Ing. Marco León Mora Méndez

SENA - ceet

DESARROLLO DE APPS CON REACT NATIVE

PARTE 2

Contenido

[RESUMEN DE COMPONENTES Y ATRIBUTOS 3](#_Toc160980330)

[MANEJO DE IMÁGENES 5](#_Toc160980331)

[Cargar imágenes 5](#_Toc160980332)

[Opción para usar la cámara 7](#_Toc160980333)

[Creando el Modal para seleccionar entre galería o cámara 7](#_Toc160980334)

[Programar la acción para utilizar la cámara 14](#_Toc160980335)

[Usando FIREBASE como repositorio de imágenes 16](#_Toc160980336)

[Cambios a la App para subir la imagen al servidor 25](#_Toc160980337)

[CONTROL DE SESION 29](#_Toc160980338)

[Guardar la sesión 29](#_Toc160980339)

[Utilizar Roles de usuario 37](#_Toc160980340)

[Listar roles de un usuario 40](#_Toc160980341)

[Desplegar los roles en la App 44](#_Toc160980342)

[Agregar los roles a la entidad 44](#_Toc160980343)

[Crear una pantalla básica 45](#_Toc160980344)

[Mejorando la pantalla de roles 48](#_Toc160980345)

[Agregando una barra de navegación 51](#_Toc160980346)

[PERFILES DE USUARIO 62](#_Toc160980347)

[Ver información del usuario 62](#_Toc160980348)

[Modificar la información del usuario 68](#_Toc160980349)

[Agregar las funcionalidades al back-end 77](#_Toc160980350)

[Creando variables de contexto 88](#_Toc160980351)

[Mensaje informativo 97](#_Toc160980352)

[Enviar Peticiones seguras 99](#_Toc160980353)

# RESUMEN DE COMPONENTES Y ATRIBUTOS

Hasta este momento, usted ha tenido la oportunidad de crear componentes visuales, modificando su apariencia. Antes de continuar con el taller, se presenta algunos conceptos importantes

## Comparativa entre las etiquetas HTML y los componentes de React Nativ:

| HTML (Etiquetas) | React Native (Componentes) |
| --- | --- |
| <div> | View |
| <span> | Text |
| <img> | Image |
| <button> | TouchableHighlight |
| <input> | TextInput |
| <a> | Linking (para enlaces web) |
| <ul> | FlatList (para listas) |
| <li> | ListItem (para elementos de lista) |
| <p> | Text (para párrafos) |
| <h1>, <h2>, … | Text (para títulos) |
| <form> | TextInput + Button (para formularios) |

Recuerda que los componentes de React Native están diseñados específicamente para dispositivos móviles y se basan en componentes nativos de la plataforma (como View, Text, Image, etc.), mientras que las etiquetas HTML son parte del lenguaje de marcado web.

## Comparativa entre las propiedades CSS y los Styles de React Native:

| **CSS (Propiedades)** | **React Native (Styles)** |
| --- | --- |
| background-color | backgroundColor |
| color | color |
| font-size | fontSize |
| font-weight | fontWeight |
| margin | margin |
| padding | padding |
| border | borderWidth, borderColor, borderRadius |
| text-align | textAlign |
| display | display |
| flex | flex |
| position | position |
| width | width |
| height | height |

Recuerde que los componentes de React Native utilizan JavaScript para definir sus estilos. Aunque los nombres y valores son similares a CSS, se escriben en camel case (por ejemplo, backgroundColor en lugar de background-color)

## Algunos conceptos clave en React Native

Conceptos que se han utilizado o se usarán más adelante en este taller:

**Props (Propiedades)**: Las props (abreviatura de propiedades) son la forma principal de pasar datos entre componentes en React Native. Son valores inmutables que se pasan de un componente padre a un componente hijo. Permiten la comunicación y la transferencia de datos entre componentes.

**Hooks:** Los hooks son funciones proporcionadas por React que permiten conectarse a características de React desde componentes funcionales. Reemplazan la necesidad de usar clases para manejar el estado y los ciclos de vida. Ejemplos comunes de hooks: useState, useEffect, useContext, etc.

**useEffect:** El hook useEffect se utiliza para manejar efectos secundarios en componentes funcionales. Permite realizar tareas como obtener datos, cambiar manualmente el DOM, etc.

**Context:** El contexto es una forma de compartir datos entre componentes sin necesidad de pasar props manualmente a través de cada nivel del árbol de componentes. Se utiliza junto con el API de React Context, cuando el proveedor de contexto se actualiza, los componentes que consumen ese contexto se vuelven a renderizar con el último valor de contexto.

# MANEJO DE IMÁGENES

En este apartado se utilizará una dependencia para seleccionar o capturar imágenes imágenes.

## Cargar imágenes

En Google buscar “React native expo image picker”, en la página “npmjs.com/package/expo-image-picker”, copiar la línea de código en la casilla install y ejecutarla en la consola:

npm i expo-image-picker

En el archivo “.../views/register/ViewModel.tsx” modificar así:

import \* as ImagePicker from "expo-image-picker";

const RegisterViewModel = () => {

    const [errorMessage, setErrorMessage] = useState('');

    const [values, setValues] = useState({

        name: '',

        lastname: '',

        email: '',

        phone: '',

        image: '',

        password: '',

        confirmPassword: ''

    });

    const [file, setFile] = useState<ImagePicker.ImagePickerAsset>();

    const pickImage = async () => {

        let result = await ImagePicker.launchImageLibraryAsync({

            mediaTypes: ImagePicker.MediaTypeOptions.All,

            allowsEditing: true,

            quality: 1,

        });

        if (!result.canceled) {

            onChange('image', result.assets[0].uri);

            setFile(result.assets[0]);

        }

    };

Tenga en cuenta agregar el atributo “image” si no está en la lista de propiedades.

AL final exponer el metódo pickImage:

    return {

        ...values,

        onChange,

        register,

        pickImage,

        errorMessage

    }

}

export default RegisterViewModel;

En el archivo “../views/register/Register.tsx”:

Al inicio agregar el método pickImage para desestructurarlo también:

export const RegisterScreen = () => {

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

Y modificar el contenedor de la imagen para que, si existe la uri, cambie la imagen por defecto (user\_image.png’:

      <View style={styles.logoContainer}>

        <TouchableOpacity onPress={() => pickImage()}>

          {

          image == ''

          ?

          <Image

            source={require('../../../../assets/user\_image.png')}

            style={styles.logoImage}

          />

        :

          <Image

            source={{ uri: image }}

            style={styles.logoImage}

          />

      }

        </TouchableOpacity>

        <Text style={styles.logoText}>SELECCIONA UNA IMAGEN</Text>

      </View>

Si pulsa sobre la imagen predefinida, se abre el explorador de imagen y puede seleccionar una:

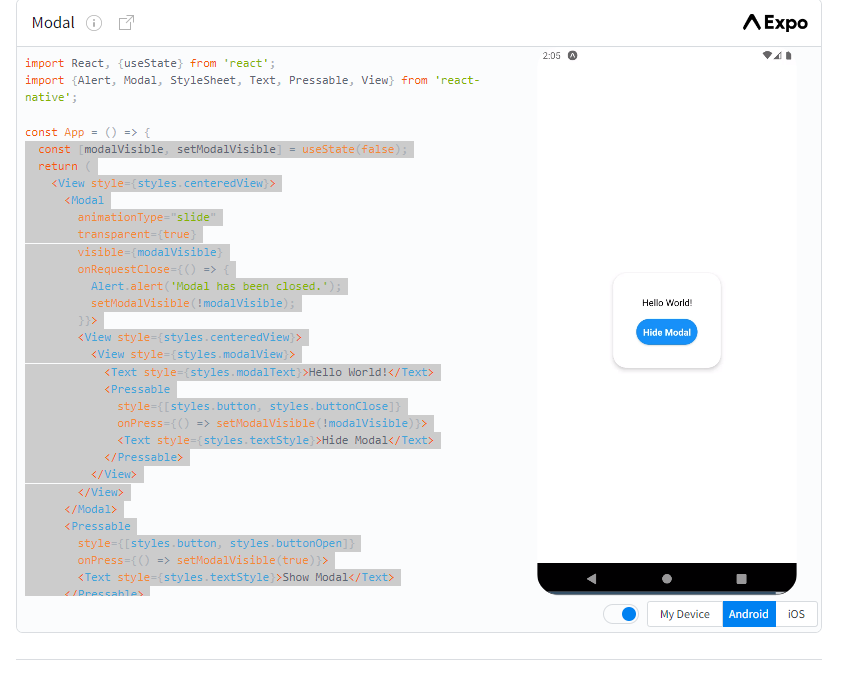


## Opción para usar la cámara

### Creando el Modal para seleccionar entre galería o cámara

Hay que crear una ventana modal para seleccionar entre cargar archivo de imagen o capturar la imagen con la cámara.

Buscar en Goole por “react native components” y abrir Core CVomponents and APIs. Buscar la documentación de “Modal” en la parte inferior. Para probar, puede ejecutar el simulador al lado derecho del bloque de código.



Copiar el código sombreado, en la imagen anterior hasta el final, justo antes de la última línea (export default App;), esta línea NO se copia.

Crear el archivo “../Presentation/Components/ModalPickImage.tsx”, genere el snippet rafce, reemplace todo el return con el código copiado. Después copie la parte superior de los import y agréguela al principio del archivo. Después hay que agregar los parámetros necesarios para hacer funcional el Modal y configurar los botones con el componente anteriormente desarrollado:

import React, { useState } from 'react';

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

import { RoundedButton } from './RoundedButton';

interface Props {

    openGallery: () => void,

    openCamera: () => void,

    modalUseState: boolean,

    setModalUseState: React.Dispatch<React.SetStateAction<boolean>>,

}

const ModalPickImage = ({ openGallery, openCamera, setModalUseState, modalUseState}: Props) => {

    const [modalVisible, setModalVisible] = useState(false);

    return (

        <View style={styles.centeredView}>

            <Modal

                animationType="slide"

                transparent={true}

                visible={modalUseState}

                onRequestClose={() => {

                    Alert.alert('Modal has been closed.');

                    setModalUseState(!modalUseState);

                }}>

                <View style={styles.centeredView}>

                    <View style={styles.modalView}>

                       <RoundedButton

                        onPress = { () => openGallery()}

                        text={'Galería'}

                        />

                       <RoundedButton

                        onPress = { () => openCamera()}

                        text={'Cámara'}

                        />

                    </View>

                </View>

            </Modal>

        </View>

    );

};

const styles = StyleSheet.create({

    centeredView: {

        flex: 1,

        justifyContent: 'center',

        alignItems: 'center',

        marginTop: 22,

    },

    modalView: {

        margin: 20,

        backgroundColor: 'white',

        borderRadius: 20,

        padding: 35,

        alignItems: 'center',

        shadowColor: '#000',

        shadowOffset: {

            width: 0,

            height: 2,

        },

        shadowOpacity: 0.25,

        shadowRadius: 4,

        elevation: 5,

    },

    button: {

        borderRadius: 20,

        padding: 10,

        elevation: 2,

    },

    buttonOpen: {

        backgroundColor: '#F194FF',

    },

    buttonClose: {

        backgroundColor: '#2196F3',

    },

    textStyle: {

        color: 'white',

        fontWeight: 'bold',

        textAlign: 'center',

    },

    modalText: {

        marginBottom: 15,

        textAlign: 'center',

    },

});

export default ModalPickImage

En la pantalla de registro (“../components/views/register/Register.tsx”), se cambia el método pickImage(). Agregue el código para que el archivo quede así:

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

import { View, Text, Image, ScrollView, ToastAndroid, TouchableOpacity } from 'react-native'

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

import useViewModel from './ViewModel';

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

import styles from './Styles';

import ModalPickImage from '../../components/ModalPickImage';

export const RegisterScreen = () => {

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

  const [modalVisible, setModalVisible] = useState(false);

  //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}>

        <TouchableOpacity onPress={() => setModalVisible(true)}>

          {

          image == ''

          ?

          <Image

            source={require('../../../../assets/user\_image.png')}

            style={styles.logoImage}

          />

        :

          <Image

            source={{ uri: image }}

            style={styles.logoImage}

          />

        }

        </TouchableOpacity>

Ejecute y en la pantalla registro, pulse sobre el icono de seleccionar imagen, la ventana modal aparecerá. mejore su presentación modificando el componente en “../Presentation/Components/ModalPickImage.tsx”:

import React, { useState } from 'react';

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

import { RoundedButton } from './RoundedButton';

interface Props {

    openGallery: () => void,

    openCamera: () => void,

    modalUseState: boolean,

    setModalUseState: React.Dispatch<React.SetStateAction<boolean>>,

}

const ModalPickImage = ({ openGallery, openCamera, setModalUseState, modalUseState }: Props) => {

    const [modalVisible, setModalVisible] = useState(false);

    return (

        <View style={styles.centeredView}>

            <Modal

                animationType="slide"

                transparent={true}

                visible={modalUseState}

                onRequestClose={() => {

                    Alert.alert('Modal has been closed.');

                    setModalUseState(!modalUseState);

                }}>

                <View style={styles.centeredView}>

                    <View style={styles.modalView}>

                        <Text style={styles.modalText}>Seleccione una opción</Text>

                        <View style={styles.buttonContainer}>

                            <RoundedButton

                                onPress={() => {

                                    openGallery()

                                    setModalUseState(false)

                                }}

                                text={'Galería'}

                            />

                        </View>

                        <View style={styles.buttonContainer}>

                            <RoundedButton

                                onPress={() => {

                                    openCamera()

                                    setModalUseState(false)}}

                                text={'Cámara'}

                            />

                        </View>

                    </View>

                </View>

            </Modal>

        </View>

    );

};

const styles = StyleSheet.create({

    centeredView: {

        flex: 1,

        justifyContent: 'center',

        alignItems: 'center',

        marginTop: 22,

    },

    modalView: {

        width: 250,

        height: 200,

        margin: 20,

        backgroundColor: 'white',

        borderRadius: 20,

        paddingTop: 25,

        paddingLeft: 20,

        paddingRight: 20,

        alignItems: 'center',

        shadowColor: '#000',

        shadowOffset: {

            width: 0,

            height: 2,

        },

        shadowOpacity: 0.25,

        shadowRadius: 4,

        elevation: 5,

    },

    button: {

        borderRadius: 20,

        padding: 10,

        elevation: 2,

    },

    buttonOpen: {

        backgroundColor: '#F194FF',

    },

    buttonClose: {

        backgroundColor: '#2196F3',

    },

    textStyle: {

        color: 'white',

        fontWeight: 'bold',

        textAlign: 'center',

    },

    modalText: {

        marginBottom: 15,

        textAlign: 'center',

    },

    buttonContainer: {

        width: '100%',

        marginTop: 8,

    },

});

export default ModalPickImage

//<View style ={ styles.buttonContainer}>

Pruebe el código, los dos botones llevaran a la galería.

### Programar la acción para utilizar la cámara

En el archivo “../views/register/ViewModel.tsx” copie el método pickImage y modifíquelo así:

    const pickImage = async () => {

        let result = await ImagePicker.launchImageLibraryAsync({

            mediaTypes: ImagePicker.MediaTypeOptions.All,

            allowsEditing: true,

            quality: 1,

        });

        if (!result.canceled) {

            onChange('image', result.assets[0].uri);

            setFile(result.assets[0]);

        }

    };

    const takePhoto = async () => {

        let result = await ImagePicker.launchCameraAsync({

            mediaTypes: ImagePicker.MediaTypeOptions.All,

            allowsEditing: true,

            quality: 1,

        });

        if (!result.canceled) {

            onChange('image', result.assets[0].uri);

            setFile(result.assets[0]);

        }

    };

Al final del archivo hay que exponer el método creado:

    return {

        ...values,

        onChange,

        register,

        pickImage,

        takePhoto,

        errorMessage

    }

}

export default RegisterViewModel;

(“../components/views/register/Register.tsx”:

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

Y al final:

      <ModalPickImage

        openGallery={pickImage}

        openCamera={takePhoto}

        setModalUseState={setModalVisible}

        modalUseState={modalVisible}

      />

    </View>

  );

}

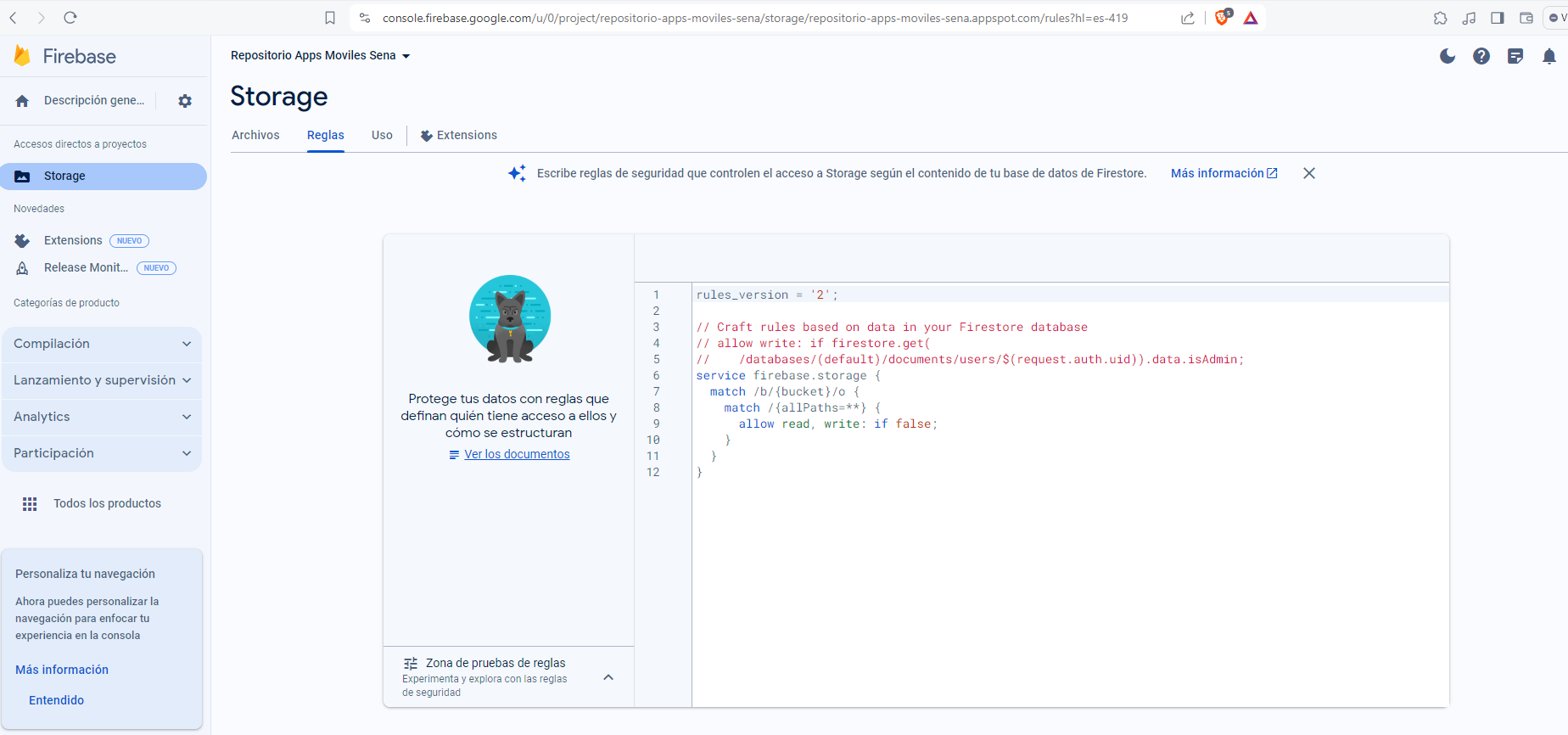
Con esto debería poder utilizar la cámara de su dispositivo físico. Recuerde que con el emulador no es posible probarla.

## Usando FIREBASE como repositorio de imágenes

En la página de firebase.google.com/, click en la opción “Ir a la consola”, registrarse si aun no lo ha hecho, después en “Crear un proyecto”, ingrese el nombre que desee (Repositorio Apps Moviles Sena), confirmar y continuar, acepte las condiciones y “Crear Proyecto”, continuar. En la nueva ventana, a la izquierda seleccionar Compilación -> Storage y el botón “Comenzar”.

En la ventana emergente seleccione “Comenzar en modo de prueba” y “siguiente” y “Listo”.

En la ventana “Storage”, seleccione la pestaña “Reglas”, deberá tener algo como:



Seleccione la pestaña “Archivos”. En este momento se pueden subir archivos de manera manual con el botón “Subir archivo”.

Antes de continuar, hay que instalar algunas dependencias, en la terminal del servidor (OJO: en la carpeta de NODE.JS) ingrese los siguientes comandos:

npm @google-cloud/storage

npm i multer

Crear el archivo “config/env.js”:

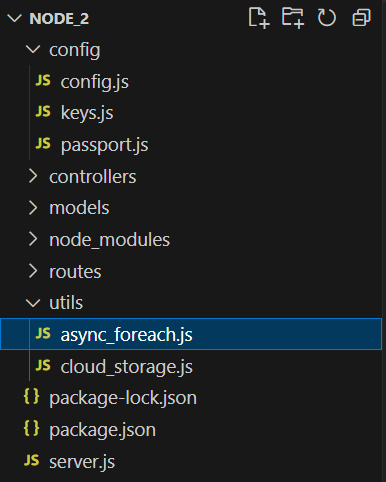
module.exports = {

    production: false,

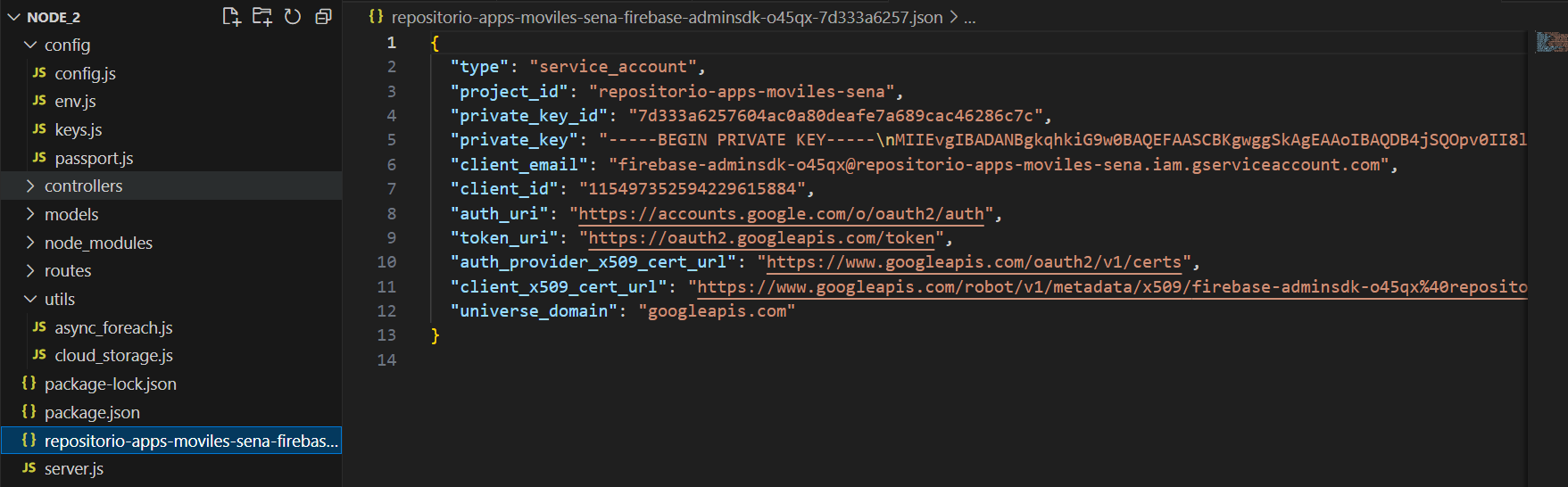
    id\_developer: 0

}

En su servidor (NODE.JS), en el primer nivel, crear una carpeta “utils” y en ella dos archivos: “async\_foreach.js” y cloud\_storage\_js”



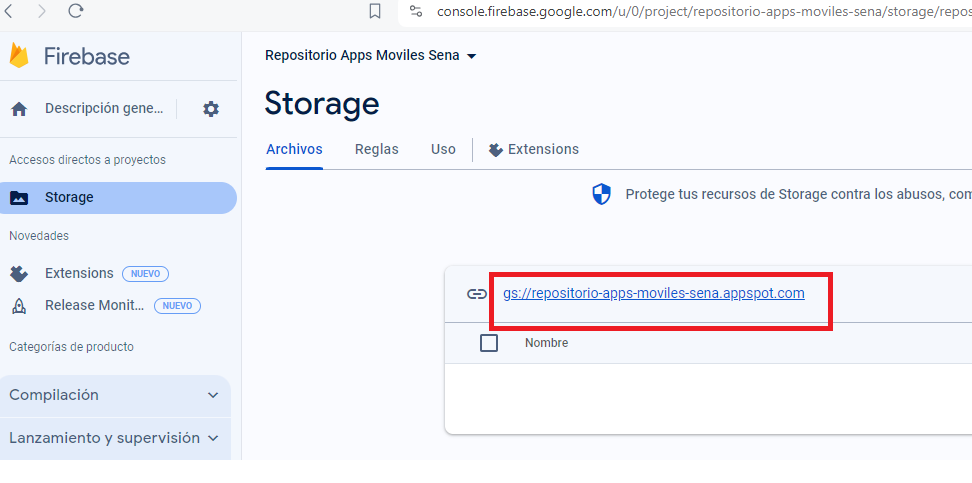
En la página de FireBase, seleccione el ícono del engranaje - “Configuración del proyecto” – “Cuentas de servicio”. En la nueva ventana debe estar seleccionada la opción “Node.js” y pulse el botón “Generar nueva clave privada”. El archivo que se descarga lo debe copiar en la carpeta raíz del servidor.



Ese archivo contiene la configuración para FireBase, ahora debe renombrarlo como “**serviceAccountKey.json**”

Hay que copiar el ID del proyecto. Buscarlo en la página de FireBase en la pestaña “General” copiar el ID del proyecto, que se usara en el siguiente archivo, en el objeto storage.

En la misma página, en la pestaña “Storage” copian la ruta:



En el archivo “../cloud\_storage.js” quedará:

const { Storage } = require('@google-cloud/storage');

const { format } = require('util');

const env = require('../config/env')

const url = require('url');

const { v4: uuidv4 } = require('uuid');

const uuid = uuidv4();

const storage = new Storage({

    projectId: "repositorio-apps-moviles-sena", // ID del proyecto COPIADO de Firebase

    keyFilename: './serviceAccountKey.json' //Archivo descargado de FireBase y renombrado

});

// RUTA COPIADA DE FIREBASE STORAGE, incluir el slash final

const bucket = storage.bucket("gs://repositorio-apps-moviles-sena.appspot.com/");

/\*\*

 \* Subir el archivo a Firebase Storage

 \* @param {File} file objeto que sera almacenado en Firebase Storage

 \*/

module.exports = (file, pathImage, deletePathImage) => {

    return new Promise((resolve, reject) => {

        console.log('delete path', deletePathImage)

        if (deletePathImage) {

            if (deletePathImage != null || deletePathImage != undefined) {

                const parseDeletePathImage = url.parse(deletePathImage)

                var ulrDelete = parseDeletePathImage.pathname.slice(23);

                const fileDelete = bucket.file(`${ulrDelete}`)

                fileDelete.delete().then((imageDelete) => {

                    console.log('se borro la imagen con exito')

                }).catch(err => {

                    console.log('Failed to remove photo, error:', err)

                });

            }

        }

        if (pathImage) {

            if (pathImage != null || pathImage != undefined) {

                let fileUpload = bucket.file(`${pathImage}`);

                const blobStream = fileUpload.createWriteStream({

                    metadata: {

                        contentType: 'image/png',

                        metadata: {

                            firebaseStorageDownloadTokens: uuid,

                        }

                    },

                    resumable: false

                });

                blobStream.on('error', (error) => {

                    console.log('Error al subir archivo a firebase', error);

                    reject('Something is wrong! Unable to upload at the moment.');

                });

                blobStream.on('finish', () => {

                    // The public URL can be used to directly access the file via HTTP.

                    const url = format(`https://firebasestorage.googleapis.com/v0/b/${bucket.name}/o/${fileUpload.name}?alt=media&token=${uuid}`);

                    console.log('URL DE CLOUD STORAGE ', url);

                    resolve(url);

                });

                blobStream.end(file.buffer);

            }

        }

    });

}

El archivo “utils/async\_foreach.js”:

module.exports = async function(array, callback) {

    for (let index = 0; index < array.length; index++) {

        await callback(array[index], index, array);

    }

}

Ahora, en el archivo “server.js” agregar las líneas señaladas:

const express = require('express');

const passport = require('passport');

const multer = require('multer'); //Agregado para FireBase

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);

//Agregado para FireBase

const upload = multer({

    storage: multer.memoryStorage()

});

/\*\*

 \* Llamar a las rutas

 \*/

usersRoutes(app, upload);

/\* Iniciar el servidor

\*/

//ETC…

En el archivo “controllers/usersController.js”, crear un nuevo método que permita almacenar los datos del usuario y la imagen. El archivo quedará así:

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

const bcrypt = require('bcryptjs');

const jwt = require('jsonwebtoken');

const keys = require('../config/keys');

const storage = require('../utils/cloud\_storage'); //Agregado para FireBase

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 o correo incorrecto'

                });

            }

        });

    },

    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

            });

        });

    },

    //Agregado para FireBase

    async registerWithImage(req, res) {

        const user = JSON.parse(req.body.user);

        const files = req.files;

        if(files.length > 0){

            const path = `image\_${Date.now()}`;

            const url = await storage(files[0], path);

            if(url != undefined && url != null){

                user.image = url;

            }

        }

        User.create(user, (err, data) => {

            if (err) {

                return res.status(501).json({

                    success: false,

                    message: 'Error al crear el usuario',

                    error: err

                });

            }

            user.id = `${data}`;

            const token = jwt.sign({ id: user.id, email: user.email }, keys.secretOrKey, {});

            user.session\_token = `JWT ${token}`

;

            return res.status(201).json({

                success: true,

                message: 'Usuario creado ',

                data: user

            });

        });

    },

}

En el archivo “routes/userRoutes”:

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

module.exports = (app, upload) => {

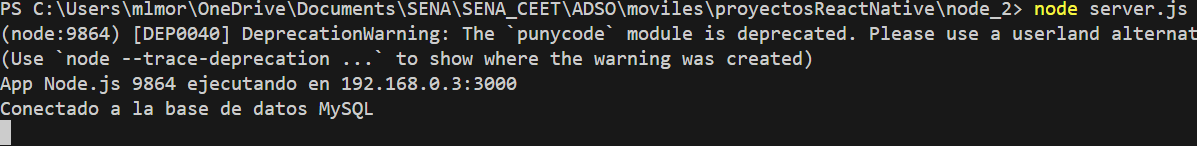
    app.post('/api/users/create', userController.register);

    app.post('/api/users/createWithImage', upload.array('image', 1), userController.registerWithImage);

    app.post('/api/users/login', userController.login);

}

Si todo lo anterior está en orden el servidor se levantará con el aviso normal:



## Cambios a la App para subir la imagen al servidor

En la App, en “../Domain/repositories/AuthRepository.tsx”, crear el método en la interfaz:

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

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

import \* as ImagePicker from 'expo-image-picker';

export interface AuthRepository {

    login(email: string, password: string ): Promise<ResponseApiDelivery>;

    register(user: User): Promise<ResponseApiDelivery>;

    registerWithImage(user: User, file: ImagePicker.ImageInfo): Promise<ResponseApiDelivery>;

}

En consola, para tener acceso a los tipos de archivos, instalar:

npm i mimi

npm i --save-dev @types/mime

En “Data/repositories/AuthRepository.tsx”:

import { ApiDelivery, ApiDeliveryForImage } from "../sources/remote/api/ApiDelivery";

import mime from 'mime';

import { ImageInfo } from "expo-image-picker";

y añadir el método:

    async registerWithImage(user: User, file: ImageInfo ): Promise<ResponseApiDelivery> {

        try {

            let data = new FormData();

            data.append('image', { //'image' is the same name of the field in the API routes

                // @ts-ignore

                uri: file.uri,

                name: file.uri.split('/').pop(),

                type: mime.getType(file.uri)!

            } as any);

            data.append('user', JSON.stringify(user)); //'user' is the same name of the field in the API routes

            const response = await ApiDeliveryForImage.post<ResponseApiDelivery>('/users/ createWithImage

', data);

            //registerWithImage is the route in the API

            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);

        }

    }

Cree el archive “../Domain/useCases/auth/RegisterWithImageAuth.tsx”, copie el archivo “RegisterAuth.tsx” y modifique así:

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

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

import \* as ImagePicker from 'expo-image-picker';

const { registerWithImage} = new AuthRepositoryImpl();

export const RegisterWithImageAuthUseCase = async (user: User, file: ImagePicker.ImageInfo) =>{

    return await registerWithImage(user, file);

}

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

import { RegisterWithImageAuthUseCase } from "../../../Domain/useCases/auth/RegisterWithImageAuth";

Modificar el método register así:

const register = async () => {

        if (!isValidForm()) {

            //const response = await RegisterAuthUseCase(values);

            const response = await RegisterWithImageAuthUseCase(values, file!);

            console.log('Result' + JSON.stringify(response));

        }

    }

Y al final de las validaciones, agregar:

        if (values.image === '') {

            setErrorMessage('Seleccione una imagen de perfil');

            return false;

        }

        return true;

En “../Data/sources/remote/api/ApyDelivery.tsx”:

import axios from 'axios';

const ApiDelivery = axios.create({

    //baseURL: 'http://192.168.0.4:3000/api',

    baseURL: 'http://192.168.0.3:3000/api',

    headers: {

        'Content-Type': 'application/json',

    }

});

const ApiDeliveryForImage = axios.create({

    baseURL: 'http://192.168.0.3:3000/api',

    headers: {

        'Content-Type': 'multipart/form-data',

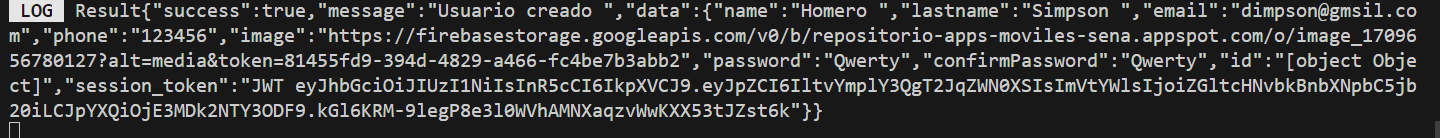
        'Accept': 'application/json',

    }

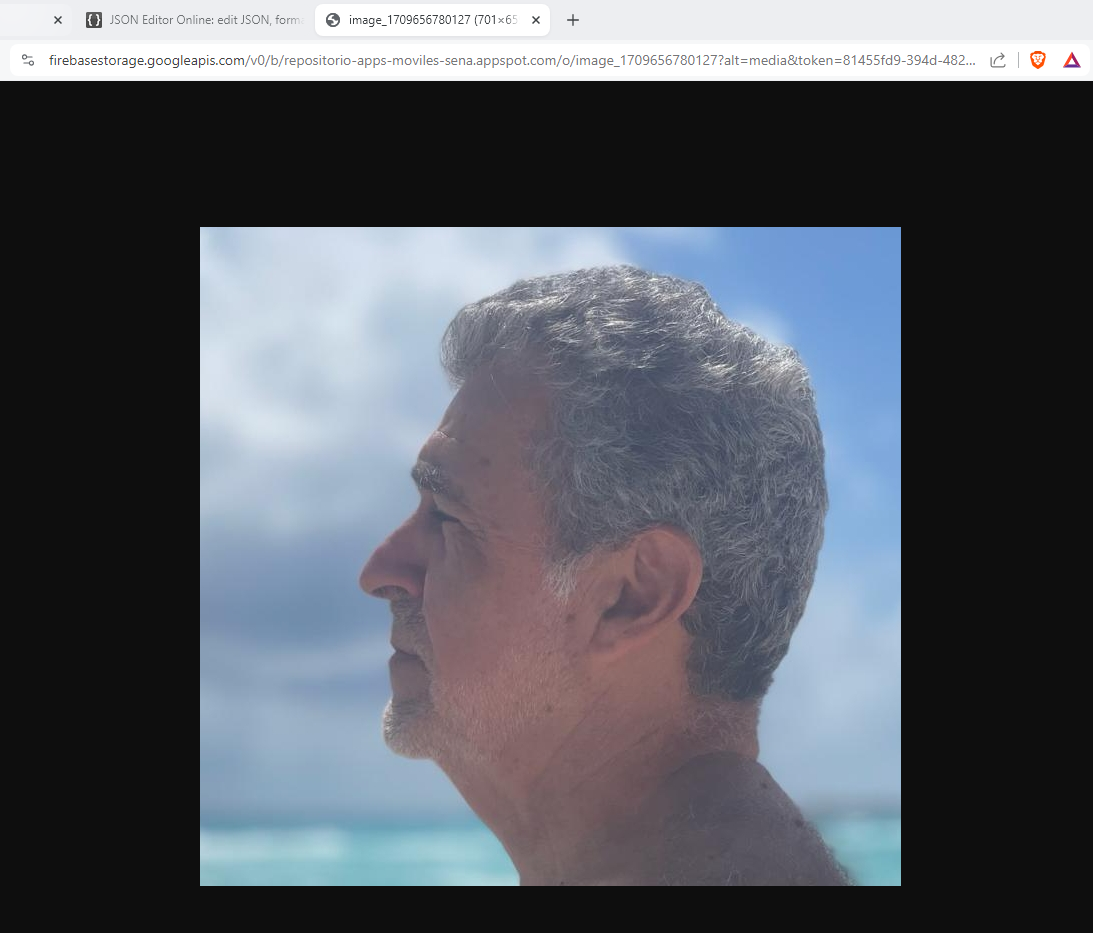
});

export { ApiDelivery, ApiDeliveryForImage}

Ejecute la aplicación y verifique errores, registre un nuevo usuario incluyendo una imagen, verifique en la consola:



Copie el valor del campo “image” y péguela en un navegador, deberá ver la imagen cargada:

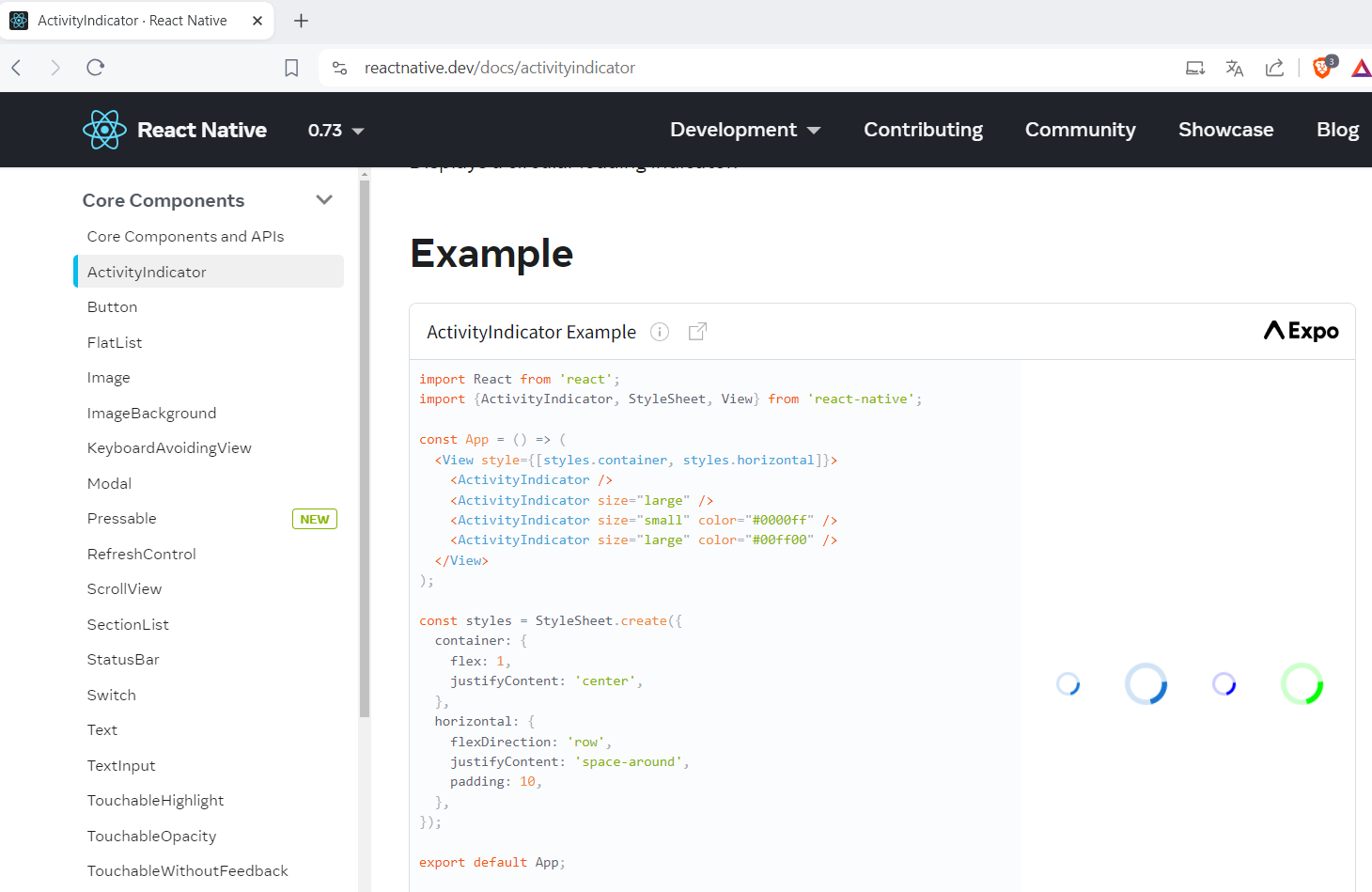


Por su puesto, esta url debe aparecer en el registro de la BD, confirme con MySQL Woekbench.

# CONTROL DE SESION

Para almacenar la información del usuario, se harán algunas modificaciones en la navegación para que, después del registro del nuevo usuario se registre localmente y lo lleve a la ventana correspondiente.

También insertar un componente para indicar que se está procesando la carga de la imagen en el servidor remoto. Consulte en la página de React Native, el componente “Activity Indicator”



## Guardar la sesión

En “..Domain/useCases/userLocal/SaveUserLocal.tsx” verificar que se haya cambiado el nombre del método:al

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);

}

En “../register/ViewModel.tsx”, verifique y agregue el código adicional:

/\*\*

 \* View model for the register view

 \* It handles the logic of the view and the state of the input values

 \* values: State variable to handle the input values

 \* setValues: Function to update the state values

 \* onChange: Function to handle the change of the input values

 \* register: Function to handle the register of the user

 \* @returns {Object} - Object with the values and functions to handle the view

 \*/

import React, { useState } from "react"

import { ApiDelivery } from "../../../Data/sources/remote/api/ApiDelivery";

import { RegisterAuthUseCase } from "../../../Domain/useCases/auth/RegisterAuth";

import \* as ImagePicker from "expo-image-picker";

import { RegisterWithImageAuthUseCase } from "../../../Domain/useCases/auth/RegisterWithImageAuth";

import { SaveUserLocalUseCase } from "../../../Domain/useCases/userLocal/SaveUserLocal";

import { useUserLocal } from "../../hooks/useUserLocal"

const RegisterViewModel = () => {

    const [errorMessage, setErrorMessage] = useState('');

    const [values, setValues] = useState({

        name: '',

        lastname: '',

        email: '',

        phone: '',

        image: '',

        password: '',

        confirmPassword: ''

    });

    const [loading, setloading] = useState(false);

    const [file, setFile] = useState<ImagePicker.ImagePickerAsset>();

    const { user, getUserSession } = useUserLocal();

    const pickImage = async () => {

        let result = await ImagePicker.launchImageLibraryAsync({

            mediaTypes: ImagePicker.MediaTypeOptions.All,

            allowsEditing: true,

            quality: 1,

        });

        if (!result.canceled) {

            onChange('image', result.assets[0].uri);

            setFile(result.assets[0]);

        }

    };

    const takePhoto = async () => {

        let result = await ImagePicker.launchCameraAsync({

            mediaTypes: ImagePicker.MediaTypeOptions.All,

            allowsEditing: true,

            quality: 1,

        });

        if (!result.canceled) {

            onChange('image', result.assets[0].uri);

            setFile(result.assets[0]);

        }

    };

    const onChange = (property: string, value: any) => {

        setValues({ ...values, [property]: value });

    }

    const register = async () => {

        if (isValidForm()) {

            setloading(true);

            const response = await RegisterWithImageAuthUseCase(values, file!);

            setloading(false);

            console.log('Result' + JSON.stringify(response));

            if(response.success){

                await SaveUserLocalUseCase(response.data);

                getUserSession();

            }

            else{

                setErrorMessage(response.message);

            }

        }

        else {  //Agregado para depuracion

            console.log('Formulario no valido');

        }

    }

    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;

        }

        if (values.image === '') {

            setErrorMessage('Seleccione una imagen de perfil');

            return false;

        }

        return true;

    }

    return {

        ...values,

        onChange,

        register,

        pickImage,

        takePhoto,

        errorMessage,

        loading,

        user

    }

}

export default RegisterViewModel;

En “../register/Register.tsx”, verifique y agregue el código adicional:

/\*\*

 \* 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, { useEffect, useState } from 'react'

import { View, Text, Image, ScrollView, ToastAndroid, TouchableOpacity, ActivityIndicator } from 'react-native'

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

import useViewModel from './ViewModel';

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

import styles from './Styles';

import ModalPickImage from '../../components/ModalPickImage';

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

import { RootStackParamList } from '../../../../App';

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

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

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

  const { name, lastname, email, phone, password, image, confirmPassword, loading, errorMessage, user, onChange, register, pickImage, takePhoto } = useViewModel();

  const [modalVisible, setModalVisible] = useState(false);

  //Para saber si la variable ya tiene establecido un valor

  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}>

        <TouchableOpacity onPress={() => setModalVisible(true)}>

          {

            image == ''

              ?

              <Image

                source={require('../../../../assets/user\_image.png')}

                style={styles.logoImage}

              />

              :

              <Image

                source={{ uri: image }}

                style={styles.logoImage}

              />

          }

        </TouchableOpacity>

        <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}

          />

          <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>

      <ModalPickImage

        openGallery={pickImage}

        openCamera={takePhoto}

        setModalUseState={setModalVisible}

        modalUseState={modalVisible}

      />

      {

        loading &&

        <ActivityIndicator

          style={styles.loading}

          size="large"

          color={MyColors.primary}

        />

      }

    </View>

  );

}

En “../register/Styles.tsx” agregar antes del export:

    loading:{

      position: 'absolute',

      bottom: 0,

      top: 0,

      left: 0,

      right: 0,

    }

  });

  export default RegisterStyles;

Cree un nuevo usuario y verifique que el ActivityIndicator se visualiza mientras se carga la imagen en el FireBase Storage.

## Utilizar Roles de usuario

En esta sección se registrará el rol del usuario por defecto. En la BD, crear dos nuevas tablas según el siguiente script:

USE nodejs\_base1;

CREATE TABLE roles(

    id BIGINT PRIMARY KEY AUTO\_INCREMENT,

    name VARCHAR(100) NOT NULL UNIQUE,

    image VARCHAR(255) NULL,

    route VARCHAR(180) NOT NULL,

    created\_at TIMESTAMP(0) NOT NULL,

    updated\_at TIMESTAMP(0) NOT NULL

);

INSERT INTO roles(name, route, created\_at, updated\_at) VALUES

    ('RESTAURANTE', '/restaurant/orders/list', NOW(), NOW()),

    ('REPARTIDOR', '/delivery/orders/list', NOW(), NOW()),

    ('CLIENTE', '/client/products/list', NOW(), NOW());

CREATE TABLE user\_has\_roles(

    id\_user BIGINT NOT NULL,

    id\_rol BIGINT NOT NULL,

    created\_at TIMESTAMP(0) NOT NULL,

    updated\_at TIMESTAMP(0) NOT NULL,

    FOREIGN KEY(id\_user) REFERENCES users(id) ON UPDATE CASCADE ON DELETE CASCADE,

    FOREIGN KEY(id\_rol) REFERENCES roles(id) ON UPDATE CASCADE ON DELETE CASCADE,

    PRIMARY KEY(id\_user, id\_rol)

);

Cada que se cree un nuevo usuario, se le deberá asignar el rol de cliente por defecto. En el servidor Node.JS, crear un archivo “models/rol.js”:

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

const Rol = {};

Rol.create = (id\_user, id\_rol, result) => {

    const sql = `INSERT INTO user\_has\_roles (id\_user, id\_rol, created\_at, updated\_at)

        VALUES (?, ?, ?, ?)`;

    db.query(sql, [id\_user, id\_rol, new Date(), new Date()],

        (err, res) => {

            if (err) {

                console.log('Error al crear: ', err);

            }

            else {

                console.log('Rol creado: ', re.insertId);

                result(null, res.insertId);

            }

        }

    );

}

module.exports = Rol;

En “controllers/usersController.js”:

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

const Rol = require('../models/rol'); //agregar esta linea

y dentro de ‘registerWithImage’, en el método User.create(), agregue el método para crear el registro en la tabla ‘user\_has\_roles’ :

    //Agregado para FireBase

    async registerWithImage(req, res) {

        const user = JSON.parse(req.body.user);

        const files = req.files;

        if(files.length > 0){

            const path = `image\_${Date.now()}`;

            const url = await storage(files[0], path);

            if(url != undefined && url != null){

                user.image = url;

            }

        }

        User.create(user, (err, data) => {

            if (err) {

                return res.status(501).json({

                    success: false,

                    message: 'Error al crear el usuario',

                    error: err

                });

            }

            user.id = `${data.id}`;

            const token = jwt.sign({ id: user.id, email: user.email }, keys.secretOrKey, {});

            user.session\_token = `JWT ${token}`;

            //Agregar desde acá: Crea registro en la tabla user\_has\_roles:

            Rol.create(user.id, 3, (err, data) => {     //3: Rol de cliente

                if (err) {

                    return res.status(501).json({

                        success: false,

                        message: 'Error al asignar el rol al usuario',

                        error: err

                    });

                }

            });

            return res.status(201).json({

                success: true,

                message: 'Usuario creado ',

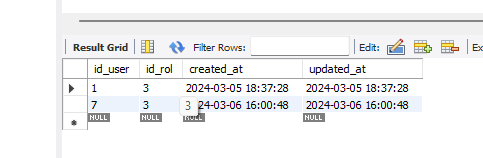
                data: user

            });

        });

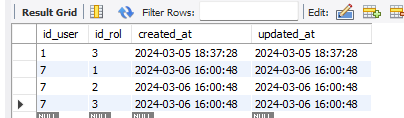
    },

Al crear un nuevo usuario, en la DB debe aparecer, además, el registro correspondiente en la tabla user\_has\_roles, indicando un rol de CLIENTE (id: 3):



## Listar roles de un usuario

Primero, prepare la BD para realizar una consulta que devuelva varios roles asignados a un usuario. En MySQL Workbench inserte dos registros en la tabla ‘user\_has\_roles’, para asignarle todos los roles a un usuario:



Ahora, crear una consulta que devuelva los datos de un usuario y la lista de roles:

SELECT

    U.id,

    U.email,

    U.name,

    U.lastname,

    U.phone,

    U.image,

    U.password,

    JSON\_ARRAYAGG(

        JSON\_OBJECT(

        'id', CONVERT(R.id, char),

        'name', R.name,

        'image', R.image,

        'route', R.route

        )

    ) AS roles

FROM users AS U

INNER JOIN user\_has\_roles AS UHR

    ON UHR.id\_user = U.id

INNER JOIN roles AS R

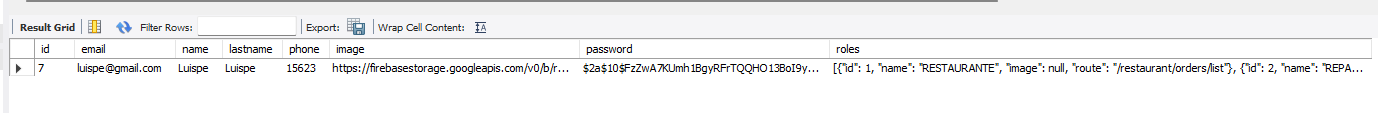
    ON UHR.id\_rol = r.id

WHERE

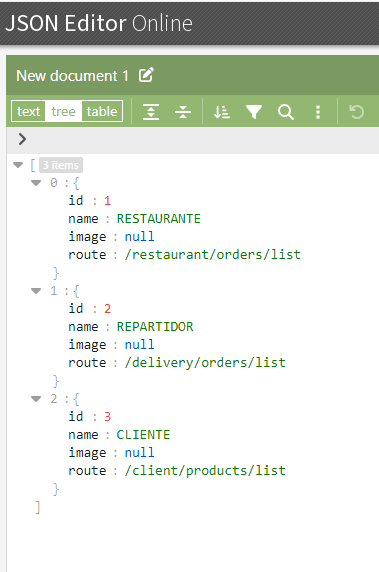
    email = 'luispe@gmail.com

GROUP BY U.id' -- Para probar reemplace  por el correo del usuario

La consulta resulta en:



Y el resultado de la columna ‘roles’ de la anterior consulta arroja algo como:



Una vez probada la consulta, reemplácela en los métodos ‘finById’ y ‘finByEmail’ del archivo “models/user.js” en el servidor Node.js:

User.findById = (id, result) => {

    const sql = `SELECT

        U.id,

        U.email,

        U.name,

        U.lastname,

        U.phone,

        U.image,

        U.password,

        JSON\_ARRAYAGG(

            JSON\_OBJECT(

            'id', CONVERT(R.id, char),

            'name', R.name,

            'image', R.image,

            'route', R.route

            )

        ) AS roles

        FROM users AS U

        INNER JOIN user\_has\_roles AS UHR

            ON UHR.id\_user = U.id

        INNER JOIN roles AS R

            ON UHR.id\_rol = r.id

        WHERE

            U.id =  ?

        GROUP BY U.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

        U.id,

        U.email,

        U.name,

        U.lastname,

        U.phone,

        U.image,

        U.password,

        JSON\_ARRAYAGG(

            JSON\_OBJECT(

            'id', CONVERT(R.id, char),

            'name', R.name,

            'image', R.image,

            'route', R.route

            )

        ) AS roles

        FROM users AS U

        INNER JOIN user\_has\_roles AS UHR

            ON UHR.id\_user = U.id

        INNER JOIN roles AS R

            ON UHR.id\_rol = r.id

        WHERE

            email =  ?

        GROUP BY U.id`;

    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]);

            }

        }

    );

}

En “controllers/usersController.js”, en la data del método “user.findByEmail()”, incluya el objeto json agregado a la consulta:

                const data = {

                    id: myUser.id,

                    email: myUser.email,

                    name: myUser.name,

                    lastname: myUser.lastname,

                    image: myUser.image,

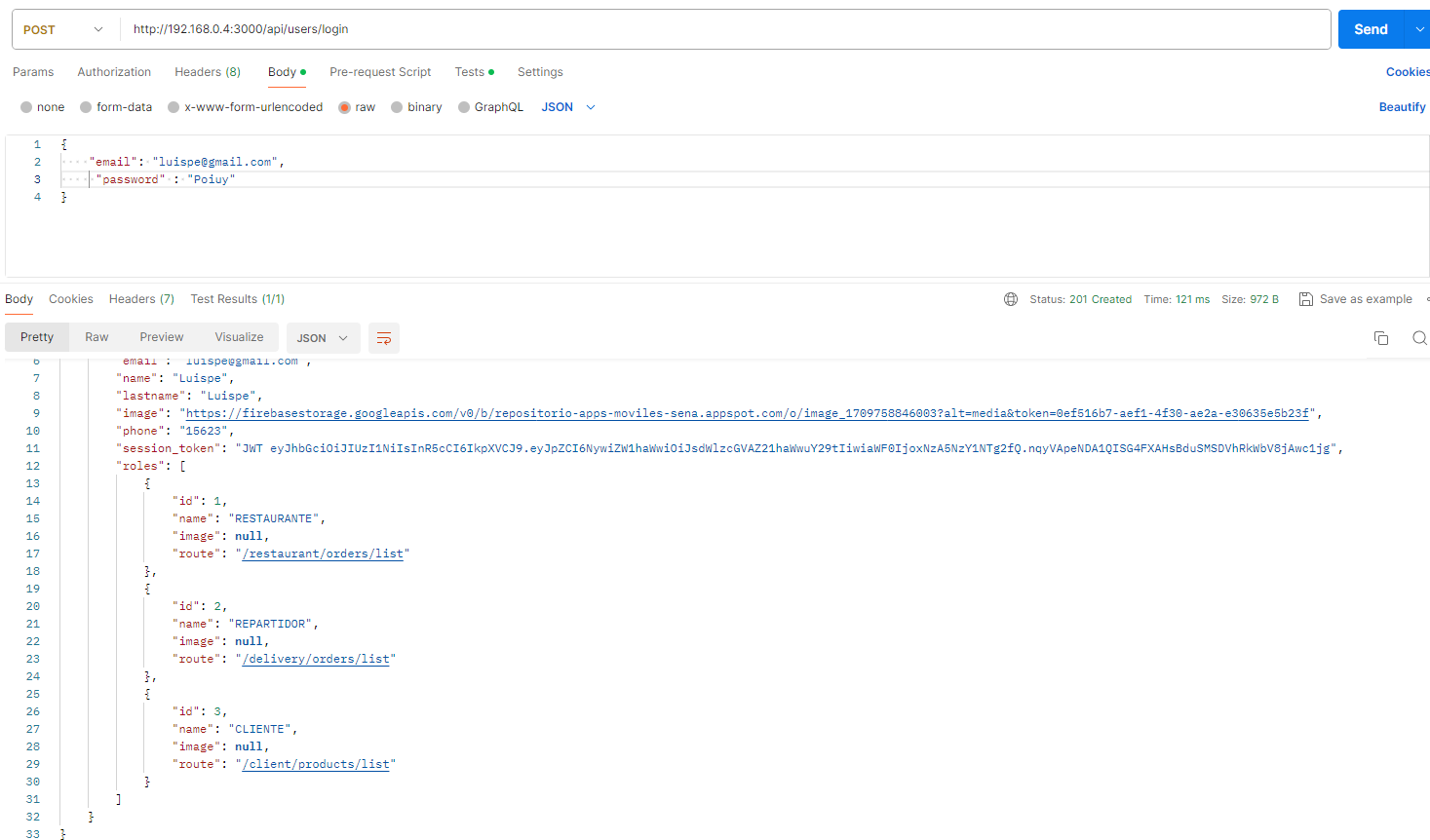
                    phone: myUser.phone,

                    session\_token: `JWT ${token}`,

                    roles: JSON.parse(myUser.roles)

                }

Antes de continuar verifique el servidor y pruebe los cambios realizados usando PostMan:



## 

## Desplegar los roles en la App

### Agregar los roles a la entidad

Se crea una nueva entidad llamada ‘rol’. Crear un archivo “../Domanin/entities/Rol.tsx”:

export interface Rol {

    id: string,

    name: string,

    image: string,

    route: string

}

En “../Domanin/entities/User.tsx”:

import { Rol } from "./Rol";

export interface User {

    id?: string;

    name: string;

    lastname: string;

    phone: string;

    email: string;

    password: string;

    confirmPassword: string;

    session\_token?: string;

    roles?: Rol[];

}

export interface User {

    id?: string;

    name: string;

    lastname: string;

    phone: string;

    email: string;

    password: string;

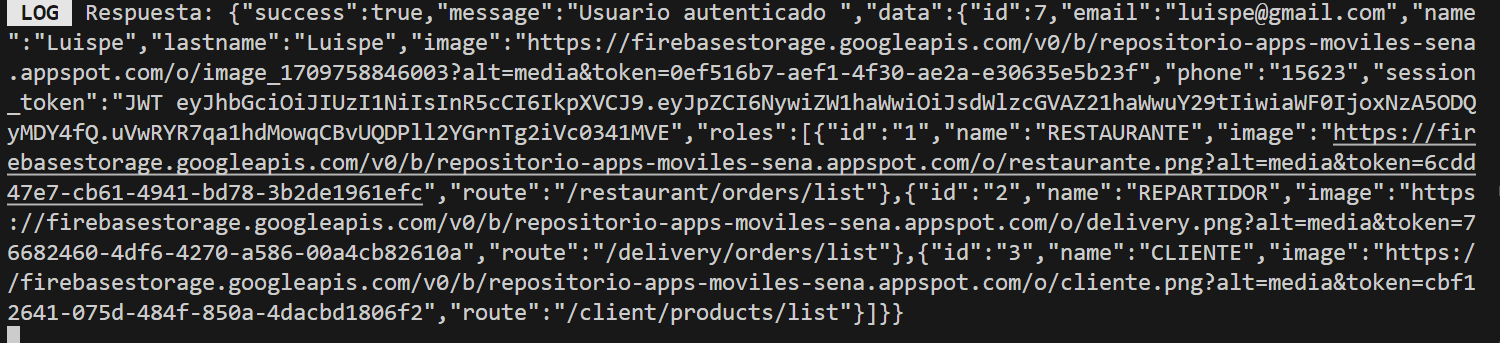
    confirmPassword: string;

    session\_token?: string;

    roles?: Rol[];

}

Levantar el servidor, la aplicación y realizar el inicio de sesión, verifique en la consola que Response incluya la lista de roles:



### Crear una pantalla básica

Ahora es momento de crear una nueva pantalla para desplegar los roles (en caso de que haya más de uno) y poder seleccionar el rol deseado por el usuario.

En el archivo “../views/Home.tsx” se controlará si el usuario tiene más de un rol, para llevarlo a la nueva pantalla:

    useEffect(() => {

        if (user?.id !== null && user?.id !== undefined) {

            if (user.roles?.length! > 1) {

                navigation.replace('RolesScreen');

            }

            else {

                navigation.replace('ProfileInfoScreen');

            }

        }

    }, [user]);

Crear la carpeta “../views/roles” y en ella el archivo “Roles.tsx”:

import React from 'react';

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

import useViewModel from './ViewModel';

export const RolesScreen = () => {

  const { user} = useViewModel();

  return (

    <View>

      <FlatList

      data={user?.roles}

      renderItem={({item}) => <Text>{item.name}</Text>}

      keyExtractor={item => item.id}

      />

    </View>

  );

}

Crear la carpeta “../views/roles” y en ella el archivo “ViewModel.tsx”:

import React from 'react'

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

const RolesViewModel = () => {

  const { user} = useUserLocal();

  return {

    user

  }

}

export default RolesViewModel;

Hay que registrar la nueva pantalla en “App.tsx” (en seguida se presenta el archivo completo hasta el momento):

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';

import { RolesScreen } from './src/Presentation/views/roles/Roles';

export type RootStackParamList = {

    HomeScreen: undefined;

    RegisterScreen: undefined;

    ProfileInfoScreen: undefined;

    RolesScreen: 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: 'Nuevo usuario'

                    }}

                />

                <Stack.Screen

                    name="ProfileInfoScreen"

                    component={ProfileInfoScreen}

                />

                <Stack.Screen

                    name="RolesScreen"

                    component={RolesScreen}

                    options={{

                        headerShown: true,

                        title: 'Selecciona un rol'

                    }}

                />

            </Stack.Navigator>

        </NavigationContainer>

    );

};

export default App;

Ahora se debe visualizar una pantalla con la lista de roles.

### Mejorando la pantalla de roles

Primero se creara un nuevo componente personalizado, pero como solo se usará para los roles, no va a residir en la carpeta “../components” sino en la misma carpeta “../roles” en un nuevo archivo Item.tsx”:

import React from 'react'

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

import { Rol } from '../../../Domain/entities/Rol'

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

interface Props {

    rol: Rol,

    height: number,

    width: number,

}

export const RolesItem = ({ rol, height, width }: Props) => {

    return (

        <TouchableOpacity style={{...styles.container, height: height, width: width}}>

            <View style={styles.imageContainer}>

                <Image style = {styles.image}

                    source={{ uri: rol.image }}

                />

                <View style={ styles.titleContainer}>

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

                </View>

            </View>

        </TouchableOpacity>

    )

}

const styles = StyleSheet.create({

    container: {

        alignSelf: 'center',

        paddingBottom: 20,

        paddingEnd: 7

    },

    imageContainer: {

        flex: 1,

        backgroundColor: 'white',

        borderRadius: 18,

    },

    image: {

        flex: 1,

        resizeMode: 'contain'

    },

    titleContainer: {

        height: 50,

        backgroundColor: MyColors.primary,

        borderBottomLeftRadius: 18,

        borderBottomRightRadius: 18,

        alignItems: 'center',

        justifyContent: 'center'

    },

    title:{

        color: 'white'

    }

})

En el archivo “../views/roles/Roles.tsx”, modificar:

import React from 'react';

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

import useViewModel from './ViewModel';

import { RolesItem } from './Item';

export const RolesScreen = () => {

  const { user} = useViewModel();

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

  //const height = Dimensions.get('window').height;

  return (

    <View>

      <FlatList

      data={user?.roles}

      renderItem={({item}) => <RolesItem rol={item} height={420} width={width - 100} />}

      keyExtractor={item => item.id}

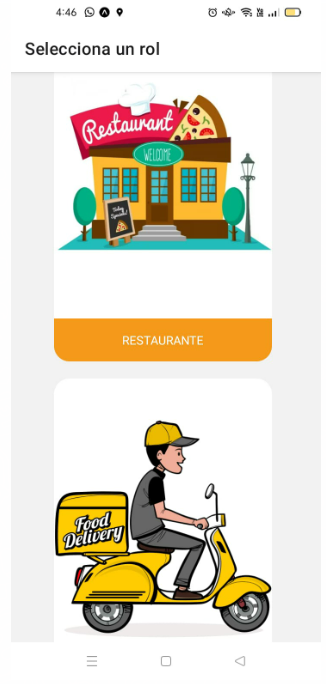
      />

    </View>

  );

}

En el dispositivo se presentará como una lista vertical, así:



### Agregando una barra de navegación

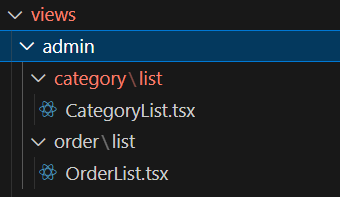
Buscar en Google react native bottom tab navigation o navegue a <https://reactnavigation.org/docs/bottom-tab-navigator/> para buscar la documentación de la barra de navegación a instalar.

En la consola de la App, el comando

npm install @react-navigation/bottom-tabs

Copie el código en ‘API definition’ y cree una carpeta “../Presentation/navigator” y en ella un archivo “AdminTabsNavigator.tsx”:

La idea es crear un sistema de carpetas para organizar de manera jerárquica todas las pantallas. Para el administrador así:



Crear una carpeta “../views/admin” para ubicar todas las pantallas del administrador. En ella una carpeta “category” y en esta otra “list”.

Crear el archivo “../views/admin/category/list/CategoryList.tsx”:

import React from 'react'

export const AdminCategoryListScreen = () => {

  return (

    <div>AdminCategoryListScreen</div>

  )

}

Crear las carpetas y el archivo “../views/admin/order/list/OrderList.tsx”:

import React from 'react'

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

export const AdminOrderListScreen = () => {

  return (

    <View>

        <Text>AdminOrderListScreen</Text>

    </View>

  )

}

Para el Cliente, Las carpetas y el archivo “../views/client/category/list/CategoryList.tsx:

import React from 'react'

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

export const ClientCategoryListScreen = () => {

  return (

    <View>

    <Text>AdminCategoryListScreen</Text>

</View>

  )

}

“../views/client/order/list/OrderList.tsx”:

import React from 'react'

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

export const ClientOrderListScreen = () => {

  return (

    <View>

    <Text>ClientOrderListScreen</Text>

</View>

  )

}

Crear el archivo “../Presentation/navigator/ClientTabsNavigator.tsx”:

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

import { ProfileInfoScreen } from '../views/profile/info/ProfileInfo';

import { ClientCategoryListScreen } from '../views/client/category/list/CategoryList';

import { ClientOrderListScreen } from '../views/client/order/list/OrderList';

const Tab = createBottomTabNavigator();

export function ClientTabsNavigator = () => {

  return (

    <Tab.Navigator>

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

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

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

    </Tab.Navigator>

  );

}

Modifique el archivo “App.tsx” para registrar las nuevas pantallas:

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 { RolesScreen } from './src/Presentation/views/roles/Roles';

import { AdminTabsNavigator } from './src/Presentation/navigator/AdminTabsNavigator';

import { ClientTabsNavigator } from './src/Presentation/navigator/ClientTabsNavigator';

export type RootStackParamList = {

    HomeScreen: undefined;

    RegisterScreen: undefined;

    RolesScreen: undefined;

    AdminTabsNavigator: undefined;

    ClientTabsNavigator: 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: 'Nuevo usuario'

                    }}

                />

                <Stack.Screen

                    name="RolesScreen"

                    component={RolesScreen}

                    options={{

                        headerShown: true,

                        title: 'Selecciona un rol'

                    }}

                />

                <Stack.Screen

                    name="AdminTabsNavigator"

                    component={AdminTabsNavigator}

                />

                <Stack.Screen

                    name="ClientTabsNavigator"

                    component={ClientTabsNavigator}

                />

            </Stack.Navigator>

        </NavigationContainer>

    );

};

export default App;

Estas modificaciones generan un error no critico en el archivo “../profile/info/ProfileInfo.tsx”, en la definición de la interfae Props, modifique la línea así:

interface Props extends StackScreenProps<RootStackParamList> { };

En “../views/register/Register.tsx” modifique el navigation:

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

  const { name, lastname, email, phone, password, image, confirmPassword, loading, errorMessage, user, onChange, register, pickImage, takePhoto } = useViewModel();

  const [modalVisible, setModalVisible] = useState(false);

  //Para saber si la variable ya tiene establecido un valor

  useEffect(() => {

    if (errorMessage !== '')

      ToastAndroid.show(errorMessage, ToastAndroid.LONG)

  }, [errorMessage]);

  useEffect(() => {

    if (user?.id !== null && user?.id !== undefined) {

      navigation.replace('ClientTabsNavigator');

    }

  }, [user]);

  return (

//Etc…

Lo mismo en “../views/home/Home.tsx” modifique el navigation:

//Etc…

    useEffect(() => {

        if (user?.id !== null && user?.id !== undefined) {

            if (user.roles?.length! > 1) {

                navigation.replace('RolesScreen');

            }

            else {

                navigation.replace('ClientTabsNavigator');

            }

        }

    }, [user]);

    return (

//Etc…

El archivo “../views/roles/item.tsx”:

import React from 'react'

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

import { Rol } from '../../../Domain/entities/Rol'

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

import { StackNavigationProp } from '@react-navigation/stack'

import { RootStackParamList } from '../../../../App'

interface Props {

    rol: Rol,

    height: number,

    width: number,

    navigation: StackNavigationProp<RootStackParamList, "RolesScreen", undefined>

}

export const RolesItem = ({ rol, height, width, navigation }: Props) => {

    return (

        <TouchableOpacity

            onPress={() => {

                if(rol.name == 'ADMIN'){

                    navigation.navigate('AdminTabsNavigator')

            }

        else if(rol.name == 'CLIENTE'){

            navigation.navigate('ClientTabsNavigator')

        }

        }}

        style={{...styles.container, height: height, width: width}}>

//Etc…

Modifique el archive “../roles/Roles.tsx”:

import React from 'react';

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

import useViewModel from './ViewModel';

import { RolesItem } from './Item';

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

import { RootStackParamList } from '../../../../App';

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

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

  const { user } = useViewModel();

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

  return (

    <View>

      <FlatList

      data={user?.roles}

      renderItem={({item}) => <RolesItem rol={item} height={420} width={width - 100} navigation={navigation} />}

      keyExtractor={item => item.id}

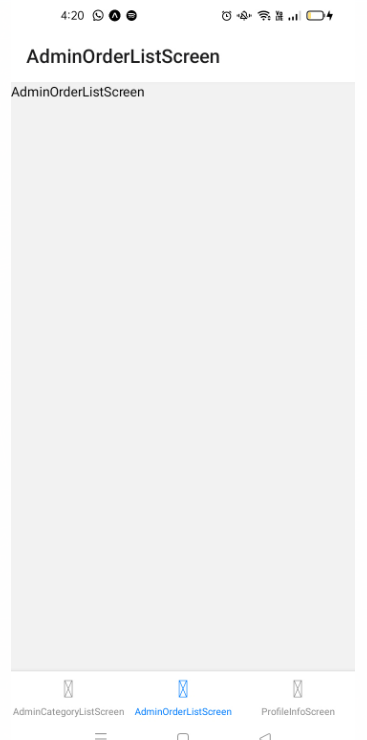
  />

    </View>

  );

}

Recargue la app, inicie una nueva sesión de usuario y pulse el rol de ADMIN[[1]](#footnote-1)



Para terminar esta sección, se agregarán imágenes y títulos en la barra inferior.

En el archivo “../navigator/AdminTabsNavigator.tsx”:

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

import { AdminCategoryListScreen } from '../views/admin/category/list/CategoryList';

import { AdminOrderListScreen } from '../views/admin/order/list/OrderList';

import { ProfileInfoScreen } from '../views/profile/info/ProfileInfo';

import { Image } from 'react-native';

const Tab = createBottomTabNavigator();

export const AdminTabsNavigator = () => {

  return (

    <Tab.Navigator>

      <Tab.Screen

        name="AdminCategoryListScreen"

        component={AdminCategoryListScreen}

        options={{

          title: 'Categorias',

          tabBarLabel: 'Categorias',

          tabBarIcon: ({ color }) => (

            <Image

              source={require('../../../assets/list.png')}

              style={{ width: 25, height: 25 }}

            />

          )

        }}

      />

      <Tab.Screen name="AdminOrderListScreen"

      component={AdminOrderListScreen}

      options={{

        title: 'Pedidos',

        tabBarLabel: 'Pedidos',

        tabBarIcon: ({ color }) => (

          <Image

            source={require('../../../assets/orders.png')}

            style={{ width: 25, height: 25 }}

          />

        )

      }}

      />

      <Tab.Screen name="ProfileInfoScreen"

      component={ProfileInfoScreen}

      options={{

        title: 'Perfil',

        tabBarLabel: 'Perfil',

        tabBarIcon: ({ color }) => (

          <Image

            source={require('../../../assets/user\_menu.png')}

            style={{ width: 25, height: 25 }}

          />

        )

      }}

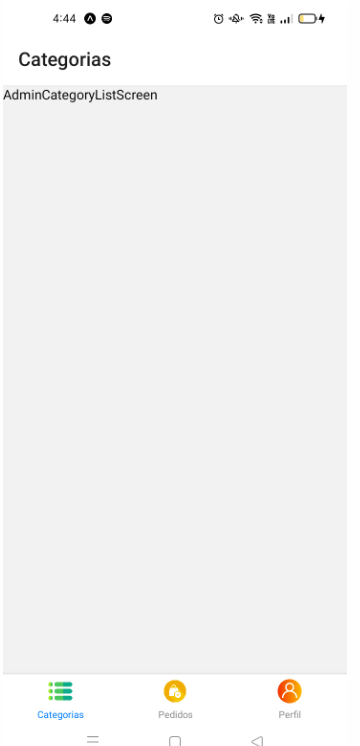
      />

    </Tab.Navigator>

  );

}

Tendremos algo como:



En “../navigator/ClientTabsNavigator.tsx”:

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

import { ProfileInfoScreen } from '../views/profile/info/ProfileInfo';

import { ClientCategoryListScreen } from '../views/client/category/list/CategoryList';

import { ClientOrderListScreen } from '../views/client/order/list/OrderList';

import { Image } from 'react-native';

const Tab = createBottomTabNavigator();

export const ClientTabsNavigator = () => {

  return (

    <Tab.Navigator>

      <Tab.Screen

        name="ClientCategoryListScreen"

        component={ClientCategoryListScreen}

        options={{

          title: 'Lista de Categorias',

          //headerShown: false,  // Para ocultar el header de la pantalla

          tabBarLabel: 'Categorias',

          tabBarIcon: ({ color }) => (

            <Image

              source={require('../../../assets/list.png')}

              style={{ width: 25, height: 25 }}

            />

          )

        }}

      />

      <Tab.Screen name="ClientOrderListScreen"

      component={ClientOrderListScreen}

      options={{

        title: 'Pedidos',

        tabBarLabel: 'Pedidos',

        tabBarIcon: ({ color }) => (

          <Image

            source={require('../../../assets/orders.png')}

            style={{ width: 25, height: 25 }}

          />

        )

      }}

      />

      <Tab.Screen name="ProfileInfoScreen"

      component={ProfileInfoScreen}

      options={{

        title: 'Perfil',

        tabBarLabel: 'Perfil',

        tabBarIcon: ({ color }) => (

          <Image

            source={require('../../../assets/user\_menu.png')}

            style={{ width: 25, height: 25 }}

          />

        )

      }}

      />

    </Tab.Navigator>

  );

}

# PERFILES DE USUARIO

## Ver información del usuario

Modifique “../profile/info/ProfileInfo.tsx:

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

import React from 'react';

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

import { RootStackParamList } from '../../../../../App';

import useViewModel from './ViewModel';

import styles from './Styles';

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

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

export const ProfileInfoScreen = () => {

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

    const { removeSession, user } = useViewModel();

    return (

        <View style={styles.container}>

            <Image

                source={require('../../../../../assets/chef.jpg')}

                style={styles.imageBackground}

            />

            <TouchableOpacity

                style={ styles.logout}

                onPress={() => {

                    removeSession();

                    navigation.replace('HomeScreen');

                }}>

                <Image

                    source={require('../../../../../assets/shutdown.png')}

                    style={styles.logoutImage}

                />

            </TouchableOpacity>

            <View style={styles.logoContainer}>

                <Image

                    source={{ uri: user?.image }}

                    style={styles.logoImage}

                />

            </View>

            <View style={styles.form}>

                <View style={styles.formInfo}>

                    <Image

                        source={require('../../../../../assets/user.png')}

                        style={styles.formImage}

                    />

                    <View style={styles.formContent}>

                        <Text>{user?.name} {user?.lastname}</Text>

                        <Text style={styles.formTextDescription}>Nombre del usuario</Text>

                    </View>

                </View>

                <View style={{ ...styles.formInfo, marginTop: 25 }}>

                    <Image

                        source={require('../../../../../assets/email.png')}

                        style={styles.formImage}

                    />

                    <View style={styles.formContent}>

                        <Text>{user?.name} {user?.email}</Text>

                        <Text style={styles.formTextDescription}>Correo electrónico</Text>

                    </View>

                </View>

                <View style={{ ...styles.formInfo, marginTop: 25, marginBottom: 40 }}>

                    <Image

                        source={require('../../../../../assets/phone.png')}

                        style={styles.formImage}

                    />

                    <View style={styles.formContent}>

                        <Text>{user?.name} {user?.phone}</Text>

                        <Text style={styles.formTextDescription}>Teléfono</Text>

                    </View>

                </View>

                <RoundedButton

                    onPress={() => { }

                    }

                    text={'ACTUALIZAR INFORMACION'}

                />

            </View>

        </View>

    )

}

En “../navigator/ClientTabsNavigator.tsx”:

      <Tab.Screen name="ProfileInfoScreen"

      component={ProfileInfoScreen}

      options={{

        title: 'Perfil',

        tabBarLabel: 'Perfil',

        headerShown: false,

        tabBarIcon: ({ color }) => (

          <Image

            source={require('../../../assets/user\_menu.png')}

            style={{ width: 25, height: 25 }}

          />

        )

      }}

      />

    </Tab.Navigator>

Igual en “../navigator/AdminTabsNavigator.tsx”:

      <Tab.Screen name="ProfileInfoScreen"

      component={ProfileInfoScreen}

      options={{

        title: 'Perfil',

        tabBarLabel: 'Perfil',

        headerShown: false,

        tabBarIcon: ({ color }) => (

          <Image

            source={require('../../../assets/user\_menu.png')}

            style={{ width: 25, height: 25 }}

          />

        )

      }}

En ../profile/info/ViewModel.tsx”:

import React from 'react'

import { RemoveUserLocalUseCase } from '../../../../Domain/useCases/userLocal/RemoveUserLocal';

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

const ProfileInfoViewModel = () => {

  const { user } = useUserLocal();

  const removeSession = async () => {

    await RemoveUserLocalUseCase();

  }

  return {

    removeSession,

    user

  }

}

export default ProfileInfoViewModel;

En “../Domain/enties/User.tsx” verifique que exista el atributo, agréguelo si no está:

 image?: string;

Cree un archivo “../profile/info/Styles.tsx”:

import { StyleSheet } from "react-native";

const ProfileInfoStyles = StyleSheet.create({

    container: {

        flex: 1,

        backgroundColor: 'black',

    },

    imageBackground: {

        width: '100%',

        height: '100%',

        opacity: 0.7,

        bottom: '25%',

    },

    form: {

        width: '100%',

        height: '45%',

        backgroundColor: 'white',

        position: 'absolute',

        bottom: 0,

        borderTopLeftRadius: 40,

        borderTopRightRadius: 40,

        padding: 30,

    },

    formText: {

        fontWeight: 'bold',

        fontSize: 16,

    },

    formInfo: {

        flexDirection: 'row',

        alignItems: 'center',

    },

    formContent:{

        marginLeft: 15,

    },

    formImage: {

        width: 30,

        height: 30,

    },

    formTextDescription:{

        fontSize: 13,

        color: 'gray'

    },

    logoContainer: {

        position: 'absolute',

        alignSelf: 'center',

        top: '14%',

        alignItems: 'center',

    },

    logoImage: {

        width: 180,

        height: 180,

        borderRadius: 100,

        borderColor: 'white',

        borderWidth: 2,

    },

    logoText: {

        color: 'white',

        textAlign: 'center',

        fontSize: 20,

        marginTop: 10,

        fontWeight: 'bold',

    },

    logout:{

        position: 'absolute',

        alignSelf: 'center',

        top: 40,

        right: 20,

    },

    logoutImage:{

        width: 40,

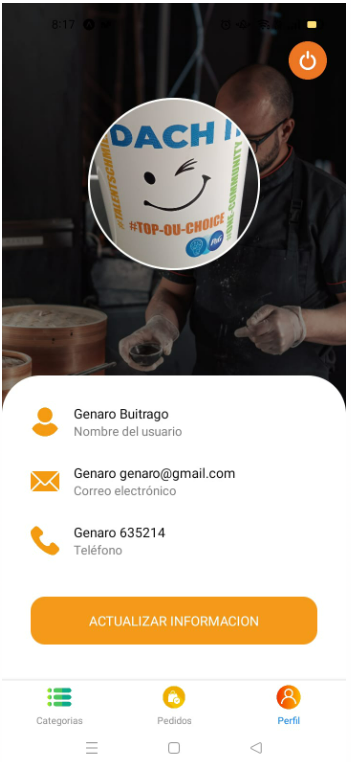
        height: 40,

    }

});

export default ProfileInfoStyles;

Obtendrá una pantalla como:



## Modificar la información del usuario

Crear la carpeta “../views/profile/update/”. Copie los tres archivos “Register.tsx”, “Styles.tsx” y “ViewModel.tsx” desde la carpeta “register”, para modificarlos después.

En la nueva carpeta “../update” cambie el nombre del archivo “Register.tsx” a “ProfileUpdate.tsx” y modifíquelo así:

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

import { View, Text, Image, ScrollView, ToastAndroid, TouchableOpacity, ActivityIndicator } from 'react-native'

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

import useViewModel from './ViewModel';

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

import styles from './Styles';

import ModalPickImage from '../../../components/ModalPickImage';

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

import { RootStackParamList } from '../../../../../App';

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

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

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

  const { name, lastname, phone, image, loading, errorMessage, user, onChange, onChangeInfoUpdate, register, pickImage, takePhoto } = useViewModel();

  const [modalVisible, setModalVisible] = useState(false);

  //Para saber si la variable ya tiene establecido un valor

  useEffect(() => {

    if (errorMessage !== '')

      ToastAndroid.show(errorMessage, ToastAndroid.LONG)

  }, [errorMessage]);

useEffect(() => {

  onChangeInfoUpdate(user?.name!, user?.lastname!, user?.phone!);

}, [user]);

  return (

    <View style={styles.container}>

      <Image

        source={require('../../../../../assets/chef.jpg')}

        style={styles.imageBackground}

      />

      <View style={styles.logoContainer}>

        <TouchableOpacity onPress={() => setModalVisible(true)}>

          {

            image == ''

              ?

              <Image

                source={{ uri: user?.image }}

                style={styles.logoImage}

              />

              :

              <Image

                source={{ uri: image }}

                style={styles.logoImage}

              />

          }

        </TouchableOpacity>

        <Text style={styles.logoText}>SELECCIONA UNA IMAGEN</Text>

      </View>

      <View style={styles.form}>

        <ScrollView>

          <Text style={styles.formText}>ACTUALIZAR</Text>

          <CustomTextInput

            image={require('../../../../../assets/user.png')}

            placeholder='Nombres'

            value={name}

            keyboardType='default'

            property='name'

            onChangeText={onChange}

          />

          <CustomTextInput

            image={require('../../../../../assets/my\_user.png')}

            placeholder='Apellidos'

            value={lastname}

            keyboardType='default'

            property='lastname'

            onChangeText={onChange}

          />

          <CustomTextInput

            image={require('../../../../../assets/phone.png')}

            placeholder='Teléfono'

            value={phone}

            keyboardType='numeric'

            property='phone'

            onChangeText={onChange}

          />

          <View style={{ marginTop: 30 }}>

            <RoundedButton text='CONFIRMAR' onPress={() => register()} />

          </View>

        </ScrollView>

      </View>

      <ModalPickImage

        openGallery={pickImage}

        openCamera={takePhoto}

        setModalUseState={setModalVisible}

        modalUseState={modalVisible}

      />

      {

        loading &&

        <ActivityIndicator

          style={styles.loading}

          size="large"

          color={MyColors.primary}

        />

      }

    </View>

  );

}

“../Styles.tsx”:

import {StyleSheet} from 'react-native';

const ProfileUpdateStyles = StyleSheet.create({

    container: {

      flex: 1,

      backgroundColor: 'black',

    },

    imageBackground: {

      width: '100%',

      height: '100%',

      opacity: 0.7,

      bottom: '25%',

    },

    form: {

      width: '100%',

      height: '55%',

      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: 20,

    },

    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: '10%',

      alignItems: 'center',

    },

    logoImage: {

      width: 100,

      height: 100,

    },

    logoText: {

      color: 'white',

      textAlign: 'center',

      fontSize: 20,

      marginTop: 10,

      fontWeight: 'bold',

    },

    loading:{

      position: 'absolute',

      bottom: 0,

      top: 0,

      left: 0,

      right: 0,

    }

  });

  export default ProfileUpdateStyles;

“../ViewModel.tsx”:

import  { useState } from "react"

import \* as ImagePicker from "expo-image-picker";

import { RegisterWithImageAuthUseCase } from "../../../../Domain/useCases/auth/RegisterWithImageAuth";

import { SaveUserLocalUseCase } from "../../../../Domain/useCases/userLocal/SaveUserLocal";

import { useUserLocal } from "../../../hooks/useUserLocal"

const ProfileUpdateViewModel = () => {

    const [errorMessage, setErrorMessage] = useState('');

    const [values, setValues] = useState({

        name: '',

        lastname: '',

        email: '',

        phone: '',

        image: '',

        password: '',

        confirmPassword: ''

    });

    const [loading, setloading] = useState(false);

    const [file, setFile] = useState<ImagePicker.ImagePickerAsset>();

    const { user, getUserSession } = useUserLocal();

    const pickImage = async () => {

        let result = await ImagePicker.launchImageLibraryAsync({

            mediaTypes: ImagePicker.MediaTypeOptions.All,

            allowsEditing: true,

            quality: 1,

        });

        if (!result.canceled) {

            onChange('image', result.assets[0].uri);

            setFile(result.assets[0]);

        }

    };

    const takePhoto = async () => {

        let result = await ImagePicker.launchCameraAsync({

            mediaTypes: ImagePicker.MediaTypeOptions.All,

            allowsEditing: true,

            quality: 1,

        });

        if (!result.canceled) {

            onChange('image', result.assets[0].uri);

            setFile(result.assets[0]);

        }

    };

    const onChange = (property: string, value: any) => {

        setValues({ ...values, [property]: value });

    }

    const onChangeInfoUpdate = (name: string, lastname: string, phone: string) => {

        setValues({ ...values, name, lastname, phone });

    }

    const register = async () => {

        if (isValidForm()) {

            setloading(true);

            const response = await RegisterWithImageAuthUseCase(values, file!);

            setloading(false);

            console.log('Result' + JSON.stringify(response));

            if(response.success){

                await SaveUserLocalUseCase(response.data);

                getUserSession();

            }

            else{

                setErrorMessage(response.message);

            }

        }

    }

    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;

        }

        if (values.image === '') {

            setErrorMessage('Seleccione una imagen de perfil');

            return false;

        }

        return true;

    }

    return {

        ...values,

        onChange,

        register,

        pickImage,

        takePhoto,

        onChangeInfoUpdate,

        errorMessage,

        loading,

        user

    }

}

export default ProfileUpdateViewModel;

En “App.tsx” registrar la nueva pantalla:

import { ProfileUpdateScreen } from './src/Presentation/views/profile/update/ProfileUpdate';

export type RootStackParamList = {

    HomeScreen: undefined,

    RegisterScreen: undefined,

    RolesScreen: undefined,

    AdminTabsNavigator: undefined,

    ClientTabsNavigator: undefined,

    ProfileUpdateScreen: undefined

};

Y al final …

                <Stack.Screen

                    name="ProfileUpdateScreen"

                    component={ProfileUpdateScreen}

                    options={{

                        headerShown: true,

                        title: 'Actualizar usuario'

                    }}

                />

            </Stack.Navigator>

        </NavigationContainer>

    );

};

Para activar el botón de la pantalla ‘profile’ y que lleve a la nueva pantalla, en “../profile/info/ProfileInfo.tsx”:

                <RoundedButton

                    onPress={() => {

                        navigation.navigate('ProfileUpdateScreen');

                     }

                    }

                    text={'ACTUALIZAR INFORMACION'}

                />

            </View>

        </View>

    )

}

## Agregar las funcionalidades al back-end

En el servidor node.js, en el archivo “models/user.js” se crean dos métodos para actualizar datos:

User.update = async (user, result) => {

    const sql = `UPDATE users SET

        name = ?,

        lastname = ?,

        phone = ?,

        image = ?,

        updated\_at = ?

        WHERE id = ?`;

    db.query(sql,

        [

            user.name,

            user.lastname,

            user.phone,

            user.image,

            new Date(),

            user.id

        ], (err, res) => {

            if (err) {

                console.log('Error al actulizar el usuario: ', err);

                result(err, null);

            }

            else {

                console.log('Usuario actualizado: ', user.id);

                result(null, user.id);

            }

        }

    );

}

User.updateWithoutImage = async (user, result) => {

    const sql = `UPDATE users SET

        name = ?,

        lastname = ?,

        phone = ?,

        updated\_at = ?

        WHERE id = ?`;

    db.query(sql,

        [

            user.name,

            user.lastname,

            user.phone,

            new Date(),

            user.id

        ], (err, res) => {

            if (err) {

                console.log('Error al actulizar el usuario: ', err);

                result(err, null);

            }

            else {

                console.log('Usuario actualizado: ', user.id);

                result(null, user.id);

            }

        }

    );

}

module.exports = User;

En “controllers/usersController.js”:

    async updateWithImage(req, res) {

        const user = JSON.parse(req.body.user);

        const files = req.files;

        if(files.length > 0){

            const path = `image\_${Date.now()}`;

            const url = await storage(files[0], path);

            if(url != undefined && url != null){

                user.image = url;

            }

        }

        User.update(user, (err, data) => {

            if (err) {

                return res.status(501).json({

                    success: false,

                    message: 'Error al actualizar el usuario',

                    error: err

                });

            }

            return res.status(201).json({

                success: true,

                message: 'Usuario actualizado ',

                data: user

            });

        });

    },

    async updateWithoutImage(req, res) {

        const user = req.body;

        const files = req.files;

        User.updateWithoutImage(user, (err, data) => {

            if (err) {

                return res.status(501).json({

                    success: false,

                    message: 'Error al actualizar el usuario',

                    error: err

                });

            }

            return res.status(201).json({

                success: true,

                message: 'Usuario actualizado ',

                data: user

            });

        });

    },

}

En “routes/userRoutes.js” agregar las dos rutas de actualización:

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

module.exports = (app, upload) => {

    app.post('/api/users/create', userController.register);

    app.post('/api/users/createWithImage', upload.array('image', 1), userController.registerWithImage);

    app.post('/api/users/login', userController.login);

    app.put('/api/users/update', upload.array('image', 1), userController.updateWithoutImage);

    app.put('/api/users/updateWithoutImage',  userController.updateWithoutImage);

}

Levante el servidor y corrija errores de digitación si los hay.

En la App, definir los métodos en la capa de dominio. Cree el archivo “../Domain/repositories/UserRepository.tsx”:

import { ResponseApiDelivery } from "../../Data/sources/remote/models/ResponseApiDelivery"

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

import \* as ImagePicker from "expo-image-picker";

export interface UserRepository {

    update: (user: User) => Promise<ResponseApiDelivery>;

    updateWithImage(user: User, file: ImagePicker.ImageInfo): Promise<ResponseApiDelivery>;

}

Implementar los métodos en repository. Crear un archivo “Data/repositories/UserRepository.tsx”:

import { Axios, AxiosError } from 'axios';

import { User } from '../../Domain/entities/User';

import { UserRepository } from '../../Domain/repositories/UserRepository';

import { ApiDelivery, ApiDeliveryForImage } from '../sources/remote/api/ApiDelivery';

import { ResponseApiDelivery } from '../sources/remote/models/ResponseApiDelivery';

import mime from 'mime';

import { ImageInfo } from 'expo-image-picker';

export class UserRepositoryImpl implements UserRepository {

    async update(user: User): Promise<ResponseApiDelivery> {

        try {

            const response = await ApiDelivery.put<ResponseApiDelivery>('users/updateWithoutImage', 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 updateWithImage(user: User, file: ImageInfo): Promise<ResponseApiDelivery> {

        try {

            let data = new FormData();

            data.append('image', { //'image' is the same name of the field in the API routes

                // @ts-ignore

                uri: file.uri,

                name: file.uri.split('/').pop(),

                type: mime.getType(file.uri)!

            } as any);

            data.append('user', JSON.stringify(user)); //'user' is the same name of the field in the API routes

            const response = await ApiDeliveryForImage.put<ResponseApiDelivery>('/users/update', data);

            // '/users/createWithImage' is the route in the API

            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);

        }

    }

}

Definir los casos de uso. Crear la carpeta “Domain/useCases/user”. En ella el archivo “UpdateUser.tsx”:

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

import { UserRepositoryImpl } from "../../../Data/repositories/UserRepository";

const { update} = new UserRepositoryImpl();

export const UpdateUserUseCase = async (user: User) => {

    return await update(user);

}

En la misma carpeta el archivo “../UpdateWithImageUser.tsx”:

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

import { UserRepositoryImpl } from "../../../Data/repositories/UserRepository";

import \* as ImagePicker from 'expo-image-picker';

const { updateWithImage} = new UserRepositoryImpl();

export const UpdateWithImageUserUseCase = async (user: User, file: ImagePicker.ImageInfo) => {

    return await updateWithImage(user, file);

}

Modificar el método onPress del botón al final de “views/profile/ProfileInfo.tsx”:

               <RoundedButton

                    onPress={() => {

                        navigation.navigate('ProfileUpdateScreen', { user: user! });

                     }

                    }

                    text={'ACTUALIZAR INFORMACION'}

                />

Modifique el archivo “profile/update/ProfileUpdate.tsx” así:

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

import { View, Text, Image, ScrollView, ToastAndroid, TouchableOpacity, ActivityIndicator } from 'react-native'

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

import useViewModel from './ViewModel';

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

import styles from './Styles';

import ModalPickImage from '../../../components/ModalPickImage';

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

import { RootStackParamList } from '../../../../../App';

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

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

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

  const { user } = route.params;

  const { name, lastname, phone, image, loading, errorMessage, onChange, onChangeInfoUpdate, update, pickImage, takePhoto } = useViewModel(user);

  const [modalVisible, setModalVisible] = useState(false);

  //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}>

        <TouchableOpacity onPress={() => setModalVisible(true)}>

          {

            image == ''

              ?

              <Image

                source={{ uri: user?.image }}

                style={styles.logoImage}

              />

              :

              <Image

                source={{ uri: image }}

                style={styles.logoImage}

              />

          }

        </TouchableOpacity>

        <Text style={styles.logoText}>SELECCIONA UNA IMAGEN</Text>

      </View>

      <View style={styles.form}>

        <ScrollView>

          <Text style={styles.formText}>ACTUALIZAR</Text>

          <CustomTextInput

            image={require('../../../../../assets/user.png')}

            placeholder='Nombres'

            value={name}

            keyboardType='default'

            property='name'

            onChangeText={onChange}

          />

          <CustomTextInput

            image={require('../../../../../assets/my\_user.png')}

            placeholder='Apellidos'

            value={lastname}

            keyboardType='default'

            property='lastname'

            onChangeText={onChange}

          />

          <CustomTextInput

            image={require('../../../../../assets/phone.png')}

            placeholder='Teléfono'

            value={phone}

            keyboardType='numeric'

            property='phone'

            onChangeText={onChange}

          />

          <View style={{ marginTop: 30 }}>

            <RoundedButton text='CONFIRMAR' onPress={() => update()} />

          </View>

        </ScrollView>

      </View>

      <ModalPickImage

        openGallery={pickImage}

        openCamera={takePhoto}

        setModalUseState={setModalVisible}

        modalUseState={modalVisible}

      />

      {

        loading &&

        <ActivityIndicator

          style={styles.loading}

          size="large"

          color={MyColors.primary}

        />

      }

    </View>

  );

}

Por último, implementarlo en “Presentation/views/profile/update/ViewModel.tsx”:

import  { useState } from "react"

import \* as ImagePicker from "expo-image-picker";

import { SaveUserLocalUseCase } from "../../../../Domain/useCases/userLocal/SaveUserLocal";

import { useUserLocal } from "../../../hooks/useUserLocal"

import { RegisterWithImageAuthUseCase } from "../../../../Domain/useCases/auth/RegisterWithImageAuth";

import { UpdateUserUseCase } from "../../../../Domain/useCases/user/UpdateUser";

import { UpdateWithImageUserUseCase } from "../../../../Domain/useCases/user/UpdateWithImageUser";

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

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

const ProfileUpdateViewModel = (user: User) => {

    const [errorMessage, setErrorMessage] = useState('');

    const [values, setValues] = useState(user);

    const [loading, setloading] = useState(false);

    const [file, setFile] = useState<ImagePicker.ImagePickerAsset>();

    const { getUserSession } = useUserLocal();

    const pickImage = async () => {

        let result = await ImagePicker.launchImageLibraryAsync({

            mediaTypes: ImagePicker.MediaTypeOptions.All,

            allowsEditing: true,

            quality: 1,

        });

        if (!result.canceled) {

            onChange('image', result.assets[0].uri);

            setFile(result.assets[0]);

        }

    };

    const takePhoto = async () => {

        let result = await ImagePicker.launchCameraAsync({

            mediaTypes: ImagePicker.MediaTypeOptions.All,

            allowsEditing: true,

            quality: 1,

        });

        if (!result.canceled) {

            onChange('image', result.assets[0].uri);

            setFile(result.assets[0]);

        }

    };

    const onChange = (property: string, value: any) => {

        setValues({ ...values, [property]: value });

    }

    const onChangeInfoUpdate = (name: string, lastname: string, phone: string) => {

        setValues({ ...values, name, lastname, phone });

    }

    const update = async () => {

        if (isValidForm()) {

            setloading(true);

            let response = {} as ResponseApiDelivery

            if (values.image?.includes('https://')) {

                response = await UpdateUserUseCase(values);

            }

            else{

                response = await UpdateWithImageUserUseCase(values, file!);

            }

            setloading(false);

            console.log('Result' + JSON.stringify(response));

            if(response.success){

                await SaveUserLocalUseCase(response.data);

                getUserSession();

            }

            else{

                setErrorMessage(response.message);

            }

        }

        else {  //Agregado para depuracion

            console.log('Formulario no valido');

        }

    }

    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.phone === '') {

            setErrorMessage('El teléfono es requerido');

            return false;

        }

        return true;

    }

    return {

        ...values,

        onChange,

        update,

        pickImage,

        takePhoto,

        onChangeInfoUpdate,

        errorMessage,

        loading,

        user

    }

}

export default ProfileUpdateViewModel;

## Creando variables de contexto

Para mantener datos que se puedan utilizar en otras partes de la App, React Native hace uso de “Context API”, que, a diferencia de los Props, permite propagar información entre componentes a nivel de toda la App (<https://legacy.reactjs.org/docs/context.html>).

El contexto se creará en la capa de presentación. Crear la carpeta “../Presentation/context” y en ella el archivo “UserContext.tsx”:

import React, { createContext, useEffect, useState} from "react";

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

import { SaveUserLocalUseCase } from "../../Domain/useCases/userLocal/SaveUserLocal";

import { GetUserLocalUseCase } from "../../Domain/useCases/userLocal/GetUserLocal";

import { RemoveUserLocalUseCase } from "../../Domain/useCases/userLocal/RemoveUserLocal";

export const UserInicialState: User ={

    id:             '',

    name:           '',

    lastname:       '',

    phone:          '',

    email:          '',

    password:       '',

    confirmPassword:'',

    image:          '',

    session\_token:  '',

    roles:          [],

}

export interface UserContextProps {

    user: User;

    saveUserSesion: (user: User) => Promise<void>;

    getUserSesion: () => Promise<void>;

    removeUserSesion: () => Promise<void>;

}

export const UserContext = createContext({} as UserContextProps);

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

    const [user, setUser] = useState(UserInicialState);

    useEffect(() => {

        getUserSesion();

    }, [])

    const saveUserSesion = async (user: User) => {

        await SaveUserLocalUseCase(user);

        setUser(user);

    }

    const getUserSesion = async () => {

        const user = await GetUserLocalUseCase();

        setUser(user);

    }

    const removeUserSesion = async () => {

        await RemoveUserLocalUseCase();

        setUser(UserInicialState);

    }

    return (

        <UserContext.Provider value={{

            user,

            saveUserSesion,

            getUserSesion,

            removeUserSesion,

        }}>

{children}

        </UserContext.Provider>

    )

}

Ahora se implementará en las pantallas donde se requiera propagar los datos del usuario.

En “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 { RolesScreen } from './src/Presentation/views/roles/Roles';

import { AdminTabsNavigator } from './src/Presentation/navigator/AdminTabsNavigator';

import { ClientTabsNavigator } from './src/Presentation/navigator/ClientTabsNavigator';

import { ProfileUpdateScreen } from './src/Presentation/views/profile/update/ProfileUpdate';

import { User } from './src/Domain/entities/User';

import { UserProvider } from './src/Presentation/context/UserContext';

export type RootStackParamList = {

    HomeScreen: undefined,

    RegisterScreen: undefined,

    RolesScreen: undefined,

    AdminTabsNavigator: undefined,

    ClientTabsNavigator: undefined,

    ProfileUpdateScreen: { user: User }

};

const Stack = createNativeStackNavigator<RootStackParamList>();

const App = () => {

    return (

        <NavigationContainer>

            <UserState>

                <Stack.Navigator screenOptions={

                    {

                        headerShown: false

                    }

                }>

                    <Stack.Screen

                        name="HomeScreen"

                        component={HomeScreen}

                    />

                    <Stack.Screen

                        name="RegisterScreen"

                        component={RegisterScreen}

                        options={{

                            headerShown: true,

                            title: 'Nuevo usuario'

                        }}

                    />

                    <Stack.Screen

                        name="RolesScreen"

                        component={RolesScreen}

                        options={{

                            headerShown: true,

                            title: 'Selecciona un rol'

                        }}

                    />

                    <Stack.Screen

                        name="AdminTabsNavigator"

                        component={AdminTabsNavigator}

                    />

                    <Stack.Screen

                        name="ClientTabsNavigator"

                        component={ClientTabsNavigator}

                    />

                    <Stack.Screen

                        name="ProfileUpdateScreen"

                        component={ProfileUpdateScreen}

                        options={{

                            headerShown: true,

                            title: 'Actualizar usuario'

                        }}

                    />

                </Stack.Navigator>

            </UserState>

        </NavigationContainer>

    );

};

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

    return (

        <UserProvider>

            {children}

        </UserProvider>

    )

}

export default App;

Observe que <UserState> envuelve todas las pantallas en la pila de navegación, lo que lo hace global a todas ellas. Pruebe la App y si esta funcional, cierre la sesión para continuar.

Para usarlo, empecemos por la pantalla Home.

En “../views/home/ViewModel.tsx”:

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

import { LoginAuthUseCase } from '../../../Domain/useCases/auth/LoginAuth';

//import { SaveUserUseCase } from '../../../Domain/useCases/userLocal/SaveUserLocal';

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

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

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

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

const HomeViewModel = () => {

    const [errorMessage, setErrorMessage] = useState('');

    const [values, setValues] = useState(

        {

            email: '',

            password: ''

        }

    );

    //const { user, getUserSession } = useUserLocal(); // Se cambia a:

    const { user, saveUserSesion} = useContext(UserContext);

    console.log('Usuario en sesión: ' + JSON.stringify(user));

    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 {

                saveUserSesion(response.data);

            }

        }

    };

    const isValidForm = () => {

//ETC...

En “../views/home/Home.tsx” agregue una comprobación:

    useEffect(() => {

        if (user?.id !== null && user?.id !== undefined && user?.id !== '') {

            if (user.roles?.length! > 1) {

                navigation.replace('RolesScreen');

            }

            else {

                navigation.replace('ClientTabsNavigator');

            }

        }

    }, [user]);

    return (

Ahora en la pantalla de perfil. En “..views/profile/info/ViewModel.tsx” cambiar a:

import React, { useContext} from 'react'

import { RemoveUserLocalUseCase } from '../../../../Domain/useCases/userLocal/RemoveUserLocal';

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

const ProfileInfoViewModel = () => {

  const { user, removeUserSesion } = useContext( UserContext);

  return {

    removeUserSesion,

    user

  }

}

export default ProfileInfoViewModel;

En “..views/profile/info/ProfileInfo.tsx” cambiar a:

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

import React, { useEffect } from 'react';

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

import { RootStackParamList } from '../../../../../App';

import useViewModel from './ViewModel';

import styles from './Styles';

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

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

export const ProfileInfoScreen = () => {

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

    const { user, removeUserSesion } = useViewModel();

    useEffect(() => {

        if (user.id === ''){

            navigation.replace('HomeScreen');

        }

    }, [user]);

    return (

        <View style={styles.container}>

            <Image

                source={require('../../../../../assets/chef.jpg')}

                style={styles.imageBackground}

            />

            <TouchableOpacity

                style={ styles.logout}

                onPress={() => {

                    removeUserSesion();

                }}>

                <Image

//ETC...

Y en el mismo archivo, en el <View> que muestra la imagen, cambiar a:

            <View style={styles.logoContainer}>

                {

                    user?.image !== ''

                    &&

                    <Image

                        source={{ uri: user?.image }}

                        style={styles.logoImage}

                    />

                }

            </View>

En “..views/profile/update/ViewModel.tsx”:

Crear la nueva constante, con sus imports:

mport  { useState, useContext } from "react"

//import { ApiDelivery } from "../../../Data/sources/remote/api/ApiDelivery";

//import { RegisterAuthUseCase } from "../../../Domain/useCases/auth/RegisterAuth";

import \* as ImagePicker from "expo-image-picker";

//import { RegisterWithImageAuthUseCase } from "../../../../Domain/useCases/auth/RegisterWithImageAuth";

import { SaveUserLocalUseCase } from "../../../../Domain/useCases/userLocal/SaveUserLocal";

import { useUserLocal } from "../../../hooks/useUserLocal"

import { RegisterWithImageAuthUseCase } from "../../../../Domain/useCases/auth/RegisterWithImageAuth";

import { UpdateUserUseCase } from "../../../../Domain/useCases/user/UpdateUser";

import { UpdateWithImageUserUseCase } from "../../../../Domain/useCases/user/UpdateWithImageUser";

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

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

import { UserContext } from "../../../context/UserContext";

const ProfileUpdateViewModel = (user: User) => {

    const [errorMessage, setErrorMessage] = useState('');

    const [values, setValues] = useState(user);

    const [loading, setloading] = useState(false);

    const [file, setFile] = useState<ImagePicker.ImagePickerAsset>();

    const { getUserSession } = useUserLocal();

    const { saveUserSesion} = useContext(UserContext);

En el método Update del mismo archivo:

    const update = async () => {

        if (isValidForm()) {

            setloading(true);

            let response = {} as ResponseApiDelivery

            if (values.image?.includes('https://')) {

                response = await UpdateUserUseCase(values);

            }

            else{

                response = await UpdateWithImageUserUseCase(values, file!);

            }

            setloading(false);

            console.log('Result' + JSON.stringify(response));

            if(response.success){

                saveUserSesion(response.data);

            }

            else{

                setErrorMessage(response.message);

            }

        }

        else {  //Agregado para depuracion

            console.log('Formulario no valido');

        }

    }

Ahora, verificar si se pasan los datos al navegar entre pantallas. Reinicie sesión,

### Mensaje informativo

Por último, para desplegar en la pantalla un mensaje de éxito al actualizar, en “../views/profile/update/viewModel.tsx”:

Al inicio

const ProfileUpdateViewModel = (user: User) => {

    const [successMessage, setSuccessMessage] = useState('');

//ETC...

Dentro del método Update, si la actualización fue exitosa:

            if(response.success){

                saveUserSesion(response.data);

                setSuccessMessage(response.message); //Agregado para mensaje al usuario

            }

            else{

                setErrorMessage(response.message);

            }

En el Return:

    return {

        ...values,

        onChange,

        update,

        pickImage,

        takePhoto,

        onChangeInfoUpdate,

        errorMessage,

        successMessage, //Agregado para mensaje al usuario

        loading,

        user

    }

En “../views/profile/update/ProfileUpdate.tsx”:

Declararlo al inicio:

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

  const { user } = route.params;

  const { name, lastname, phone, image, loading, errorMessage, **successMessage**, onChange, onChangeInfoUpdate, update, pickImage, takePhoto } = useViewModel(user);

  const [modalVisible, setModalVisible] = useState(false);

Crear un useEffect:

 useEffect(() => {

    if (errorMessage !== '')

      ToastAndroid.show(errorMessage, ToastAndroid.LONG)

  }, [errorMessage]);

  //Agregado para mensaje al usuario

  useEffect(() => {

    if (successMessage !== '')

      ToastAndroid.show(successMessage, ToastAndroid.LONG)

  }, [successMessage]);

  return (

Probar.

## Enviar Peticiones seguras

Para eso se implementará el uso del Token de sesión creado anteriormente. En el servidor, en “routes/userRouter.js” se protegen las dos últimas rutas:

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

const passport = require('passport');

module.exports = (app, upload) => {

    app.post('/api/users/create', userController.register);

    app.post('/api/users/createWithImage', upload.array('image', 1), userController.registerWithImage);

    app.post('/api/users/login', userController.login);

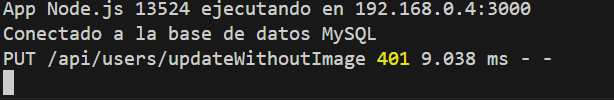
    app.put('/api/users/update', passport.authenticate('jwt', {session: false}), upload.array('image', 1), userController.updateWithImage);

    app.put('/api/users/updateWithoutImage', passport.authenticate('jwt', {session: false}),  userController.updateWithoutImage);

}

En la App, en el archivo “../Data/sources/remote/api/ApiDelivery.tsx”:

Si reinicia el servidor y desde la App intenta una actualización, obtendrá un código 401(Unauthorized).



En la App se debe configurar los Headers del HTTP para incluir el token de sesión. Para eso se implementan ‘interceptores’ en “../Data/sources/remote/api/ApiDelivery.tsx”:

import axios from 'axios';

import { LocalStorage } from '../../local/LocalStorage';

import { User } from '../../../../Domain/entities/User';

const ApiDelivery = axios.create({

    //baseURL: 'http://192.168.0.4:3000/api',

    baseURL: 'http://192.168.0.4:3000/api',

    headers: {

        'Content-Type': 'application/json',

    }

});

const ApiDeliveryForImage = axios.create({

    baseURL: 'http://192.168.0.4:3000/api',

    headers: {

        'Content-Type': 'multipart/form-data',

        'Accept': 'application/json',

    }

});

//INTERCEPTORES: Se ejecutan antes de que la petición sea enviada al servidor

ApiDelivery.interceptors.request.use(

    async (config) => {

        const data = await LocalStorage().getItem('user');

        if (data){

            const user: User = JSON.parse(data as any);

            config.headers!['Authorization'] = user?.session\_token;

            //Si la anterior linea no funciona, probar con una de las siguientes

            // (config.headers as AxiosHeaders).set('Authorization'), `${user?.session\_token!}`);

            // config.headers.set('Authorization', `${user?.session\_token!}`);

        }

        return config;

    }

);

ApiDeliveryForImage.interceptors.request.use(

    async (config) => {

        const data = await LocalStorage().getItem('user');

        if (data){

            const user: User = JSON.parse(data as any);

            config.headers!['Authorization'] = user?.session\_token;

        }

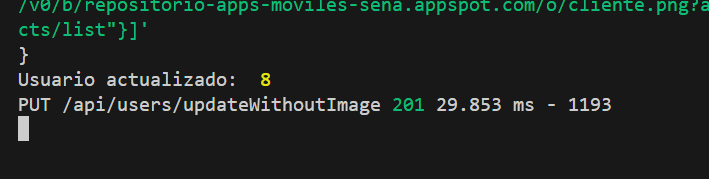
        return config;

    }

);

export { ApiDelivery, ApiDeliveryForImage}

Ahora, si se realiza un cambio, se tiene éxito puesto que el servidor ya está incluyendo la autorización con el token, dentro del header del HTTP.



1. Debe cambiar el nombre del rol en la correspondiente tabla de la BD a ‘ADMIN’ y reiniciar el servidor, la App y hacer login de nuevo, para que se carguen los últimos valores de la BD. [↑](#footnote-ref-1)