



SERENE ELDERS APP

MANUAL TECNICO



UNIVERSIDAD DON BOSCO

Desarrollo Diseño y Programación de Software Multiplataforma

DPS901 G01T



TRABAJO COLABORATIVO

Grupo de trabajo

INTEGRANTES:

Apellidos	Nombres	Carné
Robles Rodas,	Melvin Eduardo	RR191220
Lue Valdez,	Juan Jose	LV231699
Díaz Merino,	Jesus Alexander	DM171988
Fuentes Guardado,	Daniela Alejandra	FG210531
Abarca Flores,	Luis Ángel	AF231735

DOCENTE: Alexander Alberto Siguenza Campos

FECHA DE PRESENTACIÓN: 09/06/2024

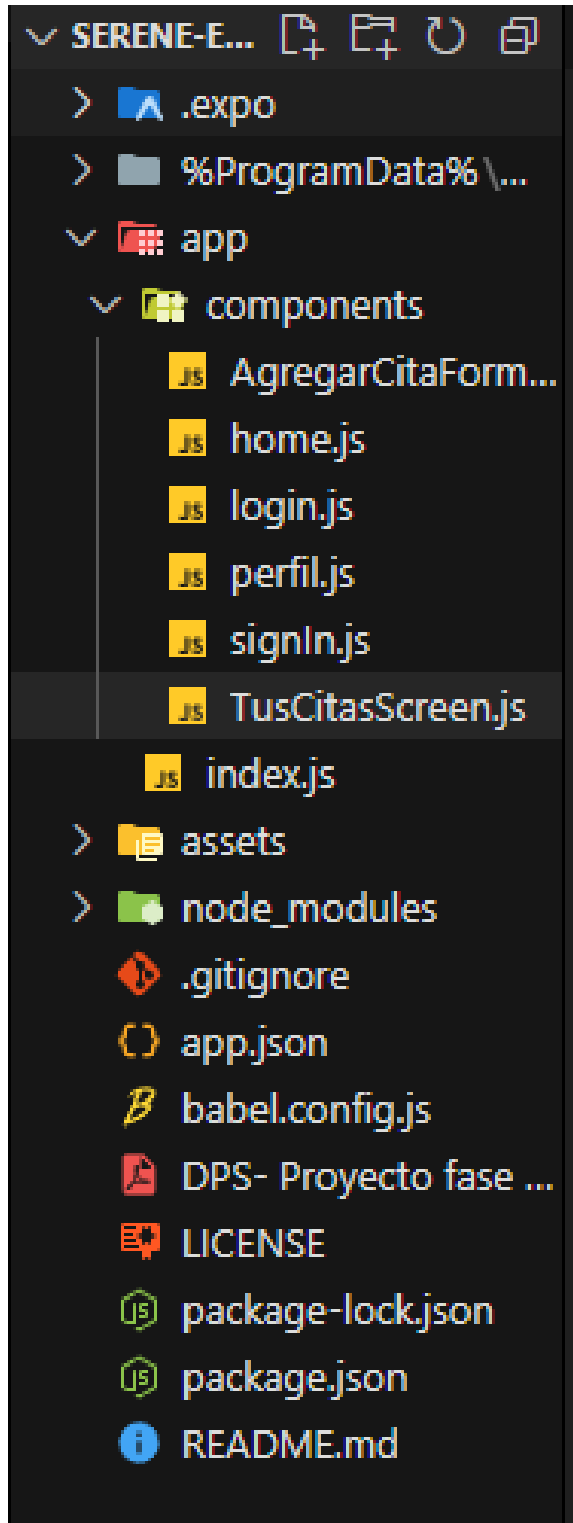
INTRODUCCION

Como principal objetivo del presente manual técnico de la aplicación "Serene Elders" es proporcionar una descripción de forma detallada el desarrollo que tuvo la implementación la aplicación móvil para la gestión de citas medicas. Durante el desarrollo de este manual, se lograra visualizar herramientas de acceso como ver las citas programadas, los pacientes y los doctores activos, entre otras funciones.

Contenido

UNIVERSIDAD DON BOSCO	2
DPS901 G01T.....	2
ESTRUCTURA DEL PROYECTO	5
ARCHIVOS JS PARA VENTANAS DE APLICACIÓN MOVIL	6
INDEX.JS:.....	6
SingIn.Js.....	9
LogIn:	12
HOME.JS:	15
PACIENTES.JS	18
PERFIL.JS	20
TusCitasScreen.Js.....	23
AgregarCitaForm	25
NuevoDoctor.Js	27
DESARROLLADORES	28

ESTRUCTURA DEL PROYECTO



Como se puede observar en la ilustración anexa a esta parte del documento, la estructura se divide en 6 carpetas, donde el cada carpeta cuenta con distintas herramientas para poder desarrollar nuestra aplicación. Pero en la carpeta “app” bajo la carpeta “components” se encuentra los archivos .js con los nombres de los archivos que hacen alusión a la sección de la aplicación móvil que se trabajo. Entre esas podemos enumerar:

- AgregarCitaForm
- Home.js
- Login.js
- Perfil.js
- SignIn.js
- TusCitasScreen.js
- Index.js

ARCHIVOS JS PARA VENTANAS DE APLICACIÓN MOVIL

INDEX.JS:

En este archivo se hallara un componente que representa la pagina de inicio de sesión de la aplicación. Gracias a ella, el usuario podra registrarse o iniciar sesión en la plataforma. Se puede mencionar lo siguiente de cada sección del código:

- *Importamos los módulos que se creyeron convenientes para usar con react Native, así como un Link de Expo Router, los hooks, LottieView, un webBrowser, entre otros. Como también importamos componentes a llamar de nuestros archivos para poder navegar entre ellos.*

```
1 import { StyleSheet, Text, View } from "react-native";
2 import { Link } from "expo-router";
3 import React, { useEffect, useState } from "react";
4 import { ImageBackground, Dimensions, Image, Button, TouchableOpacity, Platform } from "react-native";
5 import LottieView from "lottie-react-native";
6 import * as WebBrowser from "expo-web-browser";
7 import * as Google from "expo-auth-session/providers/google";
8 import AsyncStorage from "@react-native-async-storage/async-storage";
9 import { NavigationContainer } from "@react-navigation/native";
10 import { createNativeStackNavigator } from "@react-navigation/native-stack";
11 import { useNavigation } from "@react-navigation/native";
12 import SignIn from './components/signIn';
13 import Login from './components/login';
14 import Home from './components/home';
```

- En el siguiente bloque de código, hallamos el componente "LoginComponent", este componente maneja la autenticación del usuario a la aplicación, también se utilizan estados para el token de dicha autenticación y que maneje la información de usuario. Y como se vio en el apartado anterior se utilizan el hook "useAuthrequest" que proporciona Google para obtener una solicitud de autenticación. Al igual en este bloque se definen funciones para manejar la sesión y obtener dicha información, y una vez ya cargado, se muestra un formulario de inicio de sesión con dos botones "iniciar sesión con Google" o "Registrarse".

```

1  useEffect(() => {
2    AsyncStorage.removeItem("@userInfo");
3    AsyncStorage.removeItem("@user");
4    handleEffect();
5  }, [response, token]);
6
7  async function handleEffect() {
8    const user = await getLocalUser();
9    if (!user) {
10     if (response?.type === "success") {
11       setToken(response.authentication.accessToken);
12       getUserInfo(response.authentication.accessToken);
13     }
14     else {
15       setUserInfo(user);
16       console.log(user);
17       console.log("loaded locally");
18     }
19   }
20 }
21
22 const getLocalUser = async () => {
23   const data = await AsyncStorage.getItem("@user");
24   if (!data) return null;
25   return JSON.parse(data);
26 };
27
28 const getUserInfo = async (token) => {
29   if (!token) return;
30   try {
31     const response = await fetch(
32       "https://www.googleapis.com/userinfo/v2/me",
33       {
34         headers: { Authorization: `Bearer ${token}` },
35       }
36     );
37     const user = await response.json();
38     await AsyncStorage.setItem("@user", JSON.stringify(user));
39     setUserInfo(user);
40   } catch (error) {
41   }
42 };
43
44 const [loading, setLoading] = useState(true);
45
46 useEffect(() => {
47   const timer = setTimeout(() => {
48     setLoading(false);
49   }, 2000);
50
51   return () => clearTimeout(timer);
52 }, []);
53
54 if (loading) {
55   return (
56     <View style={styles.container}>
57       <ImageBackground
58         source={require("../assets/background.png")}
59         style={styles.backgroundImage}
60       >
61         <View style={styles.loginContainer}>
62           {Platform.OS === 'web' ? (
63             <View>
64               <Image
65                 style={styles.title}
66                 source={require("../assets/splash.png")}
67               ></Image>
68             </View>
69           ) : (
70             <View>
71               <LottieView
72                 source={require("../assets/loader.json")}
73                 autoPlay
74                 loop
75                 style={{ width: 300, height: 300 }}
76               />
77             </View>
78           )}
79         </View>
80       </ImageBackground>
81     </View>
82   );
83 }
84 }

```

- Como se observa en el siguiente bloque del código, se procede a crear un Stack Navigator para gestionar la navegación entre diferentes pantallas de la aplicación.

```
1  const Stack = createNativeStackNavigator();
2
3  export default function App() {
4    return (
5      <NavigationContainer independent={true}>
6        <Stack.Navigator initialRouteName="LoginComponent">
7          <Stack.Screen name="LoginComponent" component={LoginComponent} options={{ headersShown: false }} />
8          <Stack.Screen name="Registrate!" component={SigIn} />
9          <Stack.Screen name="Inicia Sesión!" component={Login} />
10         <Stack.Screen name="Home" component={Home} />
11        </Stack.Navigator>
12      </NavigationContainer>
13    );
14  }
15
```


SingIn.js

- Para iniciar se importan bibliotecas como React, componentes de React Native para el desarrollo óptimo de nuestra aplicación, Firebase y AsyncStorage para el almacenamiento local.
- Y se define una constante para poder obtener el ancho de la ventana.

```
1 import { StyleSheet, Text, View } from "react-native";
2 import React, { useState } from "react";
3 import {
4   ImageBackground,
5   Dimensions,
6   TouchableOpacity,
7   Alert,
8   TextInput,
9   KeyboardAvoidingView,
10  ActivityIndicator,
11 } from "react-native";
12 import { getAuth, createUserWithEmailAndPassword } from "firebase/auth";
13 import { initializeApp } from "firebase/app";
14 import { firebaseConfig } from "../../firebase-config";
15 import { getFirestore, doc, setDoc, getDoc } from "firebase/firestore";
16 import AsyncStorage from "@react-native-async-storage/async-storage";
17
18
19 const { width } = Dimensions.get("window");
```

- En el siguiente componente "SingIn" se maneja todo el proceso de registro para aquellos nuevos usuarios a la aplicación. Durante el bloque de código hallara como se han utilizado estados para almacenar campos como el nombre de usuario, apellido, contraseña, correo electrónico y numero de teléfono. También se halla un botón de "Registrarme", donde se intenta crear una nueva cuenta de usuario donde se utiliza el Firebase Auth. Si ha tenido éxito la creación de la cuenta, procede a almacenarse los detalles del usuario en el Firestore y en un almacenamiento local

```

1  function SignIn() {
2    const [username, setUsername] = useState("");
3    const [surname, setSurname] = useState("");
4    const [password, setPassword] = useState("");
5    const [email, setEmail] = useState("");
6    const [celphone, setCelphone] = useState("");
7
8    const [isLoading, setIsLoading] = useState(false);
9
10   const app = initializeApp(firebaseConfig);
11   const auth = getAuth(app);
12
13   const handleCreateAccount = () => {
14     setIsLoading(true);
15     createUserWithEmailAndPassword(auth, email, password)
16       .then((userCredential) => {
17         const db = getFirestore();
18         const userRef = doc(db, "users", userCredential.user.uid);
19         setDoc(userRef, {
20           email: email,
21           name: username,
22           surname: surname,
23           celphone: celphone,
24           createdAt: new Date(),
25           rol: null,
26         })
27         .then(async () => {
28           Alert.alert("Usuario registrado");
29
30           const db = getFirestore();
31           const userRef = doc(db, "users", userCredential.user.uid);
32           try {
33             const docSnap = await getDoc(userRef);
34             if (docSnap.exists()) {
35               const userInfo = await docSnap.data();
36               console.log(docSnap.data(), "docSnap");
37               await AsyncStorage.setItem("@userInfo", JSON.stringify(userInfo));
38             }
39           } catch (error) {
40             Alert.alert("Error al obtener datos del usuario: " + error.message);
41           }
42
43           navigation.navigate("home");
44           setIsLoading(false);
45         })
46       }
47     .catch((error) => {
48       Alert.alert(
49         "Error" + error.message
50       );
51       setIsLoading(false);
52     });
53   });
54   .catch((error) => {
55     console.log(error);
56     setIsLoading(false);
57     Alert.alert(error.message);
58   });
59 });
60
61 return (
62   <KeyboardAvoidingView style={{ flex: 1 }}>
63     <View style={styles.container}>
64       <ImageBackground
65         source={require("../assets/Background.png")}
66         style={styles.backgroundImage}>
67       <View style={styles.title}>
68         <View style={styles.containerForm}>
69           <TextInput
70             style={styles.input}
71             placeholder="Nombre"
72             value={username}
73             onChangeText={(text) => {
74               const formattedText = text.replace(/[^a-zA-Z\s]/g, "");
75               setUsername(formattedText);
76             }}
77             autoCapitalize="words"
78             />
79           <TextInput
80             style={styles.input}
81             placeholder="Apellido"
82             value={surname}
83             onChangeText={(text) => {
84               const formattedText = text.replace(/[^a-zA-Z\s]/g, "");
85               setSurname(formattedText);
86             }}
87             autoCapitalize="words"
88             />
89           <TextInput
90             style={styles.input}
91             placeholder="Correo Electronico"
92             value={email}
93             onChangeText={setEmail}
94             autoCapitalize="none"
95             />
96           <TextInput
97             style={styles.input}
98             placeholder="Celular"
99             value={celphone}
100            onChangeText={(text) => {
101              const formattedText = text.replace(/[^0-9]/g, "").slice(0, 8);
102              setCelphone(formattedText);
103            }}
104            keyboardType="number-pad"
105            maxLength={8}
106            autoCapitalize="none"
107            />
108           <TextInput
109             style={styles.input}
110             placeholder="Contraseña"
111             value={password}
112             secureTextEntry
113             onChangeText={setPassword}
114             autoCapitalize="none"
115             />
116           </View>
117         </View>
118         <View style={styles.bottomSection}>
119           <ActivityIndicator size="large" color="#0000ff" />
120         </View>
121       </View>
122     </KeyboardAvoidingView>
123   );
124 }
125
126 <Text style={styles.buttonText}>Registrar</Text>
127 </TouchableOpacity>
128 </View>
129 </ImageBackground>
130 </View>
131 </KeyboardAvoidingView>
132 );
133 }

```

- Por último, se definen los estilos CSS para los elementos de cada componente.

```
1  const styles = StyleSheet.create({
2    container: {
3      flex: 1,
4      flexDirection: "column",
5      margin: 0,
6    },
7    backgroundImage: {
8      width: width,
9      resizeMode: "cover",
10     flex: 1,
11   },
12   title: {
13     flex: 1,
14     justifyContent: "center",
15     alignItems: "center",
16   },
17   titleText: {
18     fontSize: 36,
19     color: "white",
20   },
21   buttonText: {
22     color: "white",
23     textAlign: "center",
24   },
25   bottomSection: {
26     width: width,
27     height: 50,
28     marginBottom: 10,
29     justifyContent: "center",
30     alignItems: "center",
31   },
32   roundedButton: {
33     paddingHorizontal: 20,
34     paddingVertical: 10,
35     textAlign: "center",
36     backgroundColor: "rgba(7, 17, 109, 1)",
37     borderRadius: 25,
38     elevation: 3,
39     shadowOpacity: 0.3,
40     shadowRadius: 3,
41     shadowOffset: { width: 0, height: 2 },
42   },
43   input: {
44     height: 40,
45     borderRadius: 25,
46     borderColor: "gray",
47     backgroundColor: "white",
48     borderWidth: 1,
49     marginBottom: 20,
50     padding: 10,
51   },
52   containerForm: {
53     width: 300,
54     marginTop: 20,
55     padding: 10,
56   },
57 });
```

LogIn:

- Para iniciar el bloque de código, se importan de igual manera que las anteriores bibliotecas como las de antes, React, componentes de React Native y Firebase.

```
1  import { StyleSheet, Text, View } from "react-native";
2  import React, { useState } from "react";
3  import {
4    ImageBackground,
5    Dimensions,
6    TouchableOpacity,
7    Alert,
8    TextInput,
9    KeyboardAvoidingView,
10   ActivityIndicator
11 } from "react-native";
12 import { getAuth, signInWithEmailAndPassword } from "firebase/auth";
13 import { initializeApp } from "firebase/app";
14 import { firebaseConfig } from "../../firebase-config";
15 import { getFirestore, doc, getDoc } from "firebase/firestore";
16 import AsyncStorage from "@react-native-async-storage/async-storage";
17
```

- El siguiente componente es el encargado de manejar todo el proceso de el inicio de sesión de usuarios que ya estén registrados de forma exitosa, o que ya estén existentes. Para ello utiliza estados para poder almacenar el correo electrónico y la contraseña que ha guardado el usuario. Si el inicio de Sesión es exitoso y no muestra ningún problema, se obtienen todos os detalles des de la base de datos de FireStore y se almacenan.

```

1  const Login = () => {
2
3      const [password, setPassword] = useState("");
4      const [email, setEmail] = useState("");
5
6      const [isLoading, setIsLoading] = useState(false);
7
8      const app = initializeApp(firebaseConfig);
9      const auth = getAuth(app);
10
11     const handleSignIn = () => {
12         setIsLoading(true);
13         signInWithEmailAndPassword(auth, email, password)
14             .then(async (userCredential) => {
15                 console.log('Signed in!');
16                 const user = userCredential.user;
17                 console.log(user);
18
19                 const db = getFirestore();
20                 const userRef = doc(db, "users", user.uid);
21                 try {
22                     const docSnap = await getDoc(userRef);
23                     if (docSnap.exists()) {
24                         console.log("Datos del usuario:", docSnap.data());
25                         const userInfo = await docSnap.data()
26                         await AsyncStorage.setItem("@userInfo", JSON.stringify(userInfo));
27                     }
28                 } catch (error) {
29                     Alert.alert("Error al obtener datos del usuario: " + error.message);
30                 }
31
32                 navigation.navigate('home');
33                 setIsLoading(false);
34             })
35             .catch(error => {
36                 setIsLoading(false);
37                 Alert.alert(
38                     error.message
39                 );
40             });
41     });
42 }
43
44
45
46
47     return (
48         <KeyboardAvoidingView style={{ flex: 1 }}>
49             <View style={styles.container}>
50                 <ImageBackground
51                     source={require("../assets/background.png")}
52                     style={styles.backgroundImage}
53                 >
54                     <View style={styles.title}>
55                         <View style={styles.containerForm}>
56
57
58                         <TextInput
59                             style={styles.input}
60                             placeholder="Correo Electronico"
61                             value={email}
62                             onChangeText={setEmail}
63
64                             autoCapitalize="none"
65                         />
66
67                         <TextInput
68                             style={styles.input}
69                             placeholder="Contraseña"
70                             value={password}
71                             onChangeText={setPassword}
72                             secureTextEntry
73                             autoCapitalize="none"
74                         />
75
76                     </View>
77                 </View>
78
79                 <View style={styles.bottomSection}>
80                     <ActivityIndicator size="large" color="#0000ff" />
81                     <TouchableOpacity
82                         onPress={handleSignIn}
83                         style={styles.roundedButton}
84                     >
85                         <Text style={styles.buttonText}>Entrar</Text>
86                     </TouchableOpacity>
87                 </View>
88             </ImageBackground>
89         </View>
90     </KeyboardAvoidingView>
91 );
92 };
93
94
95
96

```

- Se definen los estilos CSS para cada elemento del componente.

```
1  const styles = StyleSheet.create({
2    container: {
3      flex: 1,
4      flexDirection: "column",
5      margin: 0,
6    },
7    backgroundImage: {
8      width: width,
9      resizeMode: "cover",
10     flex: 1,
11   },
12   title: {
13     flex: 1,
14     justifyContent: "center",
15     alignItems: "center",
16   },
17   titleText: {
18     fontSize: 36,
19     color: "white",
20   },
21   buttonText: {
22     color: "white",
23     textAlign: "center",
24   },
25   bottomSection: {
26     width: width,
27     height: 100,
28     marginBottom: 10,
29     justifyContent: "center",
30     alignItems: "center",
31   },
32   roundedButton: {
33     paddingHorizontal: 20,
34     paddingVertical: 10,
35     textAlign: "center",
36     backgroundColor: "rgba(7, 17, 109, 1)",
37     borderRadius: 25,
38     elevation: 3,
39     shadowOpacity: 0.3,
40     shadowRadius: 3,
41     shadowOffset: { width: 0, height: 2 },
42   },
43   input: {
44     height: 40,
45     borderRadius: 25,
46     borderColor: "gray",
47     backgroundColor: "white",
48     borderWidth: 1,
49     marginBottom: 20,
50     padding: 10,
51   },
52   containerForm: {
53     width: 300,
54     marginTop: 20,
55     padding: 10,
56   },
57 });
```

HOME.JS:

- En esta parte del código se mostrará una lista de pacientes con su información, como nombre, teléfono e ID. La lista de pacientes se muestra utilizando una función que renderiza cada elemento de la lista con su imagen correspondiente al género.

```
import React, { useState } from 'react';
import { Text, View, FlatList, TouchableOpacity, Image } from 'react-native';

const App = () => {
  const [menuVisible, setMenuVisible] = useState(false);

  const toggleMenu = () => {
    setMenuVisible(!menuVisible);
  };

  const patientsData = [
    { name: 'Ernesto Araujo', phone: '0000-0000', id: '#012345', gender: 'male' },
    { name: 'Angel Abarca', phone: '0000-0000', id: '#067891', gender: 'male' },
    { name: 'Lorena Parcas', phone: '0000-0000', id: '#048795', gender: 'female' },
    { name: 'Daniela Pleitez', phone: '0000-0000', id: '#97248', gender: 'female' },
  ];

  // Función para renderizar cada elemento de la lista
  const renderItem = ({ item }) => {
    // Seleccionar la imagen correspondiente al género del paciente
    let imageSource = require('../assets/UserM.png');
    if (item.gender === 'female') {
      imageSource = require('../assets/UserF.png');
    }

    return (
      <View style={{ flex: 1, flexDirection: 'row', marginBottom: 10, alignItems: 'center' }}>
        <Image source={imageSource} style={{ width: 30, height: 30, marginRight: 10 }} />
        <View style={{ flex: 1 }}>
          <Text>{item.name}</Text>
          <Text>{item.phone}</Text>
          <Text>{item.id}</Text>
        </View>
      </View>
    );
  };
};
```

- Este fragmento de código crea un menú desplegable en la parte superior derecha de la pantalla. Cuando se presiona el botón con el icono ☰, se muestra el menú. El menú contiene varias opciones, cada una representada por un botón. Cada botón tiene un

estilo que incluye un fondo de color, bordes redondeados y un margen superior para separarlo del resto del contenido. Los botones están alineados verticalmente y están compuestos por una imagen y un texto. Al presionar cualquier parte del menú fuera de los botones, el menú se oculta nuevamente.

```
return (

  <View style={{ flex: 1, padding: 20 }}>
    /* Expand Menu */
    <View style={{ flexDirection: 'row', justifyContent: 'flex-end', position: 'relative'}}>
      /* Botón para expandir el menú */
      <TouchableOpacity style={{ marginLeft: 10 }} onPress={toggleMenu}>
        <Text>☰</Text>
      </TouchableOpacity>
      /* Menú desplegable */
      <menuVisible && (
        <View style={{ position: 'absolute', top: 30, right: 10, zIndex: 1}}>
          <TouchableOpacity style={{ backgroundColor: '#1499C3', borderRadius: 5, marginTop: 5, flexDirection: 'row', alignItems: 'center', padding: 10, paddingLeft: 25 }}>
            <Image source={require('../../assets/Calendar.png')} style={{ width: 25, height: 25 }} />
            <Text style={{ color: 'white' }}> Tus citas</Text>
          </TouchableOpacity>
          <TouchableOpacity style={{ backgroundColor: '#8D16AB', borderRadius: 5, marginTop: 5, flexDirection: 'row', alignItems: 'center', padding: 10, paddingLeft: 25 }}>
            <Image source={require('../../assets/Paciente.png')} style={{ width: 25, height: 25 }} />
            <Text style={{ color: 'white' }}> Pacientes</Text>
          </TouchableOpacity>
          <TouchableOpacity style={{ backgroundColor: '#0A7461', borderRadius: 5, marginTop: 5, flexDirection: 'row', alignItems: 'center', padding: 10, paddingLeft: 25 }}>
            <Image source={require('../../assets/doctor.png')} style={{ width: 25, height: 25 }} />
            <Text style={{ color: 'white' }}> Doctores</Text>
          </TouchableOpacity>
          <TouchableOpacity style={{ backgroundColor: '#A81C34', borderRadius: 5, marginTop: 5, flexDirection: 'row', alignItems: 'center', padding: 10, paddingLeft: 25 }}>
            <Image source={require('../../assets/user.png')} style={{ width: 25, height: 25 }} />
            <Text style={{ color: 'white' }}> Mi perfil</Text>
          </TouchableOpacity>
          <TouchableOpacity style={{ backgroundColor: '#DF4D0E', borderRadius: 5, marginTop: 5, flexDirection: 'row', alignItems: 'center', padding: 10, paddingLeft: 35 }}>
            <Image source={require('../../assets/salir.png')} style={{ width: 25, height: 25 }} />
            <Text style={{ color: 'white' }}> Salir</Text>
          </TouchableOpacity>
        </View>
      )
    </View>
  )
```

- Este bloque de código muestra un mensaje de bienvenida en la pantalla, que incluye el nombre del doctor o paciente que ingrese a sesión.

```
/* Title */
<Text style={{ fontSize: 20, fontWeight: 'bold', marginTop: 20, fontWeight: 'light'}}>
  Bienvenido de nuevo
</Text>
<Text style={{ fontSize: 40, fontWeight: 'bold', marginTop: 5 }}>
  Dr. Alex Siguenza
</Text>
<Text style={{ fontSize: 16, fontWeight: 'bold', marginTop: 5, fontWeight: 'light' }}>
  ¿Cómo podemos ayudarte ahora?
</Text>
```

- Este bloque de código representa la sección de "Próximas citas" en la pantalla. Comienza con un título en negrita que indica "Tus próximas citas:". Luego, muestra un recuadro con citas individuales, cada una con detalles como el nombre del paciente y la hora de la cita.

Además, hay un botón para agregar una nueva cita y otro para ver todas las citas existentes.

```
/* Proximas citas*/
<View style={{ marginTop: 20 }}>
  <Text style={{ fontSize: 16, fontWeight: 'bold' }}>{'\n'}Tus próximas citas:</Text>
  <View style={{ marginTop: 10 }}>
    /* Recuadro con las citas */
    <View style={{ borderWidth: 1, borderColor: 'black', padding: 10, borderRadius: 5, borderStyle: 'dashed' }}>
      <TouchableOpacity style={{ backgroundColor: '#e3f6fd', position: 'absolute', top: 5, right: 5, borderRadius: 2, padding: 5}}>
        <Text style={{ color: 'black', textAlign: 'center' }}>Agregar nueva cita </Text>
      </TouchableOpacity>
      <View style={{ flexDirection: 'row', alignItems: 'center' }}>
        <Image source={require('../assets/number-8.png')} style={{ width: 30, height: 30, marginRight: 10 }} />
        <View>
          <Text>{'\n'}Paciente: Laura Quintanilla</Text>
          <Text>Hora: 08:00 am</Text>
        </View>
      </View>
    </View>
    <View style={{ flexDirection: 'row', alignItems: 'center' }}>
      <Image source={require('../assets/number-12.png')} style={{ width: 30, height: 30, marginRight: 10 }} />
      <View>
        <Text>Paciente: Ernesto Araujo</Text>
        <Text>Hora: 09:00 am</Text>
      </View>
    </View>
  </View>
  /* Botón para ver citas */
  <TouchableOpacity style={{ backgroundColor: '#2D14C3', padding: 10, borderRadius: 5, marginTop: 10 }}>
    <Text style={{ color: 'white', textAlign: 'center' }}>Ver Todas mis citas</Text>
  </TouchableOpacity>
</View>
</View>
```

- Este bloque de código representa la sección de "Pacientes" en la pantalla. Comienza con un título en negrita que indica "Pacientes:". Luego, muestra una lista de pacientes en un diseño de dos columnas, donde cada paciente se muestra con su nombre, número de teléfono e identificación. Además, hay dos botones al final: uno para agregar un nuevo paciente y otro para ver todos los pacientes existentes.

```
/* Pacientes*/
<View style={{ marginTop: 20 }}>
  /* Mostrar Pacientes */
  <Text style={{ fontSize: 16, fontWeight: 'bold' }}>Pacientes:</Text>
  <View style={{ marginTop: 20, flexDirection: 'row', flexWrap: 'wrap', borderWidth: 1, borderColor: 'black', padding: 10, borderRadius: 5, borderStyle: 'dashed' }}>
    <FlatList
      data={patientsData}
      renderItem={renderItem}
      keyExtractor={(item) => item.id}
      numColumns={2} // Mostrar en dos columnas
    />
  </View>
  /* Botón para agregar nuevo paciente */
  <View style={{ flexDirection: 'row', marginTop: 10 }}>
    <TouchableOpacity style={{ backgroundColor: '#2D14C3', padding: 10, borderRadius: 5, flex: 1, marginRight: 5 }}>
      <Text style={{ color: 'white', textAlign: 'center' }}>Agregar nuevo paciente</Text>
    </TouchableOpacity>
    /* Botón para ver todos los pacientes */
    <TouchableOpacity style={{ backgroundColor: '#008CBA', padding: 10, borderRadius: 5, flex: 1, marginLeft: 5 }}>
      <Text style={{ color: 'white', textAlign: 'center' }}>Ver Todos mis pacientes</Text>
    </TouchableOpacity>
  </View>
</View>
</View>
</View>
export default App;
```

PACIENTES.JS

- Declaraciones de estado

Declaramos un componente con los campos a utilizar para guardar y están inicializados como vacíos

Mantenemos los cambios con el `cambiosDeTexto` y vamos capturando lo que se va ingresando

```
1 import React, { useState, useEffect } from "react";
2 import {
3   ImageBackground,
4   StyleSheet,
5   Text,
6   View,
7   Dimensions,
8   TouchableOpacity,
9   Button, TextInput, ScrollView, Alert, StatusBar,
10  SafeAreaView, Image, FlatList
11 } from "react-native";
12 import * as Font from "expo-font";
13 import {
14   getFirestore,
15   collectionGroup,
16   collection,
17   query,
18   where,
19   getDocs,
20   doc,
21   getDoc,
22 } from "firebase/firestore";
```

- Función para la creación de pacientes

Realizamos primero validación para cada campo, que no venga vacío y al no haber ningún inconveniente realizamos la inserción y muestra mensaje de registro exitoso, en caso ocurriera un error mostramos el mensaje y pintamos en el log el error.

```
1 const pacientesComponent = () => {
2
3   const [listPatient, setListPatient] = useState([]);
4   const [isLoading, setIsLoading] = useState(false);
5   const [fontsLoaded, setFontsLoaded] = useState(false);
6
7   const users = [
8     {
9       id: 1,
10      nombres: "Juan",
11      apellidos: "Pérez",
12      telefono1: "34567890",
13      telefono2: "87654321",
14      correo: "juan.perez@example.com",
15      direccion: "123 Calle Principal, Ciudad, País"
16    },
17    {
18      id: 2,
19      nombres: "María",
20      apellidos: "Gómez",
21      telefono1: "34567891",
22      telefono2: "87654322",
23      correo: "maria.gomez@example.com",
24      direccion: "456 Avenida Secundaria, Ciudad, País"
25    },
26  ],
```

```
1   useEffect(() => {
2     const fetchPatients = async () => {
3       setIsLoading(true);
4
5       const db = getFirestore();
6       try {
7         const doctorUsersQuery = query(collection(db, 'users'), where('rol', '==', 'PATIENT'));
8         const querySnapshot = await getDocs(doctorUsersQuery);
9
10        const doctorUsersList = [];
11        querySnapshot.forEach((doc) => {
12          doctorUsersList.push({ id: doc.id, ...doc.data() });
13        });
14
15        setIsLoading(false);
16        setListPatient(doctorUsersList);
17        return doctorUsersList;
18      } catch (error) {
19        console.error('Error fetching patient users: ', error);
20        setIsLoading(false);
21        return [];
22      }
23    };
24    fetchPatients()
25  }, []);
26
```

Estilos

Se dan el estilo a la vista según lo que se diseño en figma, cada input, textos y botones siguen el diseño establecido.

- Interfaz de usuario

Mostramos al usuario el titulo de la vista y un botón que redirige a la opción de registrar pacientes, luego de esto con el uso del componente que importamos de react-native-elements, hacemos uso de las funciones de ListItem para mostrar en la vista los datos que recuperamos de la bdd.

PERFIL.JS

- Se importan las librerías necesarias.

En esta parte del código se importan todas las librerías necesarias para poder empezar a trabajar, como React y componentes principales de React Native

```
1 import React, { useState } from 'react';
2 import { Text, View, TextInput, TouchableOpacity, StyleSheet } from 'react-native';
3
```

- Declaramos Estados

Estas líneas de código utilizan el hook useState de React para declarar y gestionar el estado de cinco variables: nombre, apellido, email, telefono y especialidad. Cada variable representa un campo de entrada en el formulario de perfil. Cuando el usuario ingresa datos en estos campos, el estado correspondiente se actualiza utilizando las funciones setName, setSurname, setEmail, setTelefono y setEspecialidad.

```
1 const Perfil = () => {
2   const [phone, setPhone] = useState('');
3   const [email, setEmail] = useState('');
4   const [name, setName] = useState('');
5   const [surname, setSurname] = useState('');
6   const [rol, setRol] = useState('');
7
8
9   const getLocalUser = async () => {
10    const data = await AsyncStorage.getItem("@userInfo");
11    const dataParsed = JSON.parse(data);
12    if (dataParsed) {
13      setName(dataParsed.name);
14      setSurname(dataParsed.surname);
15      setRol(dataParsed.rol);
16      setEmail(dataParsed.email);
17      setPhone(dataParsed.celphone);
18    }
19  };
20 }
```

- Guardamos y modificamos el perfil

Estas funciones son responsables de guardar y modificar el perfil del usuario. Actualmente, solo imprimen los datos del perfil en la consola, pero podrían integrarse con una API para enviar los datos al servidor y almacenarlos en una base de datos.


```
1  useEffect(() => {
2    getLocalUser();
3  }, []);
4
5  return (
6    <View style={styles.container}>
7      <View style={styles.header}>
8        <Image source={require('../assets/drs.png')} style={styles.profileImage} />
9        <Text style={styles.greeting}>¡Hola!</Text>
10       <Text style={styles.name}> {rol == "DOCTOR" ? "Dr." : "Paciente"} {name} {surname}</Text>
11     </View>
12     <View style={styles.formRow}>
13       <View style={styles.formGroup}>
14         <Text style={styles.label}>Nombre</Text>
15         <Text style={{fontWeight: 'bold'}}>{name}</Text>
16       </View>
17       <View style={styles.formGroup}>
18         <Text style={styles.label}>Apellido</Text>
19         <Text style={{fontWeight: 'bold'}}>{surname}</Text>
20       </View>
21     </View>
22     <View style={styles.formGroup}>
23       <Text style={styles.label}>Correo Electrónico</Text>
24       <Text style={{fontWeight: 'bold'}}>{email}</Text>
25       <Text style={{marginTop: '10'}}></Text>
26       <Text style={styles.label}>Celular</Text>
27       <Text style={{fontWeight: 'bold'}}>{phone}</Text>
28     </View>
29   </View>
30
31   </View>
32
33 </View>
34 );
35 };
```

- Interfaz de usuario

En el bloque de retorno, se define la interfaz de usuario utilizando elementos de React Native como View, Text, TextInput y TouchableOpacity. Estos elementos se utilizan para mostrar etiquetas de texto, campos de entrada y botones en la pantalla de perfil.

- Estilos

Los estilos se definen utilizando la función StyleSheet.create. Aquí se establecen los estilos para el contenedor principal, así como para las etiquetas, campos de entrada y botones en la pantalla de perfil. Los estilos se aplican a los elementos de la interfaz de usuario mediante la asignación de clases en el código.



```
1  const styles = StyleSheet.create({
2    container: {
3      flex: 1,
4      padding: 20,
5    },
6    label: {
7      fontSize: 16,
8      fontWeight: 'bold',
9      marginBottom: 5,
10   },
11   input: {
12     height: 40,
13     borderColor: 'gray',
14     borderWidth: 1,
15     marginBottom: 10,
16     paddingHorizontal: 10,
17   },
18   button: {
19     backgroundColor: '#008CBA',
20     padding: 10,
21     borderRadius: 5,
22     marginTop: 10,
23   },
24   buttonText: {
25     color: 'white',
26     textAlign: 'center',
27   },
28 });
29
30 export default Perfil;
31
```

TusCitasScreen Js

- Se importan las librerías y dependencias necesarias.

En esta parte del código se importan todas las librerías necesarias para poder empezar a trabajar, como React y componentes principales de React Native.

```
1 import React, { useState, useEffect } from 'react';
2 import { View, Text, StyleSheet, TouchableOpacity, Modal, FlatList, ActivityIndicator } from 'react-native';
3 import AgregarCitaForm from '../AgregarCitaForm';
4 import { getAuth, onAuthStateChanged } from 'firebase/auth';
5 import { getFirestore, doc, collection, collectionGroup, addDoc, getDoc, getDocs, query, where } from 'firebase/firestore';
6 import AsyncStorage from '@react-native-async-storage/async-storage';
7 import { useNavigation } from '@react-navigation/native';
```

- Declaramos los estados

El componente utiliza useState para manejar varios estados:

```
1 const TusCitasScreen = () => {
2   const db = getFirestore();
3   const auth = getAuth();
4   const navigation = useNavigation();
5   const [name, setName] = useState('');
6   const [surname, setSurname] = useState('');
7   const [rol, setRol] = useState('');
8   const [appointments, setAppointments] = useState([]);
9   const [appointmentsUser, setAppointmentsUser] = useState([]);
10  const [isLoading, setIsLoading] = useState(false);
```

- Autenticación del Usuario

Este useEffect se ejecuta una vez cuando el componente se monta, configurando un listener para los cambios de estado de autenticación.

```
1 useEffect(() => {
2   const unsubscribe = onAuthStateChanged(auth, (currentUser) => {
3     if (currentUser) {
4       AsyncStorage.setItem("@userLogged", JSON.stringify(currentUser));
5     }
6   });
7
8   return () => unsubscribe();
9 }, []);
```

- Obtención de información Local y de citas

Este useEffect se ejecuta una vez cuando el componente se monta para obtener la información del usuario almacenada localmente, y este useEffect se encarga de obtener las citas del doctor o usuario desde Firestore cuando el componente se monta.

- Agregamos Cita al FireStore

```
1  const agregarCita = (nuevaCita) => {
2
3    addAppointment(nuevaCita);
4  };
5
6
7  const addAppointment = async (newAppointment) => {
8    const { comentario, fechaCita, razonCita, selectedPatient } = newAppointment;
9
10   try {
11
12     const userInfo = await AsyncStorage.getItem('@userLogged');
13     const user = userInfo ? JSON.parse(userInfo) : null;
14
15     if (!user || !user.uid) {
16       throw new Error('User information is not available');
17     }
18
19     const doctorId = user.uid;
20
21
22
23     const userDocRef = doc(db, 'users', selectedPatient);
24     const appointmentsColRef = collection(userDocRef, 'appointments');
25     const patientDocSnap = await getDoc(userDocRef);
26
27     const patientName = patientDocSnap.data().name;
28
29
30     await addDoc(appointmentsColRef, {
31       comentario,
32       fechaCita,
33       razonCita,
34       doctorId,
35       patientName
36     });
37
38
39     navigation.goBack()
40   } catch (error) {
41     console.error('Error:', error);
42   }
43 };
44
```


AgregarCitaForm

- Declaraciones de estado

El componente AgregarCitaForm utiliza useState para manejar varios estados esenciales para la funcionalidad del formulario:

```
1 const AgregarCitaForm = ({ onCancel, onAgregar }) => {
2   const [patientsData, setPatientsData] = useState([]);
3   const [selectedPatient, setSelectedPatient] = useState([]);
4   const [fechaCita, setFechaCita] = useState('');
5   const [razonCita, setRazonCita] = useState('');
6   const [comentario, setComentario] = useState('');
```

- Funciones principales:

‘handleAgregar’

La función handleAgregar se encarga de validar los datos del formulario y enviarlos a la función onAgregar pasada como prop:

```
1 const handleAgregar = () => {
2   // Validar datos antes de agregar
3   if (!selectedPatient || !fechaCita || !razonCita) {
4     alert('Por favor, complete todos los campos obligatorios.');
```

fetchPatients

Esta función obtiene la lista de pacientes desde Firestore excluyendo al usuario actual:

```
1  const fetchPatients = async () => {
2    const db = getFirestore();
3    const auth = getAuth();
4    const currentUser = auth.currentUser;
5    const currentUserId = currentUser ? currentUser.uid : null;
6
7    const q = query(collection(db, "users"), where("rol", "==", "PATIENT"));
8    const querySnapshot = await getDocs(q);
9    const patients = [];
10   querySnapshot.forEach((doc) => {
11     if(doc.id !== currentUserId) {
12       patients.push({ id: doc.id, ...doc.data() });
13     }
14   });
15   return patients;
16 }
```

handleDateChange

Esta función formatea y establece la fecha de la cita ingresada por el usuario:

```
1  const handleDateChange = (text) => {
2    let newText = text.replace(/[^0-9]/g, '');
3
4    if (newText.length > 2 && newText.length <= 4) {
5      newText = newText.slice(0, 2) + '/' + newText.slice(2);
6    } else if (newText.length > 4) {
7      newText = newText.slice(0, 2) + '/' + newText.slice(2, 4) + '/' + newText.slice(4, 8);
8    }
9
10   setFechaCita(newText.slice(0, 10));
11 }
```

○ Interfaz de usuario y Estilos

El componente renderiza un formulario con campos para seleccionar un paciente, ingresar la fecha de la cita, la razón de la misma y comentarios adicionales. A su vez los estilos se definen utilizando StyleSheet de React Native para dar formato y diseño al formulario.

NuevoDoctor Js

- Estructura del Componente:

El componente NuevoDoctorForm es una función que retorna un formulario con dos campos de entrada y un botón:

```
1
2  const NuevoDoctorForm = () => {
3    return (
4      <View style={styles.formContainer}>
5        <Text style={styles.formLabel}>Nombre:</Text>
6        <TextInput style={styles.textInput} />
7
8        <Text style={styles.formLabel}>Especialidad:</Text>
9        <TextInput style={styles.textInput} />
10
11       <TouchableOpacity style={styles.addButton}>
12         <Text style={styles.addButtonText}>Agregar Doctor</Text>
13       </TouchableOpacity>
14     </View>
15   );
16 };
```

- Interfaz de Usuario y Estilos

El diseño del formulario está definido por varios estilos de StyleSheet de React Native, lo que proporciona una apariencia limpia y coherente.

DESARROLLADORES

En nombre del equipo de desarrolladores que formaron parte de este proyecto, esperamos que este manual pueda ser de utilidad para mayor comprensión de la funcionalidad del código. Cada parte ha sido desarrollada con mucho esfuerzo y dedicación, para brindar la mejor solución al problema.



Robles Rodas, Melvin Eduardo



Lue Valdez, Juan Jose



Díaz Merino, Jesus Alexander



Fuentes Guardado, Daniela
Alejandra



Abarca Flores, Luis Ángel