**Edushare J.S**

**Integrantes:**

César Francisco Escamilla Heredia

Yarid Alexander Barrientos Magaña

Gracia Lizbeth Aguirre Portillo

Mario de Jesús Rodríguez Méndez

Edwin Steven Valencia Castillo

Erika Stephanie Portillo Gómez

Complejo Educativo Jutta Steiner de Toruño

Modulo 3.2

CALEB VERENICE LOPEZ GUTIERREZ

11 de septiembre del 2024

Índice

[Introducción 4](#_Toc180427967)

[Requisitos previos 5](#_Toc180427968)

[**Herramientas necesarias** 5](#_Toc180427969)

[Requerimientos técnicos del sistema 6](#_Toc180427970)

[Instalación del proyecto 7](#_Toc180427971)

[**Instalacion de edushare** 7](#_Toc180427972)

[**Configuración de firebase** 8](#_Toc180427973)

[Configuración y Uso de Firebase 9](#_Toc180427974)

[**Autenticación de usuarios (Firebase Auth)** 9](#_Toc180427975)

[**Registro de Usuarios (Sign up)** 10](#_Toc180427976)

[**Inicio de Sesión (Sign in)** 11](#_Toc180427977)

[**Cerrar Sesión (Sign out)** 12](#_Toc180427978)

[**Escritura de Datos** 13](#_Toc180427979)

[**Lectura de Datos** 14](#_Toc180427980)

[Componentes y funcionalidades 15](#_Toc180427981)

[**Chat** 15](#_Toc180427982)

[***Librerías usadas*** 15](#_Toc180427983)

[**Gestión de Mensajes** 15](#_Toc180427984)

[**Subida y Vista de Archivos** 16](#_Toc180427985)

[***Implementación de subida de archivos*** 16](#_Toc180427986)

[**Manipulación de Imágenes** 16](#_Toc180427987)

[**Navegación y Diseño de la Aplicación** 16](#_Toc180427988)

[**Navegación** 16](#_Toc180427989)

[**Interfaz de usuario** 17](#_Toc180427990)

[Navegación en la App 17](#_Toc180427991)

[Implementación de Firebase Reglas 19](#_Toc180427992)

[Estructura del proyecto 22](#_Toc180427993)

[**Descripción de las Carpetas** 22](#_Toc180427994)

[Pruebas y Debugging 24](#_Toc180427995)

[**Ejecución de la Aplicación en Modo de Desarrollo** 24](#_Toc180427996)

[**Expo Go** 24](#_Toc180427997)

[**Herramientas de Depuración** 24](#_Toc180427998)

[Diagrama DFD 25](#_Toc180427999)

[Arquitectura de Software 27](#_Toc180428000)

[Modelado de la BD 29](#_Toc180428001)

[**Modelado Clave-Valor** 29](#_Toc180428002)

[**Modelo documental** 30](#_Toc180428003)

[Descripción de las tablas de la base de datos 32](#_Toc180428004)

[**Usuarios (Users)** 32](#_Toc180428005)

[**Archivos (Files)** 32](#_Toc180428006)

[**Mensajes (Messages)** 32](#_Toc180428007)

[Definición y descripción de variables usadas en el sistema. 33](#_Toc180428008)

[Diccionario de datos 42](#_Toc180428009)

# **Introducción**

En el presente documento se presenta el manual de programador para la aplicación Edushare, el cual proporciona una guía general y directa sobre el comportamiento y especificaciones de cada parte de la app. Este manual servirá como referencia para la estructura, mantenimiento y desarrollo de la aplicación.

Edushare.js es una plataforma de chat con un apartado de subida de archivos diseñada para la colaboración entre profesores y estudiantes. Permite a los estudiantes visualizar archivos, comunicarse a través de un chat y ayudarse mutuamente. Por su parte, los docentes tienen características similares, pero con la capacidad adicional de subir archivos.

Este manual está destinado a desarrolladores que deseen entender la arquitectura de Edushare, así como a aquellos que estén interesados en contribuir al desarrollo y mantenimiento de la aplicación.

La aplicación está construida utilizando React Native junto con expo y se integra con Firebase para la autenticación y almacenamiento de datos.

# **Requisitos previos**

Para la instalación, mantenimiento, carga y desarrollo de la aplicación Edushare, es necesario contar con las siguientes especificaciones y herramientas:

## **Herramientas necesarias**

**Node.js.** Se necesita para ejecutar JavaScript en el entorno de servidor y manejar npm (Node Package Manager). Se recomiendas versiones de 16.x o superior.

**Npm.** Administrador de paquetes para instalar dependencias. Viene incluido con Node.js , la mayoría de dependencias del proyecto usaron esto para su descarga. Para verificar que está instalado correctamente, ejecuta el siguiente comando: **npm -v**

**Expo CLI.** Utilizado para crear y ejecutar proyectos de React Native. Se instala de manera global , ya sea en CMD o powershell , ejecuta **expo –version** para confirmar su instalación .

**Tabla 1**

Comando para instalar expo de forma global

|  |
| --- |
| npm install -g expo-cli |

**Git.** Herramienta de control de versiones utilizada para gestionar el código y también para trabajar con github.

**Editor de Código:** Se recomienda usar Visual Studio Code (VS Code) por su compatibilidad con extensiones útiles para el desarrollo en React Native.

# **Requerimientos técnicos del sistema**

A continuación se presentan las especificaciones y las condiciones necesarias para desarrollar, ejecutar y mantener la aplicación Edushare.

* Sistema operativo: Windows 10 o superior
* Lenguaje de programación : JavaScript
* Framework: React native con expo
* Base de datos: firebase firestore para el almacenamiento de los datos , firebase storage para los archivos y firebase authentication para la autenticación y la validación de los usuarios.
* Requisitos de hardware: computadora con 116 gb de almacenamiento y 10/15 gb libres , 4 de RAM , procesador de Intel(R) Celeron(R) N5100 @ 1.10GHz 1.11 GHz y superiores
* Herramientas de desarrollo: node.js (versión 14 y superiores),npm (versión 6 y superiores), Expo CLI (versión 5.0 o superior)
* Conexión a internet: conexión estable para el acceso a firebase , con frecuencia de 2.5 funciona decente.

# **Instalación del proyecto**

Para seguir con el proceso de la app se le pide que clone el repositorio de GitHub, descargue o que descomprima el archivo en USB que le pasemos.

## **Instalación de edushare**

A continuación se explicara cada uno de los procesos que se pueden realizar.

**Tabla 2**

En caso de clonar , Copiar directorio de GitHub.

|  |
| --- |
| **git clone** [**https://github.com/usuario/edushare.git**](https://github.com/usuario/edushare.git)  **cd edushare** |

<https://github.com/YaridMagFu/EduShareJS/tree/interfaces> descarga el archivo zip de la rama de interfaces , este es el proyecto completo.

Si ya tienes el proyecto en tu PC , ábrelo en un editor de código y ejecuta el siguiente comando para instalar todas las dependencias.

**Tabla 3**

Esto instalara todas las dependencias que estén en el packaje.json

|  |
| --- |
| **npm install** |

## **Configuración de firebase**

Edushare utiliza Firebase para la autenticación de usuarios y almacenamiento de datos. El proyecto lleva una conexión de prueba pero si quieres iniciar una nueva haz lo siguiente:

**Crear un proyecto en Firebase.** Ve a Firebase Console , Crea un nuevo proyecto y asigna un nombre.

**Instalación de dependencias de Firebase.** Asegúrate de tener las bibliotecas de Firebase instaladas, ejecuta en la consola de tu editor el siguiente código : **npm install firebase**.

**Configurar Firebase en la App.** En el archivo de firebase.js configura tu información de tu proyecto creado en firebase, se debe de ver asi.

**Tabla 4**

Firease.js configuración , este es un ejemplo de cómo debe verse.

|  |
| --- |
| import { initializeApp } from ‘firebase/app’;  import { getAuth } from ‘firebase/auth’;  import { getFirestore } from ‘firebase/firestore’;  import { getStorage } from ‘firebase/storage’;  const firebaseConfig = {      apiKey: “AIzaSyAiEVuLVat3NGmHv0zYLcayOQE3B45Seus”,      authDomain: “intentobueno-8bc83.firebaseapp.com”,      projectId: “intentobueno-8bc83”,      storageBucket: “intentobueno-8bc83.appspot.com”,      messagingSenderId: “1096943338637”,      appId: “1:1096943338637:web:dc44f3fffab904bd90cec4”  };  const app = initializeApp(firebaseConfig);  export const auth = getAuth(app);  export const db = getFirestore(app);  export const storage = getStorage(app); |

# **Configuración y Uso de Firebase**

Firebase es una parte esencial de Edushare, que se utiliza tanto para la autenticación de usuarios como para el almacenamiento de datos a través de Firestore. A continuación, se explican los pasos para configurar Firebase, cómo manejar la autenticación y la base de datos, y las reglas de seguridad necesarias para controlar el acceso a los datos.

## **Autenticación de usuarios (Firebase Auth)**

Firebase Auth se utiliza en Edushare para gestionar la autenticación de los usuarios (profesores y estudiantes). Este sistema permite iniciar sesión mediante correo electrónico y contraseña.

**Tabla 5**

Firebase.js

|  |
| --- |
| import { initializeApp } from ‘firebase/app’;  import { getAuth } from ‘firebase/auth’;  import { getFirestore } from ‘firebase/firestore’;  import { getStorage } from ‘firebase/storage’;  const firebaseConfig = {  apiKey: “AIzaSyAiEVuLVat3NGmHv0zYLcayOQE3B45Seus”,  authDomain: “intentobueno-8bc83.firebaseapp.com”,  projectId: “intentobueno-8bc83”,  storageBucket: “intentobueno-8bc83.appspot.com”,  messagingSenderId: “1096943338637”,  appId: “1:1096943338637:web:dc44f3fffab904bd90cec4”  };  const app = initializeApp(firebaseConfig);  export const auth = getAuth(app);  export const db = getFirestore(app);  export const storage = getStorage(app); |

## **Registro de Usuarios (Sign up)**

El registro de usuarios se encuentra en el archivo register.js. Aquí se utilizan las funciones de Firebase para crear usuarios y almacenar información adicional como el nombre y el rol (alumno o docente).

**Tabla 6**

RegistersScreen.js , código relevante.

|  |
| --- |
| const handleRegister = async () => {  try {  // Crear un nuevo usuario con email y contraseña  const userCredential = await createUserWithEmailAndPassword(auth, email, password);    // Actualizar el perfil del usuario con el nombre  await updateProfile(userCredential.user, { displayName: name });    // Guardar datos adicionales en Firestore  await setDoc(doc(db, ‘users’, userCredential.user.uid), {  name,  email,  role, // ‘docente’ o ‘alumno’  year: role === ‘alumno’ ? year : null, // Solo almacenar el año si es alumno  });  // Navegar a la pantalla principal después de registrar  navigation.navigate(‘Home’);  } catch (error) {  Alert.alert(‘Error’, error.message);  }  }; |

**Nota:** Registro: Se crea el usuario con email y contraseña mediante createUserWithEmailAndPassword.El perfil se del usuario se actualiza con su nombre usando updateProfile. Firestore se guarda información adicional del usuario (nombre, rol, año para alumnos) en la colección users.

## **Inicio de Sesión (Sign in)**

El inicio de sesión está implementado en login.js, donde se valida el email y la contraseña, además de manejar errores comunes como usuario no encontrado o contraseña incorrecta.

**Tabla 7**

|  |
| --- |
| const handleLogin = async () => {  try {  // Verificar conexión a internet  const netInfo = await NetInfo.fetch();  if (!netInfo.isConnected) throw new Error(‘No hay conexión a internet’);  // Iniciar sesión con email y contraseña  await signInWithEmailAndPassword(auth, email, password);  // Navegar a la pantalla principal  navigation.navigate(‘Home’);  } catch (error) {  // Manejo de errores como usuario no encontrado o contraseña incorrecta  let errorMessage = ‘Ocurrió un error al iniciar sesión. Por favor, inténtelo de nuevo.’;  if (error.code === ‘auth/user-not-found’ || error.code === ‘auth/wrong-password’) {  errorMessage = ‘Email o contraseña incorrectos.’;  }  Alert.alert(‘Error de inicio de sesión’, errorMessage);  }  }; |

LoginScreen.js,código relevante.

**Nota**. Inicio de sesión utiliza signInWithEmailAndPassword para autenticar al usuario.

Se verifica la conexión a internet antes de proceder. Mensajes específicos para errores comunes como usuario no encontrado o demasiados intentos.

## **Cerrar Sesión (Sign out)**

El proceso de cierre de sesión está implementado usando Firebase y el método signOut de Firebase Auth. Este método desconecta al usuario de la sesión actual y luego navega a la pantalla de inicio de sesión.

**Tabla 8**

HomeScreen.js, código relevante

|  |
| --- |
| import { signOut } from ‘firebase/auth’;  import { Alert } from ‘react-native’;  import NetInfo from ‘@react-native-community/netinfo’;  const handleLogout = async () => {  try {  const netInfo = await NetInfo.fetch();  if (!netInfo.isConnected) {  Alert.alert(‘Error’, ‘No hay conexión a internet. Verifique su conexión e inténtelo nuevamente.’);  return;  }  await signOut(auth); // Cierra la sesión del usuario actual  navigation.reset({  index: 0,  routes: [{ name: ‘Login’ }], // Navega a la pantalla de inicio de sesión  });  } catch (error) {  Alert.alert(‘Error’, ‘Hubo un problema al cerrar sesión. Por favor, inténtelo de nuevo.’);  }  }; |

**Nota.** El cerrar sesión utiliza el método signOut de Firebase Auth para cerrar la sesión del usuario.

Tras el cierre de sesión, redirige al usuario a la pantalla de inicio de sesión usando navigation.reset para evitar que el usuario pueda volver a la pantalla anterior mediante el botón de “atrás”. Si no hay conexión a internet o si ocurre algún error durante el proceso, muestra un mensaje de alerta adecuado al usuario.

## **Escritura de Datos**

La escritura de datos en Firestore se realiza al subir archivos en FilesScreen.js. Se utiliza setDoc para guardar los detalles del archivo en la base de datos.

**Tabla 9**

FileScreen.js, código relevante

|  |
| --- |
| const handleUpload = async () => {  try {  const fileDoc = {  name: selectedFile.name,  url: downloadURL,  description: description,  year: year,  uploadedBy: currentUser.displayName || ‘Usuario Anónimo’,  createdAt: new Date(),  };  // Guardar los detalles del archivo en la colección ‘files’  await setDoc(doc(db, ‘files’, selectedFile.name), fileDoc);  Alert.alert(‘Éxito’, ‘Archivo subido correctamente’);  setFiles(prevFiles => [...prevFiles, fileDoc]);  } catch (error) {  Alert.alert(‘Error’, ‘Hubo un problema al subir el archivo’);  }  }; |

**Nota.** Se almacena la información del archivo en Firestore, como nombre, URL de descarga, descripción y fecha de creación.

## **Lectura de Datos**

La lectura de archivos desde Firestore se realiza usando getDocs para obtener todos los archivos en la colección files.

**Tabla 10**

FileScreen.js , código relevante.

|  |
| --- |
| useEffect(() => {  const fetchFiles = async () => {  const filesCollection = collection(db, 'files');  const filesSnapshot = await getDocs(filesCollection);  const filesList = filesSnapshot.docs.map(doc => doc.data());  setFiles(filesList);  };  fetchFiles();  }, []); |

**Nota.** La lectura de archivos se obtiene de todos los documentos de la colección files usando getDocs.

# **Componentes y funcionalidades**

En esta sección se describen los principales componentes y funcionalidades que forman parte de la arquitectura de la aplicación Edushare. Cada funcionalidad se basa en la integración de diversas librerías y tecnologías para ofrecer una experiencia fluida y eficiente en la colaboración entre profesores y estudiantes.

## **Chat**

El chat de Edushare permite la comunicación en tiempo real entre estudiantes y profesores. Se utiliza Firebase Firestore para almacenar y gestionar los mensajes de manera eficiente. Los usuarios se agrupan en diferentes salas de chat según su grado (por ejemplo “Chat de Primer Año” que solo pueden acceder a ese).

### ***Librerías usadas***

**@react-navigation/native y @react-navigation/stack.**  Para la navegación entre las diferentes pantallas de la aplicación.

**Firebase.** Para la autenticación de usuarios y almacenamiento de mensajes en Firestore.

**react-native-keyboard-aware-scroll-view.** Permite ajustar automáticamente la vista cuando aparece el teclado mientras se escribe un mensaje.

### **Gestión de Mensajes**

Los mensajes se almacenan en Firebase Firestore y se actualizan en tiempo real usando listeners de Firebase. Cada usuario puede ver el historial del chat al ingresar a una sala, y los nuevos mensajes se añaden automáticamente.

## **Subida y Vista de Archivos**

Los usuarios con permisos pueden subir archivos , todos pueden visualizar documentos dentro de la aplicación. Firebase Storage se utiliza como backend para el almacenamiento de los archivos.

### ***Implementación de subida de archivos***

La funcionalidad de subida de archivos se gestiona con expo-document-picker para permitir a los usuarios seleccionar archivos, y luego se suben a Firebase Storage usando las APIs de Firebase y el manejo de archivos en el sistema de archivos local utiliza expo-file-system.

***Vista de archivos***

Los archivos subidos (PDFs y otros formatos) pueden visualizarse mediante react-native-webview o componentes personalizados, dependiendo del tipo de archivo.

## **Manipulación de Imágenes**

Para optimizar y manipular imágenes antes de subirlas, se utiliza la librería expo-image-manipulator. Esto permite ajustar el tamaño, rotar y comprimir las imágenes, mejorando el rendimiento y reduciendo el tamaño de los archivos subidos. Después de manipular las imágenes, estas se suben a Firebase Storage, siguiendo un proceso similar al de otros archivos.

## **Navegación y Diseño de la Aplicación**

La navegación en Edushare está estructurada utilizando el sistema de navegación de React Navigation, lo que permite un flujo intuitivo y eficiente entre las diferentes pantallas de la aplicación.

### **Navegación**

Se utiliza @react-navigation/native y @react-navigation/stack para la navegación entre pantallas dentro de la aplicación.

### **Interfaz de usuario**

La interfaz está diseñada para ser simple y funcional, utilizando react-native-vector-icons para iconografía y react-native-linear-gradient para efectos visuales de fondo y el uso de react-native-safe-area-context asegura que los componentes se ajusten correctamente en diferentes dispositivos y tamaños de pantalla.

# **Navegación en la App**

La navegación en la aplicación se gestiona utilizando React Navigation, que permite una transición fluida entre las diferentes pantallas. Se emplea createStackNavigator para definir una pila de pantallas que incluye:

**LoginScreen.** Pantalla de inicio de sesión, sin barra de encabezado visible.

**RegisterScreen.** Pantalla de registro de usuarios.

**RoleSelectionScreen.** Permite seleccionar el rol de usuario (estudiante o maestro).

**HomeScreen.** Pantalla principal (oculta la barra de encabezado).

**ChatScreen.** Para gestionar los chats (también sin barra de encabezado).

**ProfileScreen.** Muestra el perfil del usuario.

**FilesScreen.** Permite la visualización y gestión de archivos.

**PasswordResetScreen y PasswordResetConfirmationScreen.** Pantallas para restablecer la contraseña, con la primera sin barra de encabezado.

El contenedor de navegación NavigationContainer envuelve la configuración de la pila, gestionando el estado y las rutas de la aplicación.

**Tabla 11**

AppNavigator.js , como se debe de ver.

|  |
| --- |
| import React from 'react';  import { NavigationContainer } from '@react-navigation/native';  import { createStackNavigator } from '@react-navigation/stack';  import LoginScreen from '../screens/LoginScreen';  import RegisterScreen from '../screens/RegisterScreen';  import RoleSelectionScreen from '../screens/RoleSelectionScreen';  import HomeScreen from '../screens/HomeScreen';  import ChatScreen from '../screens/ChatScreen';  import ProfileScreen from '../screens/ProfileScreen';  import FilesScreen from '../screens/FilesScreen';  import PasswordResetScreen from '../screens/PasswordResetScreen';  import PasswordResetConfirmationScreen from '../screens/PasswordResetConfirmationScreen';  const Stack = createStackNavigator();  export default function AppNavigator() {  return (  <NavigationContainer>  <Stack.Navigator>  <Stack.Screen name="Login" component={LoginScreen} options={{ headerShown: false }} />  <Stack.Screen name="Register" component={RegisterScreen} />  <Stack.Screen name="RoleSelection" component={RoleSelectionScreen} />  <Stack.Screen name="Home" component={HomeScreen} options={{ headerShown: false }} />  <Stack.Screen name="Chat" component={ChatScreen} options={{ headerShown: false }} />  <Stack.Screen name="Profile" component={ProfileScreen} />  <Stack.Screen name="Files" component={FilesScreen} />  <Stack.Screen name="PasswordReset" component={PasswordResetScreen} options={{ headerShown: false }}/>  <Stack.Screen name="PasswordResetConfirmation" component={PasswordResetConfirmationScreen} />  </Stack.Navigator>  </NavigationContainer>  );  } |

# **Implementación de Firebase Reglas**

En esta sección se detallan las reglas de seguridad implementadas en Firebase Firestore para gestionar el acceso y las operaciones en la base de datos. Estas reglas están diseñadas para asegurar que solo los usuarios autenticados tengan acceso a los recursos apropiados, diferenciando entre los permisos de los estudiantes y los profesores.

**Tabla 12**

Reglas de seguridad en firestore

|  |
| --- |
| rules\_version = '2';  service cloud.firestore {  match /databases/{database}/documents {  function isAuthenticated() {  return request.auth != null;  }    function isUserOwner(userId) {  return isAuthenticated() && request.auth.uid == userId;  }    function isTeacher() {  return isAuthenticated() &&  get(/databases/$(database)/documents/users/$(request.auth.uid)).data.role == 'docente';  }  match /users/{userId} {  allow read: if isAuthenticated();  allow create: if isAuthenticated();  allow update, delete: if isUserOwner(userId);  }    match /chats/{chatRoom} {  allow read, write: if isAuthenticated();    match /messages/{messageId} {  allow read: if isAuthenticated();  allow create: if isAuthenticated() &&  request.resource.data.uid == request.auth.uid;  allow update: if isAuthenticated() &&  (resource.data.uid == request.auth.uid ||  request.resource.data.diff(resource.data).affectedKeys().hasOnly(['replies']));  allow delete: if isAuthenticated() && resource.data.uid == request.auth.uid;  }  }    match /modules/{moduleId} {  allow read: if isAuthenticated();  allow create, update, delete: if isTeacher();  }    match /files/{fileId} {  allow read: if isAuthenticated();  allow create, update, delete: if isTeacher();  allow create: if request.resource.data.keys().hasAll(['name', 'url', 'description', 'uploadedBy', 'createdAt', 'year']) && isTeacher();  allow update: if request.resource.data.keys().hasAny(['description', 'uploadedBy', 'year']) && isTeacher();  }  }  } |

**Tabla 13**

Reglas de seguridad en storage

|  |
| --- |
| rules\_version = '2';  service firebase.storage {  match /b/{bucket}/o {  function isAuthenticated() {  return request.auth != null;  }    function isDocente() {  return firestore.get(/databases/(default)/documents/users/$(request.auth.uid)).data.role == 'docente';  }    match /avatars/{userId}/{allPaths=\*\*} {  allow read: if true;  allow write: if isAuthenticated() && request.auth.uid == userId;  }    match /profileImages/{userId} {  allow read: if isAuthenticated();  allow write: if isAuthenticated() && request.auth.uid == userId;  }    match /files/{fileName} {  allow read: if isAuthenticated();  allow write: if isAuthenticated() && isDocente();  }  }  } |

# **Estructura del proyecto**

La organización de archivos y carpetas en el proyecto Edushare es fundamental para mantener un código limpio y fácil de mantener. A continuación se muestra la estructura recomendada:

**Tabla 14**

Diagrama de carpetas

|  |
| --- |
| **/Edushare**  ├── **/assets** # Recursos estáticos (imágenes, fuentes, etc.)  │ └── /logo # Ejemplo de logo  **└── /src** # Código fuente de la aplicación  ├── **/components** # Componentes reutilizables de UI  │ └── messageItems.js # Ejemplo de componente  ├── **/navigation** # Configuración de navegación  │ └──AppNavigator.js # Ejemplo de componente  **├── /screens** # Pantallas principales de la aplicación  │ ├── HomeScreen.js # Ejemplo de pantalla principal  │ └── ChatScreen.js # Ejemplo de pantalla de chat  ├**── /utils** # Funciones utilitarias reutilizables  └── permissions.js # Ejemplo de función de permisos  **└── App.js** # Entrada principal de la aplicación |

## **Descripción de las Carpetas**

**Assets.** Contiene todos los recursos estáticos que se utilizan en la aplicación, como imágenes, fuentes y otros archivos multimedia. Sin embargo se puede usar otros espacios para esto como es el caso de screens con la carpeta logo , aunque aquí seria lo más recomendable.

**Src:** Carpeta que contiene todo el código fuente de la aplicación.

**Components.** Aquí se encuentran los componentes de interfaz de usuario que se pueden reutilizar en diferentes partes de la aplicación, como botones, formularios y tarjetas. Como ejemplo el archivo de messageItems.js

**Navigation.** Gestiona y organizar la navegación entre diferentes pantallas o vistas de la aplicación.

**Screens.** Almacena las pantallas principales de la aplicación, que representan las distintas vistas a las que los usuarios pueden acceder. Como ejemplo HomeScreen.js, ChatScreen.js,etc.

**Archivos styles.** Son los que tienen los estilos y especificaciones que se utilizan para aplicar estilos a los componentes de la aplicación. Están en la carpeta screen con nombres como chatScrenStyles.js.

**Utils.** Almacena funciones utilitarias que son reutilizables en diferentes partes de la aplicación. En este caso el archivo permissions.js y firebase.js

**App.js.** Este es el archivo principal de la aplicación, donde se configura la navegación y se inicializan los componentes de alto nivel.

# **Pruebas y Debugging**

Durante el desarrollo de la aplicación, se implementan herramientas para facilitar la depuración y probar el funcionamiento de la app

## **Ejecución de la Aplicación en Modo de Desarrollo**

Utilizo el comando npx expo start para iniciar la aplicación en modo de desarrollo. Este comando proporciona una interfaz interactiva que permite seleccionar la plataforma (Android, iOS o web) en la que se desea ejecutar la aplicación.

## **Expo Go**

Para probar la aplicación en dispositivos móviles reales, se utiliza la aplicación Expo Go. Con Expo Go, la aplicación puede cargarse rápidamente en dispositivos Android e iOS al escanear el código QR generado, permitiendo la vista previa de cambios en tiempo real.

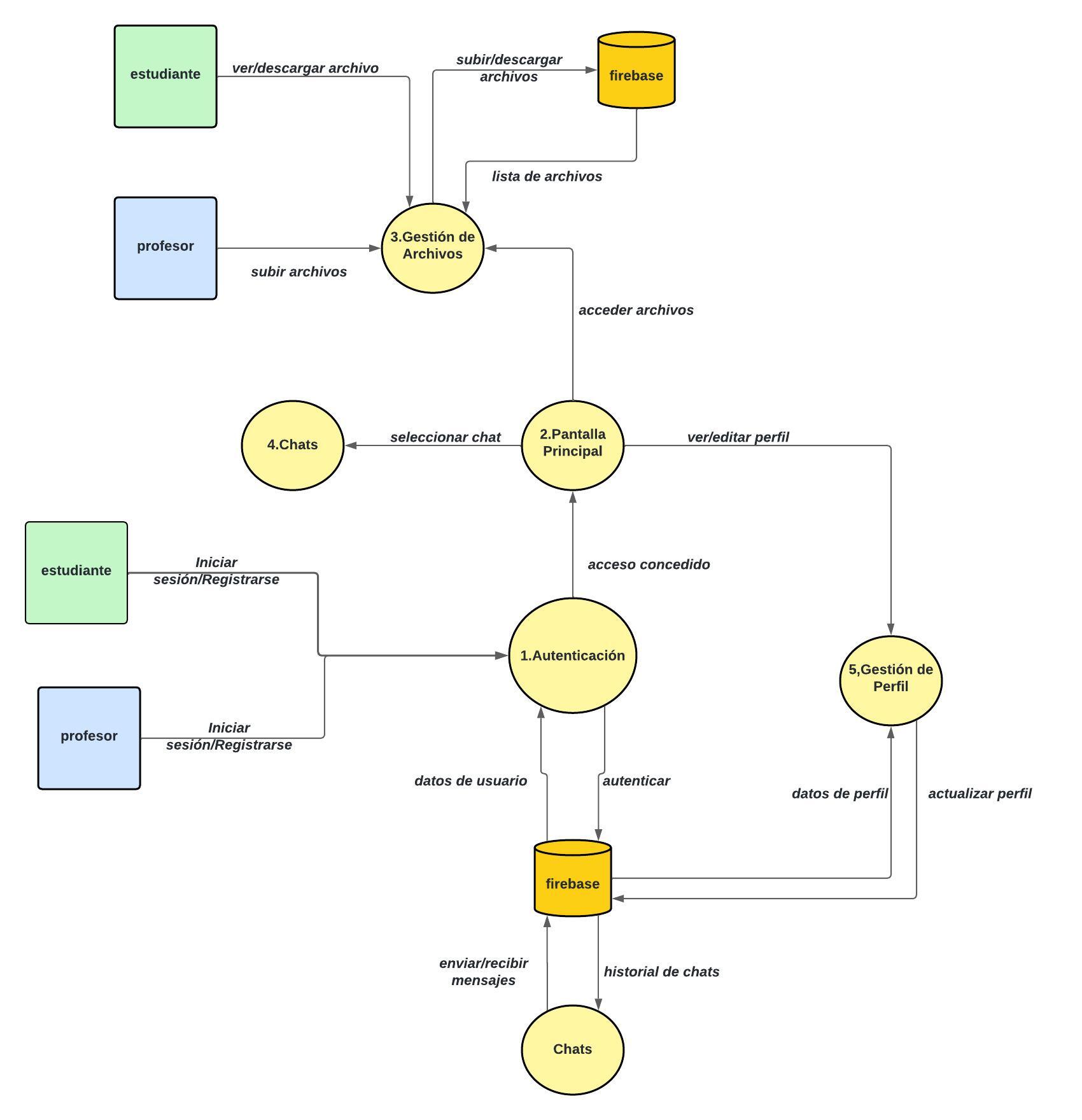
## **Herramientas de Depuración**

Aunque la aplicación puede visualizarse fácilmente con Expo Go, para debugging más avanzado, React Native incluye soporte para herramientas como React Native Debugger o Chrome DevTools, que permiten inspeccionar el estado de los componentes y hacer seguimiento del rendimiento o errores en la aplicación.

# **Diagrama DFD**

Este Diagrama de Flujo de Datos (DFD) representa la estructura y el funcionamiento de la aplicación Edushare, una plataforma educativa que facilita la interacción entre estudiantes y profesores.

**Figura 1**

Diagrama de datos(DFD)

**Nota:** A continuación se explica cada uno de los apartados

* Entidades Externas: estudiantes y profesores
* Procesos : autenticación , pantalla principal , gestión de archivos , chats , gestión de perfil.
* Almacén de datos : firebase que es el que se encarga de la autenticación , los datos de los archivos y usuarios.
* Flujo de datos(las flechas): muestran las interacciones entre las entidades , procesos y el almacén.

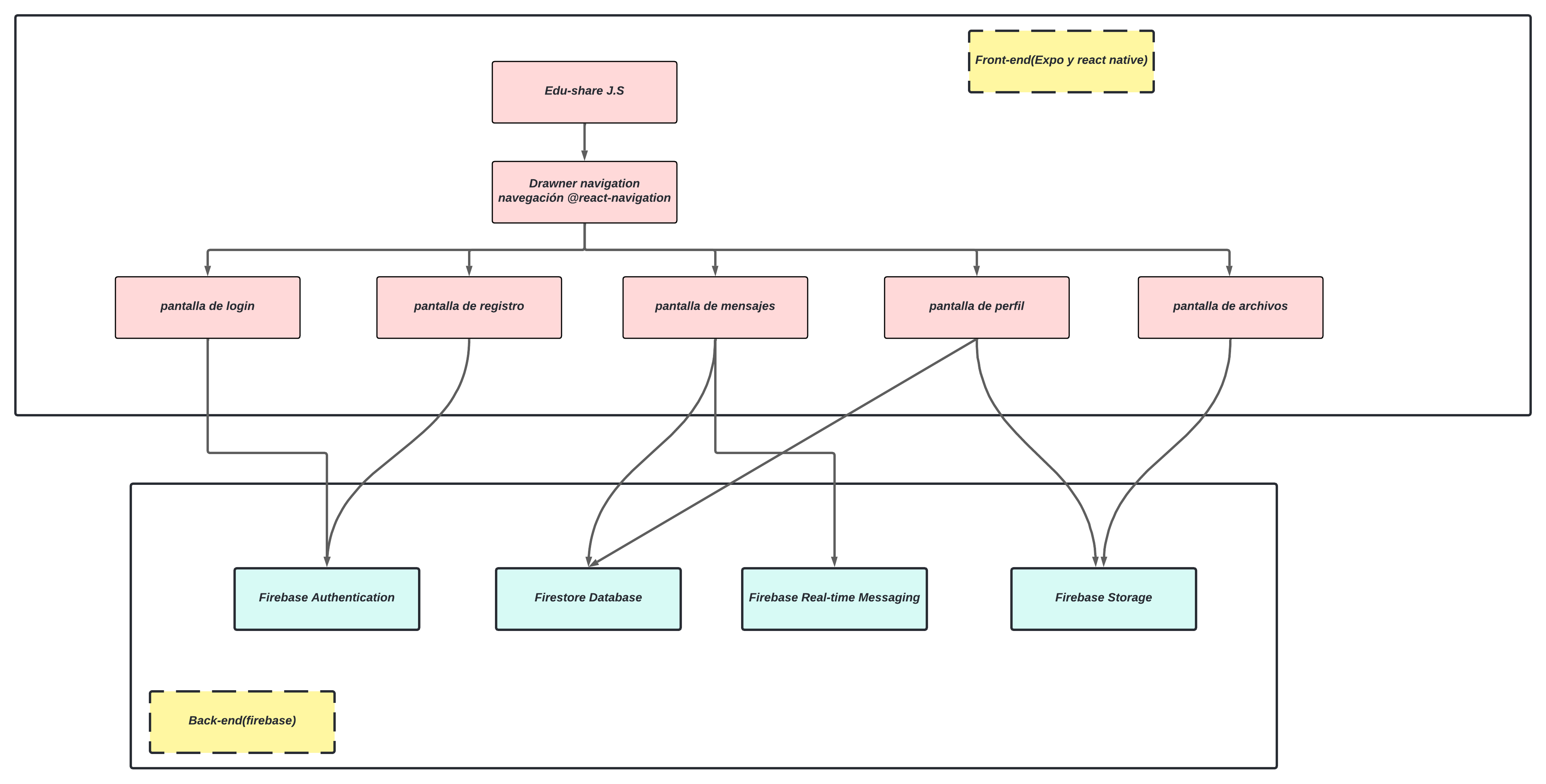
# **Arquitectura de Software**

Como para cada proyecto informático se necesita hacer una arquitectura , ya que es el diseño estructurado de los sistemas y aplicaciones para que sean eficientes. En este caso usaremos la arquitectura de:

**Cliente-servidor**

**Figura 2**

Imagen representativa del modelo con sus elementos claves además de una explicación del mismo.

****

***Frontend (React Native)***

**Drawer Navigator (@react-navigation).** Este es el sistema de navegación principal de la app. Utiliza el Drawer Navigator de la biblioteca @react-navigation, lo que permite a los usuarios acceder a diferentes secciones de la app deslizando desde el borde de la pantalla o a través de un icono de menú.

***Pantallas***

**Pantalla de Login**. Maneja la autenticación de usuarios.

**Pantalla de Registro.** Permite a nuevos usuarios crear una cuenta.

**Pantalla de Mensajes.** Interfaz para la funcionalidad de chat.

**Pantalla de Perfil.** Muestra y permite editar la información del usuario.

**Pantalla de Archivos.** Nueva adición para manejar la carga y descarga de archivos.

***Backend (Firebase)***

**Firebase Authentication.** Gestiona todo el proceso de autenticación de usuarios, incluyendo registro, inicio de sesión y gestión de sesiones.

**Firestore Database.** Base de datos NoSQL en tiempo real que almacena y sincroniza datos como perfiles de usuario, mensajes y metadatos de archivos.

**Firebase Storage.** Servicio para almacenar y recuperar archivos de usuario, como imágenes de perfil y archivos compartidos.

**Firebase Real-time Messaging**. Facilita la funcionalidad de mensajería en tiempo real para el chat.

# **Modelado de la BD**

A continuación se presentara el modelado de la base de datos del proyecto , este modelado está basado en una BD NoSQL específicamente en firebase por lo que la representación tendrá sus respectivas explicaciones para comprender cada parte.

En nuestro proyecto con firebase se usaran 2 modelados : clave valor para el caso de firebase authenthication , documental en el caso de firestore y se tendrá un sistema de guardado de archivos en Cloud Storage por lo que solo se dará una representación simple de las carpetas además de las reglas que se usaran.

## **Modelado Clave-Valor**

Es como un diccionario de datos gigante donde cada clave tiene un identificador único y cada clave está asociada a un valor que puede ser cualquier dato . Esto se usa en firebase authentication.

**Tabla 15**

|  |  |
| --- | --- |
| **Clave** | **Valor** |
| Identificador | graciaportillo8@gmail.com |
| UID de usuario | uhgotn9sNySBfv9QlaxaZzaW... |
| Proveedores | Externos como (Facebook,Gmail,Google juegos) |
| Fecha de creación | 6 sept 2024 |
| Fecha de acceso | 6 sept 2024 |

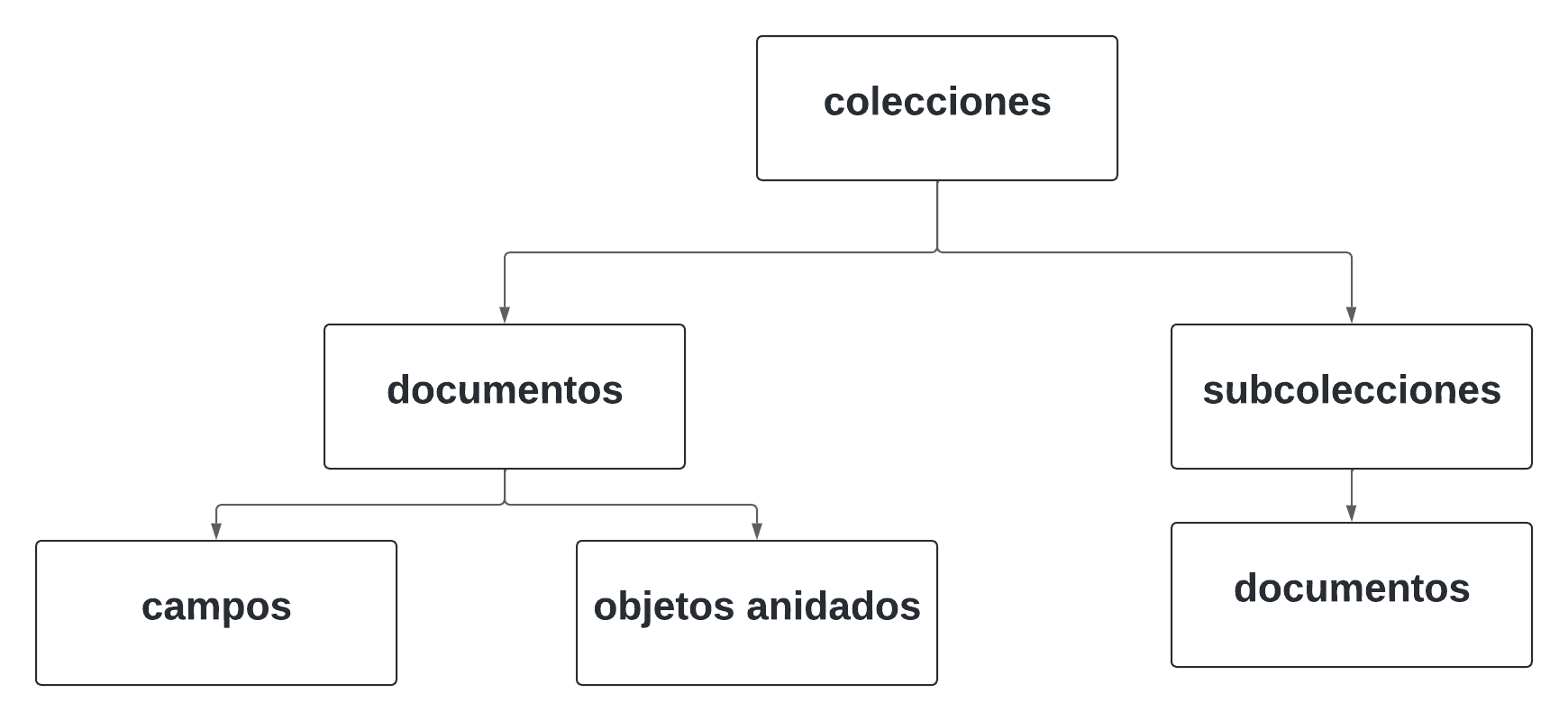
Representación de los datos.

## **Modelo documental**

A continuación se te presenta la estructura del modelo en firestore.

Figura 3

Estructura usada en las colecciones

****

**Colecciones.** Son los contenedores de los documentos.

**Documentos. S**on las unidades de almacenamiento dentro de la colección.

**Campos.** Son los pares clave-valor que contienen los datos dentro del documento. Un campo puede ser una cadena de texto, un número, un booleano hasta una lista.

**Objetos Anidados.** Son estructuras de datos complejas dentro de un documento, que pueden contener varios campos dentro de un solo campo.

**Subcolecciones.** Un documento puede contener una o más subcolecciones, que a su vez contienen más documentos. Esto permite una estructura jerárquica de datos, donde se pueden organizar los datos en partes más pequeñas y organizadas. Ideal para hacer menos consultas.

**Documentación de la BD en Firestore**

Aquí se explicaran los colecciones , documentos , campos ,subcolecciones , objetos anidados de nuestra BD.

**Tabla 16**

***Users.*** Almacena la información de los usuarios.

|  |
| --- |
| **uhgotn9sNySBfv9QlaxaZzaW** |
| email"eivlonfans@gmail.com"  name"Eivilona"  role"docente"  year*null* |

**Table 17**

***Chats—1,2,3,global--Messages .*** Messages almacena los textos que se envían , este es una subcoleccion dentro de los documentos de chats

|  |
| --- |
| **ws6nk2FVD0bQhnTBD12K** |
| createdAt18 de octubre de 2024, 10:13:53 p.m. UTC-6  displayName"ChakiBv-uwu"  **replyTo**  displayName"Merlin"  id"SHAemvgQBFpx9mIZeWt2"  text"Calla \*\*\*\*\*\*"  text"Zi"  uid"qsXQOgil5fM2WyAem3RTqWr6xak1" |

**Tabla 18**

***Files.*** lmacena los archivos que se suben

|  |
| --- |
| **ws6nk2FVD0bQhnTBjiieei** |
| createdAt17 de octubre de 2024, 12:40:52 p.m. UTC-6  description"Shingas "  name"IMG-20241017-WA0026.jpg"  uploadedBy"César "  url"https://firebasestorage.googleapis.com/v0/b/intentobueno-8bc83.appspot.com/o/files%2FIMG-20241017-WA0026.jpg?alt=media&token=2f77dfe4-ee74-4310-b561-966e66a14247"  year"tercer" |

# **Descripción de las tablas de la base de datos**

## **Usuarios (Users)**

* email: Correo electrónico del usuario.
* name: Nombre completo del usuario.
* role: Rol del usuario (puede ser 'docente' o 'alumno').
* year: Año de estudio (solo aplicable si el rol es 'alumno').

## **Archivos (Files)**

* name: Nombre del archivo.
* url: URL del archivo en Firebase.
* description: Descripción del archivo.
* uploadedBy: Nombre del usuario que subió el archivo.
* createdAt: Fecha y hora en que se subió el archivo.
* year: Año asociado al archivo(lo que usamos para filtrarlos).

## **Mensajes (Messages)**

* text: Contenido del mensaje.
* createdAt: Fecha y hora en que se envió el mensaje.
* displayName: Nombre del usuario que envió el mensaje.
* replyTo: Mensaje al que se está respondiendo.
* uid: Identificador único del usuario.

# **Definición y descripción de variables usadas en el sistema.**

A continuación, se presenta una tabla con la definición y descripción de las variables usadas en cada uno de los archivos clave del sistema

**Tabla 19**

loginScreen.js

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Nombre de la variable** | **Tipo de dato** | **Descripción** | **Alcance** | **Valores posibles** |
| email | string | Almacena el correo electrónico ingresado por el usuario en el campo de email. | Local (estado) | Correo electrónico en formato texto |
| password | string | Almacena la contraseña ingresada por el usuario en el campo de contraseña. | Local (estado) | Contraseña en formato texto |
| loading | boolean | Indica si el sistema está procesando la solicitud de inicio de sesión, deshabilitando elementos de la interfaz mientras está en true. | Local (estado) | true, false |
| navigation | object | Objeto proporcionado por React Navigation para navegar entre las pantallas. | Parámetro de función | Métodos de navegación disponibles |
| netInfo | object | Contiene la información de la red obtenida mediante la librería @react-native-community/netinfo para verificar la conexión a internet. | Local (dentro de handleLogin) | Objeto con detalles de la conexión |
| errorMessage | string | Almacena el mensaje de error que se mostrará al usuario en caso de un fallo en el inicio de sesión. | Local (dentro de handleLogin) | Mensaje de error personalizado |

**Tabla 20**

RegisterScreen.js

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Nombre de la variable** | **Tipo de dato** | **Descripción** | **Alcance** | **Valores posibles** |
| role | string | Almacena el rol del usuario (docente o alumno) que se pasa como parámetro desde la pantalla anterior. | Parámetro (route) | "docente", "alumno" |
| name | string | Almacena el nombre completo del usuario ingresado en el campo de texto. | Local (estado) | Nombre en formato texto |
| email | string | Almacena el correo electrónico ingresado por el usuario en el campo correspondiente. | Local (estado) | Correo electrónico en formato texto |
| password | string | Almacena la contraseña ingresada por el usuario en el campo correspondiente. | Local (estado) | Contraseña en formato texto |
| year | string | Almacena el año seleccionado por el usuario (si es alumno), utilizando un componente Picker. | Local (estado) | "1", "2", "3" (Año de estudio) |
| code | string | Almacena el código de docente ingresado para validación. Solo es relevante si el usuario es docente. | Local (estado) | Código de docente en formato texto |
| userCredential | object | Contiene la información del usuario autenticado tras crear una cuenta con Firebase. | Local (dentro de handleRegister) | Objeto con datos del usuario |

**Tabla 21**

Homescreen.js

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Nombre de la variable** | **Tipo de dato** | **Descripción** | **Alcance** | **Valores posibles** |
| chatNames | object | Contiene los nombres de los chats asociados a cada año. | Local (constante) | Objeto con claves: global, year1, year2, year3 |
| chatRooms | array | Almacena los permisos del usuario en forma de salas de chat a las que puede acceder. | Local (estado) | Array de nombres de salas de chat |
| loading | boolean | Indica si la aplicación está en el estado de carga al obtener los permisos de usuario. | Local (estado) | true, false |
| menuVisible | boolean | Controla la visibilidad del menú lateral en la pantalla principal. | Local (estado) | true, false |
| menuAnimation | Animated.Value | Almacena el valor de animación para el menú (abierto o cerrado). | Local (estado) | Valor entre 0 y 1 |
| iconScale | Animated.Value | Controla la escala animada de los iconos en el menú. | Local (estado) | Valor de escala (entre 0 y 1.1) |
| isLoggingOut | boolean | Controla si el usuario está cerrando sesión (para evitar múltiples cierres de sesión simultáneos). | Local (estado) | true, false |
| netInfo | object | Almacena el estado de la conexión de red obtenido de NetInfo. | Temporal (dentro de funciones) | Objeto con propiedades de conexión |

**Tabla 22**

Filesscreen.js

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Nombre de la variable** | **Tipo de dato** | **Descripción** | **Alcance** | **Valores posibles** |
| files | Array | Lista de archivos recuperados desde Firestore. | Local | Lista de objetos de archivo |
| description | String | Descripción del archivo que se está subiendo. | Local | Cualquier texto |
| year | String | Año asociado al archivo que se está subiendo. | Local | "primer", "segundo", "tercer", "" |
| isModalVisible | Boolean | Estado que indica si el modal para subir archivos está visible. | Local | true, false |
| selectedFile | Object | Archivo seleccionado para cargar. | Local | Objeto con uri y name |
| isUploading | Boolean | Estado que indica si se está subiendo un archivo. | Local | true, false |
| currentUser | Object | Usuario actualmente autenticado. | Local | Objeto de usuario de Firebase |
| isDocente | Boolean | Indica si el usuario actual tiene el rol de docente. | Local | true, false |
| searchYear | String | Año seleccionado para filtrar los archivos mostrados. | Local | "primer", "segundo", "tercer", "" |
| years | Array | Lista de opciones de años para el filtro. | Local | [{ label: 'Todos', value: '' }, ...] |

**Tabla 23**

**Profilescreen.js**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Nombre de la variable** | **Tipo de dato** | **Descripción** | **Alcance** | **Valores posibles** |
| userData | Object | Datos del usuario autenticado recuperados de Firestore. | Local | Objeto con datos del usuario |
| name | String | Nombre del usuario que se está editando. | Local | Cualquier texto |

**Tabla 24**

**Chatscreen.js**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Nombre de la variable** | **Tipo de dato** | **Descripción** | **Alcance** | **Valores posibles** |
| chatRoom | String | Identificador de la sala de chat actual. | Propiedades de la ruta | Cualquier identificador de sala de chat |
| messages | Array | Lista de mensajes obtenidos de Firestore. | Local | Arreglo de objetos de mensaje |
| newMessage | String | Mensaje nuevo que el usuario está escribiendo. | Local | Cualquier texto |
| menuVisible | Boolean | Indica si el menú está visible o no. | Local | true o false |
| menuAnimation | Animated.Value | Valor animado que controla la animación del menú. | Local | Valor numérico entre 0 y 1 |
| loading | Boolean | Indica si se están cargando mensajes. | Local | true o false |
| displayName | String | Nombre de usuario que se muestra en los mensajes. | Local | Cualquier texto |
| replyTo | Object | Mensaje al que se está respondiendo. | Local | Objeto de mensaje o null |
| flatListRef | Ref | Referencia a la lista de mensajes para desplazamiento. | Local | Referencia a FlatList |
| highlightedMessageId | String | ID del mensaje destacado al responder. | Local | null o ID de un mensaje |
| atTop | Boolean | Indica si se encuentra en la parte superior de la lista de mensajes. | Local | true o false |

**Tabla 25**

**Passwordresetscreen.js**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Nombre de la variable** | **Tipo de dato** | **Descripción** | **Alcance** | **Valores posibles** |
| email | string | Almacena el correo electrónico ingresado por el usuario para la recuperación de la contraseña. | Local (estado) | Cualquier dirección de correo |
| setEmail | function | Función para actualizar el estado de email. | Local (estado) | - |
| handlePasswordReset | function | Función asincrónica que maneja el proceso de restablecimiento de contraseña. | Local (método) | - |
| navigation | object | Objeto que permite la navegación entre pantallas en la aplicación. | Prop de componente | - |
| styles | object | Objeto que contiene los estilos para el componente, importado de un archivo de estilos. | Local (importado) | - |

**Tabla 26**

**Messageitem.js**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Nombre de la variable** | **Tipo de dato** | **Descripción** | **Alcance** | **Valores posibles** |
| message | object | Objeto que representa el mensaje. | Prop de componente | - |
| isOwnMessage | boolean | Indica si el mensaje pertenece al usuario actual. | Prop de componente | true o false |
| onDelete | function | Función que se ejecuta al presionar el botón de eliminar el mensaje. | Prop de componente | - |
| onReply | function | Función que se ejecuta al presionar el botón de responder el mensaje. | Prop de componente | - |
| onSelectReply | function | Función que se ejecuta al seleccionar un mensaje al que se está respondiendo. | Prop de componente | - |
| highlightedMessageId | string | ID del mensaje que está resaltado. | Prop de componente | - |
| styles | object | Objeto que contiene los estilos para el componente. | Local (importado) | - |

# **Diccionario de datos**

Este diccionario de datos documenta la estructura y los elementos de la base de datos de la aplicación Edushare, que utiliza Firebase para su backend.

**Tabla 27**

Colección: Users

|  |  |  |  |
| --- | --- | --- | --- |
| Nombre del Campo | Tipo de Dato | Descripción | Ejemplo |
| email | string | Correo electrónico del usuario | "juan@gmail.com" |
| name | string | Nombre completo del usuario | "Edwin" |
| role | string | Rol del usuario (docente o alumno) | "alumno" |
| year | string | Año de estudio del usuario | "3" |

**Tabla 28**

Colección: Files

|  |  |  |  |
| --- | --- | --- | --- |
| Nombre del Campo | Tipo de Dato | Descripción | Ejemplo |
| createdAt | timestamp | Fecha y hora en que se subió el archivo | 17 de octubre de 2024, 12:40:52 p.m. UTC-6 |
| description | string | Descripción del archivo | "Mysql" |
| name | string | Nombre del archivo subido | "IMG-20241017-WA0026.jpg" |
| uploadedBy | string | Nombre del usuario que subió el archivo | "César" |
| url | string | URL donde se almacena el archivo en Firebase | "https://firebasestorage.googleapis.com/v0/b/intentobueno-8bc83.appspot.com/o/files%2FIMG-20241017-WA0026.jpg?alt=media&token=2f77dfe4-ee74-4310-b561-966e66a14247" |
| year | string | Año asociado al archivo | "tercer" |

**Tabla 29**

Colección Chats. Esta colección se compone por sub colecciones llamadas

* Global
* Year 1
* Year 2
* Year 3

Cada una de estas sub colecciones contiene otra llamada Messages , la cual está conformada por

|  |  |  |  |
| --- | --- | --- | --- |
| **Nombre del Campo** | **Tipo de Dato** | **Descripción** | **Ejemplo** |
| createdAt | timestamp | Fecha y hora en que se envió el mensaje | 18 de octubre de 2024, 10:13:53 p.m. UTC-6 |
| displayName | string | Nombre del usuario que envió el mensaje | "ChakiBv-uwu" |
| replyTo | object | Mensaje al que se está respondiendo | { displayName: "Merlin La perreadora", id: "SHAemvgQBFpx9mIZeWt2", text: "Calla \*\*\*\*\*\*", uid: "qsXQOgil5fM2WyAem3RTqWr6xak1" } |
| text | string | Contenido del mensaje | "Zi" |
| uid | string | Identificador único del usuario que envió el mensaje | "qsXQOgil5fM2WyAem3RTqWr6xak1" |