValidForm()) {

```



```

        const response = await LoginAuthUseCase(values.email,
values.password);
        console.log('Respuesta: ' + JSON.stringify(response));
        if(!response.success) {
            setErrorMessage(response.message);
        }
        else {
            await SaveUserUseCase(response.data);
        }
    }
};

```

Agregar el Código para el get:

```

const [values, setValues] = useState(
    {
        email: '',
        password: ''
    }
);
//Insertar el código desde acá:
useEffect(() => { //Se ejecuta cuando se instancia el viewModel
    getUserSession();
}, []);

const getUserSession = async () => {
    const user = await GetUserUseCase();
    console.log('Usuario Sesion: ' + JSON.stringify(user));
}

```

Recargar la aplicación, en la consola se tendrá algo así:

```

Android Bundled 5116ms (C:\Users\mImor\OneDrive\Documents\SENA\SENA_CEEI\ADSO\moviles\proyectosRe
ve\app-movil-2\node_modules\expo\AppEntry.js)
LOG Usuario Sesion: null

```

Inicie sesión con un usuario válido, se obtendrá:

```

LOG Respuesta: {"success":true,"message":"Usuario autenticado ","data":{"id":25,"email":"luisito@gmail
.com","name":"Luis Antonio","lastname":"Rodriguez","image":null,"phone":"456321","session_token":"JWT ey
JhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6MjUsImVtYWlsIjoibHVpc2l0b0BnbWVpbC5jb20iLCJpYXQiOiE3MDkwNjY2O
TR9.Fo15eUDZCcY8_xu0hCevxEYV-a52Ct0bvlazjzqXphQ"}}

```

Recargue de nuevo la aplicación (en la consola: r + <ENTER>) observará que ya no aparece el mensaje “Usuario Sesión: NULL”, sino la respuesta obtenida anteriormente, debido a que está activa la sesión de usuario.

CUSTOM HOOKS

Los custom hooks en React son un tipo de función JavaScript que simula el funcionamiento de los hooks en React. Estos custom hooks son muy útiles cuando tenemos una lógica que se repite entre varios componentes. Aquí están las características clave de los custom hooks:

Nombre Empieza por “use”: La primera regla para los custom hooks en React es que su nombre debe comenzar con la palabra “use”. Esta convención sigue el patrón de los hooks originales de React, como useEffect, useState y useRef. Aunque no es obligatorio, seguir esta norma facilita el reconocimiento de un custom hook.

Puede Llamar a Otros Hooks: Lo que realmente hace especial a un custom hook es que puede llamar a otros hooks. React considera como custom hook a una función que, dentro de ella, llama a un hook original o a otro custom hook.

Implementación del Custom Hook

En “../Presentation/” cree una carpeta “hooks” y en ella el archivo “useUserLocal.tsx”:

```
import React, { useEffect, useState } from 'react'
import { GetUserUseCase } from
'../../Domain/useCases/userLocal/GetUserLocal';
import { User } from '../../Domain/entities/User';

export const useUserLocal = () => {

  const [user, setUser] = useState<User>()
  useEffect(() => { //Permite efectos secundarios. obtiene el usuario de la
sesion
    getUserSession();
  }, []);

  const getUserSession = async () => {
    const user = await GetUserLocalUseCase();
    setUser(user);
  }

  return {
    user,
    getUserSession
  }
}
```

En “../home/ViewModel.tsx”:

```
import React, { useEffect, useState } from 'react'
import { LoginAuthUseCase } from '../../../Domain/useCases/auth/LoginAuth';
import { SaveUserUseCase } from
'../../../Domain/useCases/userLocal/SaveUserLocal';
import { GetUserLocalUseCase } from
'../../../Domain/useCases/userLocal/GetUserLocal';
import { useUserLocal } from '../../../hooks/useUserLocal';

const HomeViewModel = () => {
  const [errorMessage, setErrorMessage] = useState('');
  const [values, setValues] = useState(
    {
      email: '',
      password: ''
    }
  );

  const { user, getSession } = useUserLocal();
  console.log('Usuario: ' + JSON.stringify(user));

  useEffect(() => { //Se ejecuta cuando se instancia el viewModel
    getSession();
  }, []);

  const onChange = (property: string, value: any) => {
    setValues({...values, [property]: value});
  }

  const login = async () => {
    if (isValidForm()) {
      const response = await LoginAuthUseCase(values.email,
values.password);
      console.log('Respuesta: ' + JSON.stringify(response));
      if(!response.success) {
        setErrorMessage(response.message);
      }
      else {
        await SaveUserUseCase(response.data);
      }
    }
  }
}
```

```

        getUserSession();
    }
}
};

const isValidForm = () => {
    if (values.email === '') {
        setErrorMessage('El email es requerido');
        return false;
    }
    if (values.password === '') {
        setErrorMessage('La contraseña es requerida');
        return false;
    }
    return true;
}

return {
    ...values,
    user,
    onChange,
    login,
    errorMessage
}
}

export default HomeViewModel;

```

Recargue la App y obtendrá de nuevo los datos del usuario, pero usando el custom hook.

Navegar usando el Custom hook

Se creará una nueva pantalla a desplegar después del inicio de sesión.

En “../views” crear una carpeta “profile/” y en ella una carpeta “info/” y en esta última, el archivo “ProfileInfo.tsx”:

```

import { StackScreenProps } from '@react-navigation/stack';
import React from 'react';
import { View, Text, Button } from 'react-native';
import { RootStackParamList } from '../../App';
import useViewModel from '../ViewModel';

```

```

interface Props extends StackScreenProps<RootStackParamList,
'ProfileInfoScreen'> { };

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

  const { removeSession } = useViewModel();

  return (
    <View style={{ flex: 1, justifyContent: 'center', alignItems:
'center' }}>
      <Button
        onPress={() => {
          removeSession();
          navigation.navigate('HomeScreen');
        }}
        title="Cerrar Sesion"
      />
    </View>
  )
}

```

Registrar la nueva pantalla en el stack navigator. En el archivo App.tsx:

```

import * as React from 'react';
import { NavigationContainer } from '@react-navigation/native';
import { createNativeStackNavigator } from '@react-navigation/native-stack';
import { HomeScreen } from '../src/Presentation/views/home/Home';
import { RegisterScreen } from '../src/Presentation/views/register/Register';
import { ProfileInfoScreen } from
'../src/Presentation/views/profile/info/ProfileInfo';

export type RootStackParamList = {
  HomeScreen: undefined;
  RegisterScreen: undefined;
  ProfileInfoScreen: undefined;
};

```

y el stack screen:

```

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

```

```

    </NavigationContainer>
  );
};

```

En el archivo “../views/home/ViewModel.tsx”, exportar el user:

```

    return {
      ...values,
      user,
      onChange,
      login,
      errorMessage
    }
  }
}

export default HomeViewModel;

```

En el archivo “../views/home/Home.tsx”:

```

import React, { useState, useEffect } from 'react'
import { useNavigation } from '@react-navigation/native';

import { View, Text, Image, TextInput, StyleSheet, ToastAndroid,
TouchableOpacity } from 'react-native'
import { RoundedButton } from
'../../../../Presentation/components/RoundedButton';
import { StackNavigationProp, StackScreenProps } from '@react-
navigation/stack';
import { RootStackParamList } from '../../../../App';
import useViewModel from './ViewModel';
import { CustomTextInput } from '../../../../components/CustomTextInput';
import styles from './Styles';

interface Props extends StackScreenProps<RootStackParamList,
'HomeScreen'>{};

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

```

```

    const { email, password, errorMessage, user, onChange, login } =
useViewModel();

    //const navigation =
useNavigation<StackNavigationProp<RootStackParamList>>();

    useEffect(() => {
        if (errorMessage !== '') {
            ToastAndroid.show(errorMessage, ToastAndroid.LONG);
        }
    }, [errorMessage]);

    useEffect(() => {
        if (user?.id !== null && user?.id !== undefined) {
            navigation.replace('ProfileInfoScreen');
        }
    }, [user]);

    return (
        <View style={styles.container}>
            <Image
                source={require('../../../../assets/chef.jpg')}
                style={styles.imageBackground}
            />
            <View style={styles.logoContainer}>
                <Image
                    source={require('../../../../assets/logo.png')}
                    style={styles.logoImage}
                />
                <Text style={styles.logoText}>FOOD APP</Text>
            </View>
            <View style={styles.form}>
                <Text style={styles.formText}>INGRESAR</Text>

                <CustomTextInput
                    image={require('../../../../assets/email.png')}
                    placeholder='Correo electrónico'
                    value={email}
                    keyboardType='email-address'
                    property='email'
                    onChangeText={onChange}
                />
            </View>
        </View>
    );

```



```

        <CustomTextInput
          image={require('../assets/password.png')}
          placeholder='Contraseña'
          value={password}
          keyboardType='default'
          secureTextEntry={true}
          property='password'
          onChangeText={onChange}
        />
        <View style={{ marginTop: 30 }}>
          <RoundedButton text='ENTRAR' onPress={() => login()} />
        </View>

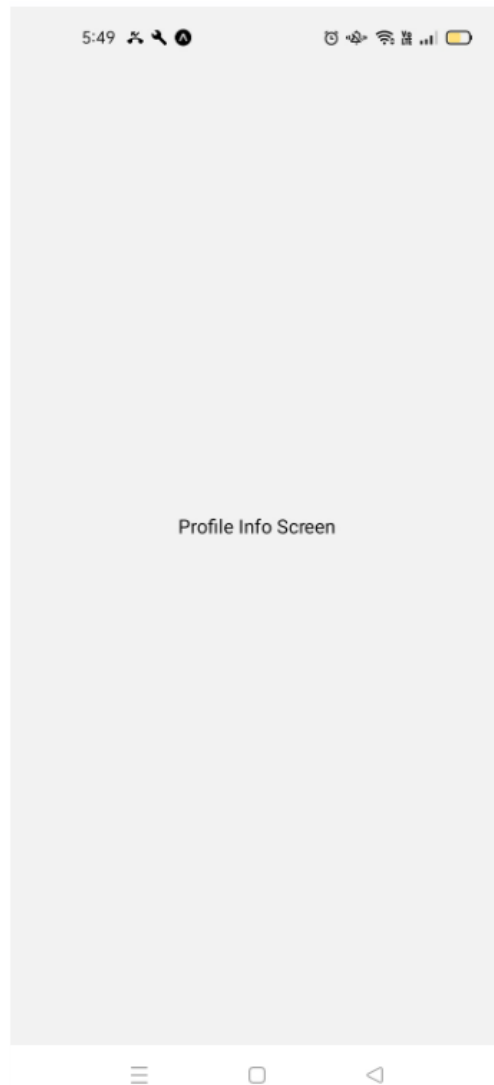
        <View style={styles.formRegister}>
          <Text>¿No tienes cuenta?</Text>
          <TouchableOpacity onPress={() =>
navigation.navigate('RegisterScreen')}>
            <Text
style={styles.formRegisterText}>Registrate</Text>
          </TouchableOpacity>
        </View>

      </View>

    </View>
  );
}

```

Al recargar la App, se mostrará la nueva pantalla, puesto que ya existe una sesión de usuario:



Cerrar la sesión

En “../useCases/userLocal/” crear un nuevo archivo “RemoveUserLocal.tsx”:

```
import { UserLocalRepositoryImp } from
'../../../../Data/repositories/UserLocalRepository';
import { User } from '../../../../entities/User';

const { remove } = new UserLocalRepositoryImp();

export const RemoveUserLocalUseCase = async () => {
  return await remove();
}
```

Cree el archivo “views/profile/info/ViewModel.tsx”:

```
import React from 'react'
import { RemoveUserUseCase } from
'../../../../../Domain/useCases/userLocal/RemoveUserLocal';

export const ProfileInfoViewModel = () => {
  const removeSession = async () => {
    await RemoveUserLocalUseCase();
  }
  return {
    removeSession
  }
}

export default ProfileInfoViewModel;
```

Modificar el archivo ../home/profile/info/ProfileInfo.tsx”:

```
import { StackScreenProps } from '@react-navigation/stack';
import React from 'react';
import { View, Text, Button } from 'react-native';
import { RootStackParamList } from '../../../../../App';
import useViewModel from './ViewModel';

interface Props extends StackScreenProps<RootStackParamList,
'ProfileInfoScreen'> { };

export const ProfileInfoScreen = ({ navigation, route }: Props) => {
  const { removeSession } = useViewModel();
  return (
    <View style={{ flex: 1, justifyContent: 'center', alignItems:
'center' }}>
      <Button
        onPress={() => {
          removeSession();
          navigation.navigate('HomeScreen');
        }}
        title="Cerrar Sesion"
      />
    </View>
  )
}
```