**JEGYZŐKÖNYV**

**Mobil programozás alapjai**

**LoginApp**

**Készítette: Kozsup Krisztián**

**Neptunkód: IL8SLQ Dátum: 2024. november. 11.**

**Miskolc, 2024.**

Tartalom

[**1.Projekt összefoglaló** 3](#_Toc182239908)

[**2. Alapvető importok és Firebase inicializálása** 5](#_Toc182239909)

[**3. Firebase konfiguráció és inicializálás** 5](#_Toc182239910)

[**4. Felhasználói bejelentkezési/regisztrációs felület (AuthScreen komponens)** 6](#_Toc182239911)

[**5. Autentikált felhasználói felület (AuthenticatedScreen komponens)** 7](#_Toc182239912)

[**6. Alap App komponens (Az alkalmazás beállítása és hitelesítési folyamat kezelése)** 9](#_Toc182239913)

[**7. Stílusok definiálása** 10](#_Toc182239914)

[**8. Megjelenítéssel kapcsolatos kódrészletek dokumentálása** 11](#_Toc182239915)

[**9.Végeredmény** 14](#_Toc182239916)

### **1.Projekt összefoglaló**

Ez a projekt egy React Native Expo alkalmazás, amely egy Firebase-alapú bejelentkezési rendszert valósít meg. A felhasználók regisztrálhatnak, bejelentkezhetnek, hozzáadhatnak jegyzeteket a Firebase Firestore adatbázishoz, és törölhetik saját fiókjukat. Az alkalmazás megjelenésének javítása érdekében két kép (bg.jpg és logo.jpg) lett beágyazva a háttérhez és logóhoz.

**Használt technológiák és eszközök**

1. **React Native és Expo:**
   * A React Native biztosítja a mobil alkalmazások fejlesztésének alapjait.
   * Az Expo egy fejlesztői környezet, amely egyszerűsíti a React Native alkalmazások létrehozását és futtatását.
2. **Firebase:**
   * Firebase Authentication: Lehetővé teszi a felhasználók regisztrációját, bejelentkezését és hitelesítését.
   * Firestore Database: A Firebase felhő alapú adatbázisa, amelyet a jegyzetek tárolására használunk.
3. **Fájlrendszer:**
   * assets/images: Ebben a mappában találhatóak a vizuális elemek, például a háttér (bg.jpg) és a logó (logo.jpg) képfájlok.
   * App.js: A fő komponensfájl, amely az alkalmazás logikáját és felhasználói interfészét tartalmazza.

**Fájlstruktúra magyarázata**

* assets/images: Képek tárolására szolgál, amelyeket az alkalmazásban használunk a háttérhez és a logóhoz.
* App.js: Az alkalmazás főfájlja, amely az összes React komponens logikáját, stílusát és Firebase kapcsolatait kezeli.
* package.json: Tartalmazza a projekt függőségeit és alapvető konfigurációit.

babel.config.js: A Babel konfigurációját tartalmazza, amely segíti a kód fordítását a megfelelő formátumba.

**2. Alapvető importok és Firebase inicializálása**

* import React, { useState, useEffect } from 'react';
* import { View, Text, TextInput, TouchableOpacity, StyleSheet, ScrollView, Modal, FlatList } from 'react-native';
* import { initializeApp } from '@firebase/app';
* import { getAuth, createUserWithEmailAndPassword, signInWithEmailAndPassword, onAuthStateChanged, signOut, deleteUser } from '@firebase/auth';
* import { getFirestore, collection, addDoc, getDocs, query, where, deleteDoc, doc } from '@firebase/firestore';
* import { ImageBackground } from 'react-native';
* import { Image } from 'react-native';
* import backgroundImage from './images/bg.jpg';
* import logoImage from './images/logo.jpg';
* **React Native komponensek**: A szükséges React Native komponensek importálása (View, Text, TextInput, stb.) lehetővé teszi a felhasználói felület felépítését és interaktív elemek hozzáadását.
* **Firebase modulok**: A Firebase autentikáció (getAuth, createUserWithEmailAndPassword, stb.) és Firestore adatbázis (getFirestore, addDoc, deleteDoc, stb.) funkciók biztosítják a felhasználói hitelesítést és a jegyzetek kezelését.

**3. Firebase konfiguráció és inicializálás**

* const firebaseConfig = {
* apiKey: "AIzaSyBnE18Z93oLyVI8DbYhR2--1-v\_tYyCsAQ",
* authDomain: "loginapp-kk.firebaseapp.com",
* projectId: "loginapp-kk",
* storageBucket: "loginapp-kk.firebasestorage.app",
* messagingSenderId: "714527906588",
* appId: "1:714527906588:web:493d75338a69fa36ccf8d9"
* };
* const app = initializeApp(firebaseConfig);
* const auth = getAuth(app);
* const firestore = getFirestore(app);
* **Firebase konfiguráció**: A firebaseConfig objektumban a projekthez szükséges kulcsok találhatók (API-kulcs, projektazonosító stb.).
* **App inicializálása**: initializeApp(firebaseConfig) meghívása elindítja a Firebase alkalmazást, lehetővé téve a hitelesítés és adatbázis funkciók elérését.

**4. Felhasználói bejelentkezési/regisztrációs felület (AuthScreen komponens)**

* const AuthScreen = ({ email, setEmail, password, setPassword, isLogin, setIsLogin, handleAuthentication }) => {
* return (
* <View style={styles.authContainer}>
* <Text style={styles.title}>{isLogin ? 'Sign In' : 'Sign Up'}</Text>
* <TextInput
* style={styles.input}
* value={email}
* onChangeText={setEmail}
* placeholder="Email"
* autoCapitalize="none"
* />
* <TextInput
* style={styles.input}
* value={password}
* onChangeText={setPassword}
* placeholder="Password"
* secureTextEntry
* />
* <TouchableOpacity style={styles.button} onPress={handleAuthentication}>
* <Text style={styles.buttonText}>{isLogin ? 'Sign In' : 'Sign Up'}</Text>
* </TouchableOpacity>
* <Text style={styles.toggleText} onPress={() => setIsLogin(!isLogin)}>
* {isLogin ? 'Need an account? Sign Up' : 'Already have an account? Sign In'}
* </Text>
* </View>
* );
* };
* **Komponens leírása**: A AuthScreen komponens egy egyszerű bejelentkezési és regisztrációs felületet biztosít.
  + TextInput: A felhasználónak itt kell megadnia az email-címét és jelszavát.
  + TouchableOpacity: Ez a gomb indítja el a hitelesítést a handleAuthentication függvény hívásával.
  + **Regisztráció/bejelentkezés váltása**: A setIsLogin(!isLogin) kapcsolóval a felhasználó átválthat bejelentkezési és regisztrációs módok között.

**5. Autentikált felhasználói felület (AuthenticatedScreen komponens)**

* const AuthenticatedScreen = ({ user, handleLogout, handleDeleteAccount }) => {
* const [note, setNote] = useState('');
* const [notes, setNotes] = useState([]);
* const userId = user.uid;
* const fetchNotes = async () => {
* const q = query(collection(firestore, 'notes'), where('userId', '==', userId));
* const querySnapshot = await getDocs(q);
* const notesData = querySnapshot.docs.map((doc) => ({ id: doc.id, ...doc.data() }));
* setNotes(notesData);
* };
* useEffect(() => {
* fetchNotes();
* }, [userId]);
* // Add a new note to Firestore
* const handleAddNote = async () => {
* if (note.trim()) {
* await addDoc(collection(firestore, 'notes'), { text: note, userId });
* setNote(''); // Clear the input field
* fetchNotes(); // Refresh the notes list
* }
* };
* const handleDeleteNote = async (noteId) => {
* await deleteDoc(doc(firestore, 'notes', noteId));
* fetchNotes(); // Refresh the notes list after deletion
* };
* // Delete account and associated notes
* const deleteAccount = async () => {
* try {
* const userRef = auth.currentUser;
* const q = query(collection(firestore, 'notes'), where('userId', '==', userId));
* const querySnapshot = await getDocs(q);
* // Delete each note from Firestore
* await Promise.all(querySnapshot.docs.map((noteDoc) => deleteDoc(doc(firestore, 'notes', noteDoc.id))));
* // Delete user account
* await deleteUser(userRef);
* } catch (error) {
* console.log('Error deleting account:', error);
* }
* };
* **fetchNotes függvény**:
  + Lekéri az aktuális felhasználóhoz tartozó jegyzeteket a Firestore adatbázisból a where feltétel használatával.
* **Jegyzet hozzáadása és törlése**:
  + handleAddNote: Ha a felhasználó beír egy jegyzetet, ez a függvény a Firestore-ban tárolja azt.
  + handleDeleteNote: Törli a kiválasztott jegyzetet a Firestore-ból.
* **Fiók törlése**:
  + deleteAccount: A függvény törli az összes felhasználóhoz tartozó jegyzetet, majd magát a felhasználói fiókot is.

**6. Alap App komponens (Az alkalmazás beállítása és hitelesítési folyamat kezelése)**

* export default App = () => {
* const [email, setEmail] = useState('');
* const [password, setPassword] = useState('');
* const [user, setUser] = useState(null);
* const [isLogin, setIsLogin] = useState(true);
* const [error, setError] = useState(null);
* const [isAlertVisible, setIsAlertVisible] = useState(false);
* useEffect(() => {
* const unsubscribe = onAuthStateChanged(auth, (user) => {
* setUser(user);
* });
* return () => unsubscribe();
* }, []);
* const handleAuthentication = async () => {
* try {
* if (isLogin) {
* await signInWithEmailAndPassword(auth, email, password);
* } else {
* await createUserWithEmailAndPassword(auth, email, password);
* }
* } catch (error) {
* console.log('Authentication error:', error.message);
* setError(error.message);
* setIsAlertVisible(true);
* }
* };
* const closeAlert = () => {
* setIsAlertVisible(false);
* setError(null);
* };
* **Felhasználói állapotok**: A user változóban tárolja az aktuális bejelentkezett felhasználót, míg a isLogin a bejelentkezési vagy regisztrációs módot határozza meg.
* **Ellenőrzés useEffect-tel**: Az onAuthStateChanged függvény figyeli a bejelentkezési állapot változásait, és ennek megfelelően frissíti a user állapotot.
* **handleAuthentication függvény**: A felhasználó hitelesítését végzi el, akár bejelentkezéssel, akár regisztrációval.
* **Hibaüzenet megjelenítése (Modal)**: Hiba esetén a Modal jeleníti meg az üzenetet, amelyet a setError beállítással és closeAlert függvénnyel lehet kezelni.

**7. Stílusok definiálása**

* const styles = StyleSheet.create({
* container: {
* flexGrow: 1,
* justifyContent: 'center',
* alignItems: 'center',
* padding: 16,
* },
* authContainer: { … },
* **Alap stílusok**: Meghatározza az alkalmazás főbb elemeinek stílusát, beleértve a konténereket, gombokat, szövegdobozokat és figyelmeztető ablakokat.
* **Testreszabott stílusok**: Az alkalmazás modern és letisztult megjelenést kap a bejelentkezési képernyőn és a jegyzetek megjelenítésében is.

### **8. Megjelenítéssel kapcsolatos kódrészletek dokumentálása**

**Fő kép háttér megjelenítése (App.js-ben)**

<ImageBackground source={backgroundImage} style={styles.background}>

      <ScrollView contentContainerStyle={styles.container}>

        <Image source={logoImage} style={styles.logo} />

{/\* Itt jelennek meg a bejelentkezési vagy jegyzetekkel kapcsolatos komponensek \*/}

</ScrollView>

</ImageBackground>

* **ImageBackground komponens**: Ez a háttérképet jeleníti meg az egész alkalmazás háttereként. A style={styles.background} biztosítja, hogy a kép teljesen kitöltse a képernyőt.
* **ScrollView**: A felület görgethető, hogy kisebb képernyőkön is könnyen használható legyen. Az contentContainerStyle segítségével beállítjuk a középre igazítást.
* **Image komponens**: A logó az alkalmazás tetején jelenik meg, és a styles.logo stílust alkalmazva megfelelő méretben és igazítással.

**AuthScreen (Bejelentkezés és Regisztráció) felület**

* <View style={styles.authContainer}>
* <Text style={styles.title}>{isLogin ? 'Sign In' : 'Sign Up'}</Text>
* <TextInput
* style={styles.input}
* value={email}
* onChangeText={setEmail}
* placeholder="Email"
* autoCapitalize="none"
* />
* <TextInput
* style={styles.input}
* value={password}
* onChangeText={setPassword}
* placeholder="Password"
* secureTextEntry
* />
* <TouchableOpacity style={styles.button} onPress={handleAuthentication}>
* <Text style={styles.buttonText}>{isLogin ? 'Sign In' : 'Sign Up'}</Text>
* </TouchableOpacity>
* <Text style={styles.toggleText} onPress={() => setIsLogin(!isLogin)}>
* {isLogin ? 'Need an account? Sign Up' : 'Already have an account? Sign In'}
* </Text>
* </View>
* **authContainer stílus**: Az autentikációs komponenseket tartalmazó nézet, amely egy lekerekített, árnyékolt dobozként jelenik meg, fehér háttérrel, és központilag igazítva.
* **Text komponens**: A title stílusú szöveg, amely megjeleníti, hogy a felhasználó éppen bejelentkezik vagy regisztrál, dinamikusan változik az isLogin változó értékétől függően.
* **TextInput komponensek**: Két beviteli mező, az egyik az emailhez, a másik a jelszóhoz. A secureTextEntry tulajdonság a jelszó mezőnél gondoskodik róla, hogy a jelszó karakterei rejtve maradjanak.
* **TouchableOpacity**: Ez a gomb funkcióval rendelkezik. A felhasználói művelet alapján (bejelentkezés vagy regisztráció) változik a gomb szövege.
* **toggleText**: Egy választható szöveg, amely segítségével a felhasználó válthat a Bejelentkezés és a Regisztráció között.

**AuthenticatedScreen (Jegyzetek kezelése és kijelentkezés) felület**

* <View style={styles.authContainer}>
* <Text style={styles.title}>Welcome, {user.email}</Text>
* <TextInput
* style={styles.input}
* placeholder="Enter a note"
* value={note}
* onChangeText={setNote}
* />
* <TouchableOpacity style={styles.button} onPress={handleAddNote}>
* <Text style={styles.buttonText}>Save Note</Text>
* </TouchableOpacity>
* <FlatList
* data={notes}
* keyExtractor={(item) => item.id}
* renderItem={({ item }) => (
* <View style={styles.noteContainer}>
* <Text style={styles.noteText}>{item.text}</Text>
* <TouchableOpacity style={styles.deleteButton} onPress={() => handleDeleteNote(item.id)}>
* <Text style={styles.deleteButtonText}>Delete</Text>
* </TouchableOpacity>
* </View>
* )}
* />
* <TouchableOpacity style={styles.logoutButton} onPress={handleLogout}>
* <Text style={styles.buttonText}>Logout</Text>
* </TouchableOpacity>
* <TouchableOpacity style={styles.deleteAccountButton} onPress={deleteAccount}>
* <Text style={styles.buttonText}>Delete Account</Text>
* </TouchableOpacity>
* </View>
* **Üdvözlő szöveg**: A felhasználó e-mail címét megjelenítő Text elem, amely személyesebb élményt nyújt a felhasználónak.
* **Jegyzet beviteli mező és gomb**:
  + Az input stílusú TextInput mezőbe írható az új jegyzet szövege.
  + A button stílusú TouchableOpacity segítségével a "Save Note" gomb aktiválja a jegyzet mentését.
* **FlatList komponens**: A jegyzeteket listázza ki, ahol minden egyes jegyzet egy külön konténerben jelenik meg a noteContainer stílussal.
  + Minden egyes jegyzet tartalmazza a jegyzet szövegét (noteText stílus), valamint egy deleteButton nevű törlés gombot, amely lehetőséget ad a jegyzet eltávolítására.
* **Kijelentkezés és Fiók törlése gombok**:
  + logoutButton és deleteAccountButton gombok, piros háttérrel, egyértelmű és látványos lehetőséget biztosítanak a kijelentkezésre és a fiók törlésére.

**Hibaüzenet Modal**

* Modal transparent={true} visible={isAlertVisible} animationType="slide">
* <View style={styles.overlay}>
* <View style={styles.alertBox}>
* <Text style={styles.alertMessage}>{error}</Text>
* <TouchableOpacity style={styles.alertButton} onPress={closeAlert}>
* <Text style={styles.buttonText}>Close</Text>
* </TouchableOpacity>
* </View>
* </View>
* </Modal>
* **Modal**: Az isAlertVisible állapottól függően jelenik meg. Átlátszó háttérrel, és "slide" animációval jelenik meg, amikor hiba történik.
* **overlay stílus**: Átlátszó fekete hátteret biztosít a Modal körül, hogy a felhasználó figyelmét a központi hibaüzenetre irányítsa.
* **alertBox és alertMessage**: A hibaüzenet megjelenését biztosítja. Az üzenet az error változó értékétől függően dinamikusan változik.
* **alertButton**: Egy kék háttérszínű gomb, amellyel bezárható a hibaüzenet Modal.

### **9.Végeredmény**

A képen szöveg, képernyőkép, multimédia, szoftver látható

Automatikusan generált leírás

A képen szöveg, képernyőkép, Mobiltelefon, multimédia látható

Automatikusan generált leírás