

Universitatea de Vest din Timișoara

ME – proiect individual

Sudent:

Pop Raluca Daniela

Facultatea de Matematică și Informatică

Specializarea Informatică, anul II

# Contents

Contents.....	2
Motivația alegerii proiectului/ Cui se adresează.....	3
Tehnologii folosite: .....	4
Autentificare si inregistrare:	

## Motivația alegerii proiectului/ Cui se adresează

Tema aplicației “ME” este una actuală, în ultimul timp oamenii au început să prezinte un interes deosebit în ceea ce privește dezvoltarea lor personală și profesională, astfel, regăsim tot mai des pe rețele sociale grupuri sau profile specifice care promovează procese de evoluție.

Domeniu de aplicabilitate proiect: social, inspirație, creație, dezvoltare personală.

Această aplicație este dedicată în special fetelor/femeilor care caută o sursă de motivație sau inspirație, un loc în care găsească persoane cu care împărtășesc aceleași experiențe și valori.

Descriere generală: Aplicația “ME” are ca scop dezvoltarea personală, atât din punct de vedere spiritual, cât și profesional. Fiecare utilizator își creează un profil și poate alege unul din cele două roluri active, cel de scriitor, sau cel de cititor. Rolul de scriitor permite utilizatorului să încare conținut în aplicație în una din cele patru secțiuni prezentate. Fiecare secțiune are un scop bine definit și o temă specifică, astfel încât în fiecare secțiune se vor regăsi articole specifice. Rolul de cititor permite utilizatorului să acceseze conținutul secțiunilor, și profilele scriitorilor pentru a putea lua legătura cu aceștia.

### Tehnologii folosite:

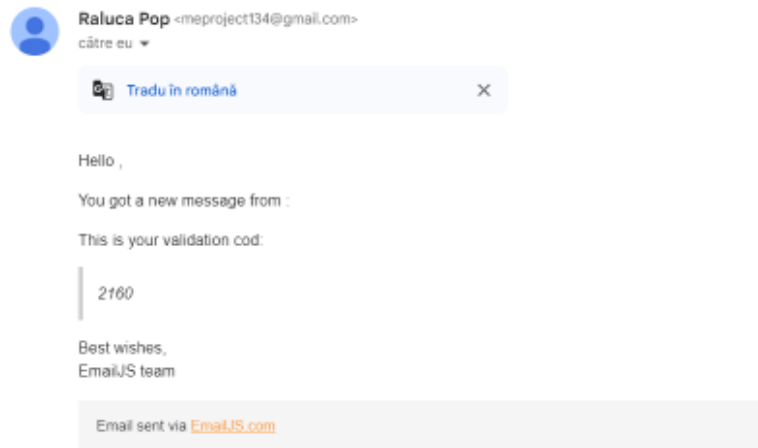
Aplicatia a fost realizata cu react-native (framework de JavaScript) si Expo.

Pentru stocarea si securitatea datelor am folosit Firebase: (pentru autentificare s-a folosit “Authentication Firebase”, iar pentru stocarea altor informatii despre utilizatori si a povestilor se foloseste “Firestore” – o tabela pentru utilizatori si o tabela pentru povesti).

Pentru transmiterea de e-mail-uri utilizatorilor s-a folosit EmailJS (un API folosit pentru a transmite e-mail-uri direct catre user)

## Autentificare si inregistrare

Crearea contului de utilizator presupune introducerea unei adrese de e-mail si a unei parole. Pentru crearea contului se realizeaza o verificare a adresei de e-mail prin introducerea unui cod de validare pe care utilizatorul il primeste pe adresa de e-mail introdusa.



In momentul in care utilizatorul introduce codul corect, contul acestuia v-a fi creat. Fiecare utilizator are un “uid” – user id prin care este identificat unic in baza de date. De asemenea, se adauga un obiect si tabela “users” care contine urmatoarele attribute: userId- id-ul user-ului din Authentication, e-mail acestuia, o descriere a contului(care are valoarea initiala “profile description” si data la care a fost realizat contul).

Astfel, dupa introducerea e-mail-ului si a parolei, userul primeste un email:

```
const generatedCode = Math.floor(1000 + Math.random() * 9000);

const sendEmail = async () => {
  try {
    const templateParams = {
      code: generatedCode,
      email: email,
    };
  }
```

```

const response = await emailjs.send(
  'service_xhrfr5c',
  'template_6zs0u2a',
  templateParams,
  'M2nZ6qEBEoijAaW4m'
);

console.log('Email sent successfully:', response);
} catch (error) {
  console.error('Error sending email:', error);
}
};

```

Utilizatorul va fi redirectionat catre o alta pagina unde trebuie sa introduce codul primit pe e-mail. Daca codul este corect, atunci va fi creat contul si se va adauga o noua inregistrare in tabela “users”.

Se foloseste functia “createUserWithEmailAndPassword” din libraria “firebase/auth” pentru crearea contului si “addDoc” pentru adaugarea unei noi inregistrari in tabela “users” fin libraria firebase/firestore.

```

const handleVerify = async () => {
  const codeAsNumber = parseInt(verificationCode, 10);
  console.log(verificationCode);
  console.log(verificationCode);
  if (codeAsNumber === generatedCode) {
    Alert.alert('Verification Successful', 'Your account has been successfully verified!');

    try {
      const userCredential = await createUserWithEmailAndPassword(auth, email, password);
      const user = auth.currentUser;
      await addDoc(collection(db, 'users'), {
        userId: user.uid,
        email: email,
        description: "profile description",
        timestamp: serverTimestamp(),
      });
      navigation.navigate('Home');
      console.log("Signup successfully");
    }
  }
};

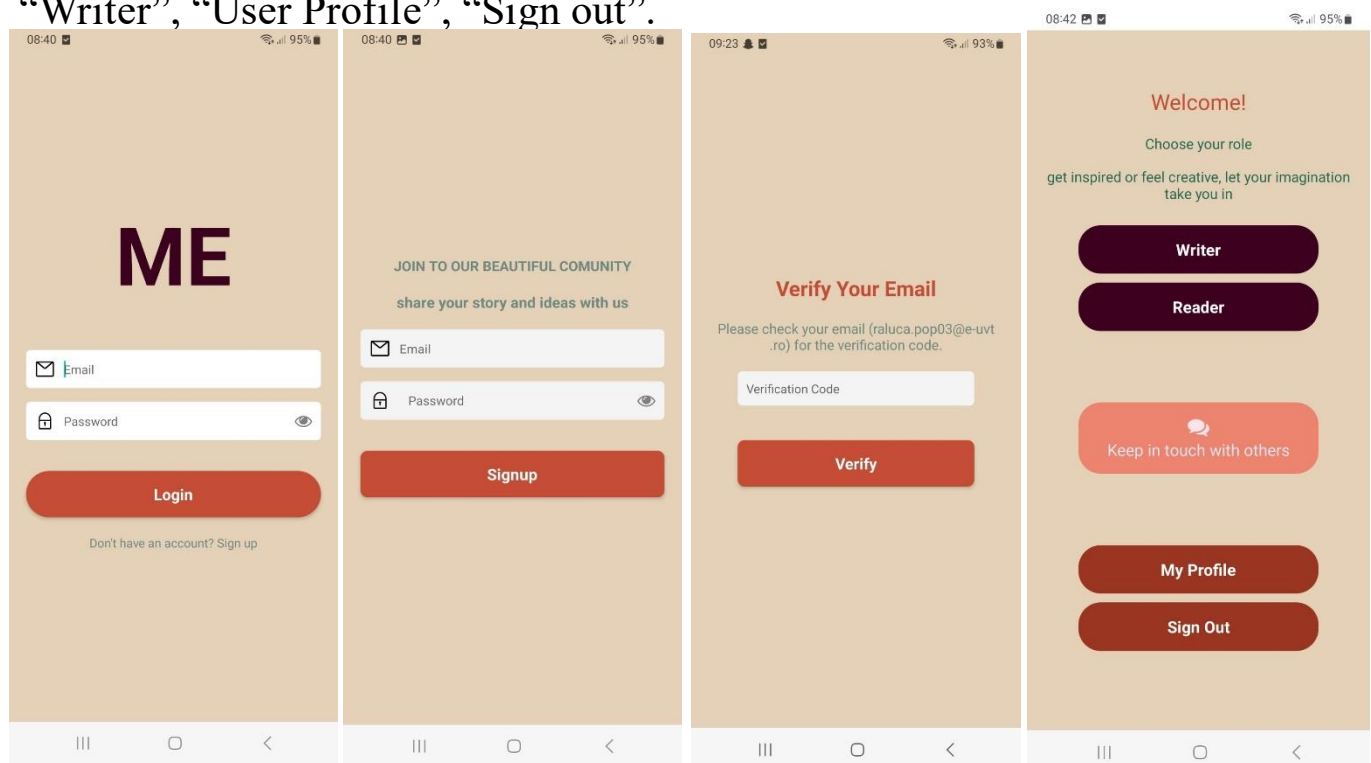
```

Pentru autentificare utilizatorul trebuie sa introduca adresa de e-mail si de parola. Se foloseste functia “signInWithEmailAnsPassword” din libraria “firebase/auth” pentru autentificare.

```
const [email, setUsername] = useState('');
const [password, setPassword] = useState('');
const [showPassword, setShowPassword] = useState(false);

const handleLogin = async () => {
  try {
    await signInWithEmailAndPassword(auth, email, password);
    console.log('Login successful!');
    navigation.navigate('Home');
  } catch (error) {
    if (error.code === 'auth/invalid-login-credentials') {
      Alert.alert('Wrong email or password. Try again');
    } else if (error.code === 'auth/invalid-email') {
      Alert.alert('Invalid email. Try again');
    }
    else {
      console.error('Login error:', error.message);
    }
  }
};
```

Dupa autentificare sau inregistrare, utilizatorul este redirectionat catre pagina principala. In pagina principala se gasesc patru optiuni: “Reader”, “Writer”, “User Profile”, “Sign out”.



## Rolul de “Reader”

Daca autorul alege rolul de reader, acesta va fi redirectionat catre o alta pagina unde va trebui sa aleaga una din cele patru sectiuni din care vrea sa citeasca. Dupa ce va face alegerea, va fi redirectionat catre pagina in care vor aparea toate articolele din categoria selectata.

Pentru a face filtrarea postarilor s-a folosit functia “fetchPosts” care afiseaza doar inregistrarile din tabela pentru care atributul “category” corectunde sectiunii selectate (variabila section, a carei valoare este transmisa de la ecranul precedent).

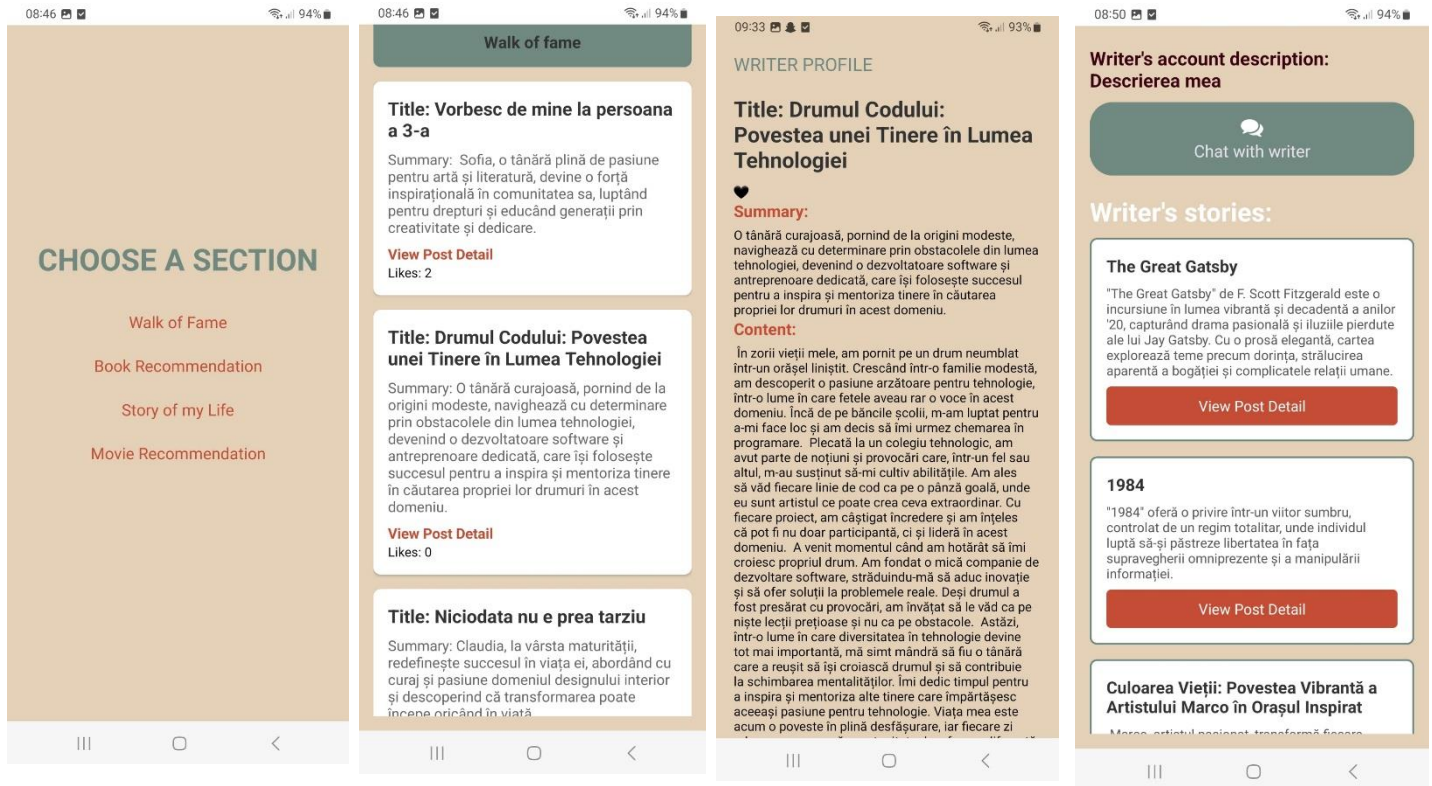
```
const ReaderPage = ({ route }) => {
  const { section } = route.params;
  const [stories, setStories] = useState([]);
  const navigation = useNavigation();
  const currentUser = auth.currentUser;
  const [isLiked, setIsLiked] = useState(false);

  useEffect(() => {
    const fetchPosts = async () => {
      try {
        // Fetch posts based on the selected section
        const postsCollection = collection(db, 'stories');
        const sectionQuery = query(postsCollection, where('category', '==',
section));
        const unsubscribe = onSnapshot(sectionQuery, (snapshot) => {
          setStories(snapshot.docs.map(doc => ({ id: doc.id, ...doc.data() })));
        });

        return () => unsubscribe(); // Cleanup the subscription when the
component unmounts
      } catch (error) {
        console.error('Error fetching posts:', error);
      }
    };

    fetchPosts();
  }, [section]);
```





In ecranul principal din “Reader” apare titlul fiecărei postări, un rezumat al acesteia, numărul de like-uri pe care îl are în prezent și un buton “View Post Detail”. Dacă butonul este apăsător, utilizatorul va fi redirectionat către o pagină unde se află întregul conținut al postării, un buton de like pe care poate să îl apese pentru a aprecia postarea, și un buton numit “View Writer Profile”.

Pentru a afișa întregul conținut al postării selectate, din ecranul “Reader” se transmite id-ul postării din tabela “stories” către pagina “PostDetail”, iar de acolo se afișează datele corespunzătoare.

```
const fetchPostDetails = async () => {
  try {
    // Fetch details of the specific post
    const postDocRef = doc(db, 'stories', postId);
    const postDoc = await getDoc(postDocRef);

    if (postDoc.exists()) {
      setPost({ id: postDoc.id, ...postDoc.data() });
    }
  }
}
```

Fiecare inregistrare din tabela “stories” are un atribut numit “likedBy” unde sunt retinute toate id-urile utilizatorilor care au apreciat postarea respectiva. Pentru gestionarea butonului de like se face o verificare a atributului respectiv pentru a observa daca id-ul utilizatorului current apare in lista respectiva. Daca apare, atunci butonul de like va fi o inimioara rosie, iar daca acesta va fi apasat, atunci autorul isi retrage like-ul. In caz contrar o inimioara va fi neaga, iar daca o va apasa aceasta se va transforma in rosu si postarea va fi apreciata.

```
const [isLiked, setIsLiked] = useState(false);
const likedBy = post?.likedBy || [];

useEffect(() => {
  setIsLiked(likedBy.includes(currentUser.uid));
}, [likedBy, currentUser.uid]);

const handleLike = async () => {
  try {
    const postDocRef = doc(db, 'stories', post.id);

    // If already liked, remove user from likedBy
    if (isLiked) {
      await updateDoc(postDocRef, {
        likedBy: arrayRemove(currentUser.uid),
      });
    } else {
      // If not liked, add user to likedBy
      await updateDoc(postDocRef, {
        likedBy: arrayUnion(currentUser.uid),
      });
    }

    // Toggle the like status
    setIsLiked((prevIsLiked) => !prevIsLiked);
  } catch (error) {
    console.error('Error updating like status:', error);
  }
};
```

```
<TouchableOpacity onPress={handleLike}>
  <Text style={{ color: isLiked ? 'red' : 'black' }}>{isLiked ? '❤️' :
'❤️'}</Text>
</TouchableOpacity>
```

Butonul “View writer profile” va redirectiona utilizatorul catre pagina de profil a scriitorului. Pentru aceasta, se transmite id-ul scriitorului (care reprezinta un atribut in tablea ‘stories’) si se cauta apoi in tabela ‘users’ inregistrarea corespunzatoare.

Tabela “users” contine urmatoarele attribute: “id-ul utilizatorului din Authentication, descrierea profilului utilizatorului, si adresa acestuia de e-mail. Astfel, dupa ce scriitorul a fost identificat dupa atributul “userId”, se vor afisa email-ul acestuia si descriere.

```
const fetchWriterProfile = async () => {
  try {
    // Fetch details of the user with the specified writerId
    const usersCollectionRef = collection(db, 'users');
    const userQuery = query(usersCollectionRef, where('userId', '==',
writerId));
    const userQuerySnapshot = await getDocs(userQuery);

    // Check if the user with the specified userId exists
    if (userQuerySnapshot.docs.length > 0) {
      const userData = userQuerySnapshot.docs[0].data();
      setWriterProfile({ id: userQuerySnapshot.docs[0].id, ...userData });
    } else {
      console.log('User not found');
    }
  } catch (error) {
    console.error('Error fetching user details:', error);
  }
};

fetchWriterProfile();
fetchUserStories();
}, [writerId]);
```

Pentru a afisa postarile pe care scriitorul le-a postat, se cauta in tabela “stories” acele inregistrari pentru care atributul `userId` este egal cu id-ul scriitorului.

```
const fetchUserStories = async () => {
  try {
    const storiesQuery = query(collection(db, 'stories'), where('userId', '=', writerId));
    const storiesSnapshot = await getDocs(storiesQuery);

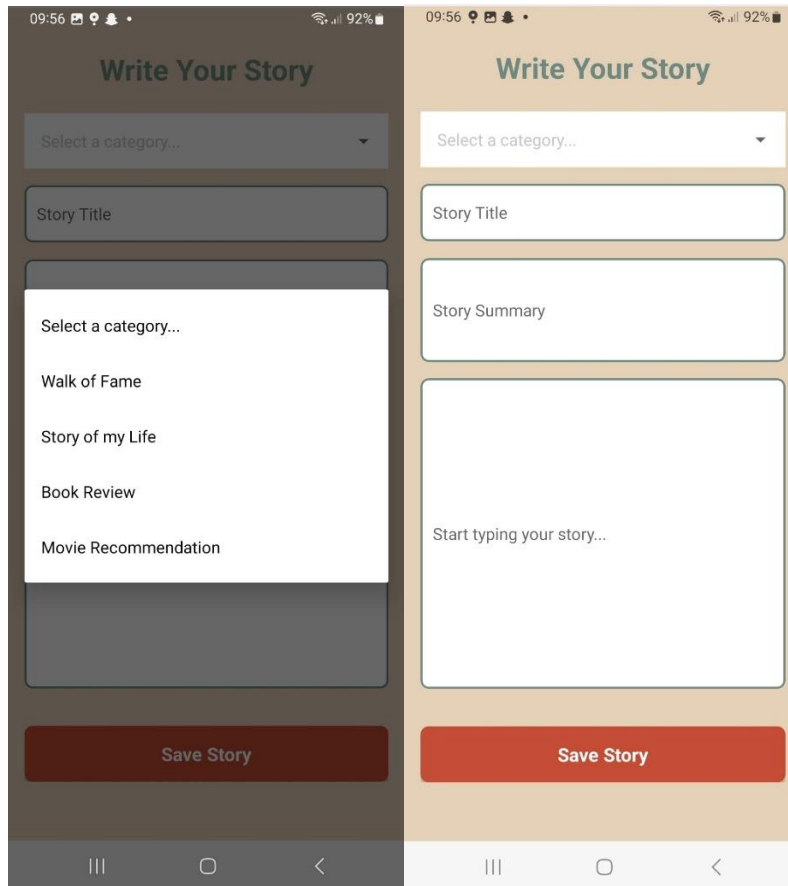
    const storiesData = storiesSnapshot.docs.map((doc) => ({
      id: doc.id,
      ...doc.data(),
    }));

    setUserStories(storiesData);
  } catch (error) {
    console.error('Error fetching user stories:', error);
  }
};
```

Fiecare astfel de postare are un buton de “View post detail” care functioneaza identic cu cel prezentat precedent.

## Rolul de “Writer”

Daca utilizatorul alege acest rol, va fi redirectionat catre pagina de write, unde va putea sa isi scrie articolul. Acesta trebuie sa aleaga una din cele 4 sectiuni in care doreste sa scrie, si sa completeze cele 3 casute de text.



In momentul in care utilizatorul apasa butonul “Save Story” articolul este salvat in tabela “stories”, iar acesta va primi un mail de confirmare si va fi redirectionat la pagina de “Home”.

```
const handleSaveStory = async () => {  
  try {  
  
    const user = auth.currentUser;  
    if (!user) {
```

```

        console.error('User not authenticated');
        return;
    }
    // Add the story data to Firestore with the user ID
    await addDoc(collection(db, 'stories'), {
        userId: user.uid,
        title: storyTitle,
        summary: storySummary,
        content: storyContent,
        category: selectedCategory,
        likedBy: [],
        timestamp: serverTimestamp(),
    });

    console.log('Story saved successfully to Firebase!');
    sendEmail(storyContent, storyTitle, storySummary);
} catch (error) {
    console.error('Error saving story to Firebase:', error);
}
navigation.navigate('Home');
});

```

```

const sendEmail = async (storyContent, storyTitle, storySummary) => {
    try {
        const templateParams = {
            email: auth.currentUser.email,
            subject: 'Your Story has been posted!',
            message: 'Thank you for posting your story. It has been successfully
saved.',
        };

        const response = await emailjs.send(
            'service_xhrfr5c',
            'template_fqd0s5i',
            templateParams,
            'M2nZ6qEBEoijAaW4m'
        );

        console.log('Email sent successfully:', response);
    } catch (error) {
        console.error('Error sending email:', error);
    }
};

```

## User Profile

In pagina de user profile, apare adresa de e-mail a utilizatorului, descriere contului sau pe care o poate edita, postarile pe care acesta le-a scris si cele la care a dat like.

Pentru a afisa descrierea si adresa de e-mail, se cauta in tabela “users” inregistrarea cu atributul `userId` egal cu id-ul utilizatorului curent.

```
const fetchUserData = async () => {
  try {
    const usersCollectionRef = collection(db, 'users');
    const userQuery = query(usersCollectionRef, where('userId', '==',
currentUser.uid));
    const userQuerySnapshot = await getDocs(userQuery);
    if (userQuerySnapshot.docs.length > 0) {
      const userData = userQuerySnapshot.docs[0].data();
      setUserProfile({ id: userQuerySnapshot.docs[0].id, ...userData });
    } else {
      console.log('User not found');
    }

    setLoading(false);
  } catch (error) {
    console.error('Error fetching user data:', error);
    setLoading(false);
  }
};
```

Pentru a afisa articolele postate, se cauta in tabela “stories” inregistrarile cu atributul `userId` egal cu id-ul utilizatorului curent, iar pentru a afisa postarile apreciate, se cauta in fiecare inregistrare din tabela “stories” daca atributul “`likedBy`” contine id-ul utilizatorului curent.

```
const fetchUserStories = async () => {
  try {
    const storiesQuery = query(collection(db, 'stories'), where('userId',
'==', currentUser.uid));
    const storiesSnapshot = await getDocs(storiesQuery);

    const storiesData = storiesSnapshot.docs.map((doc) => ({
      id: doc.id,
      ...doc.data(),
    }));

    setUserStories(storiesData);
  } catch (error) {
    console.error('Error fetching user stories:', error);
  }
};
```

```
const fetchLikedStories = async () => {
  try {
    const likedStoriesQuery = query(collection(db, 'stories'),
where('likedBy', 'array-contains', currentUser.uid));
    const likedStoriesSnapshot = await getDocs(likedStoriesQuery);

    const likedStoriesData = likedStoriesSnapshot.docs.map((doc) => ({
      id: doc.id,
      ...doc.data(),
    }));

    setLikedStories(likedStoriesData);
  } catch (error) {
    console.error('Error fetching liked stories:', error);
  }
};
```

Pentru afisarea listei cu postarile create de utilizatorul curent, sau cu cele apreciate, se foloseste o valoare numita “activeTab”, care initial are valoarea “Your stories”. In momentul in care utilizatorul apasa pe butonul “Liked Stories”, valoarea tablui se va schimba si vor fi afisate

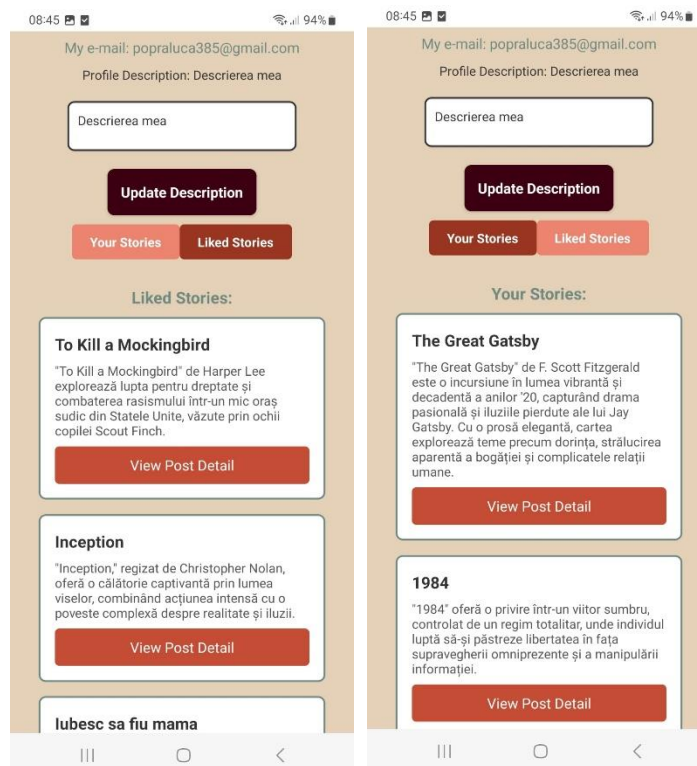


postările apreciate.

Pentru editarea descrierii, exista o casuta in care deja apare descrierea curenta, care poate fi editata direct de acolo, iar in momentul in care se apasa butonul “Update description”, se salveaza modificarile in baza de date. Se foloseste functia “setDoc” pentru salvare in baza de date, si “setUserProfile” pentru reactualizarea datelor in aplicatie.

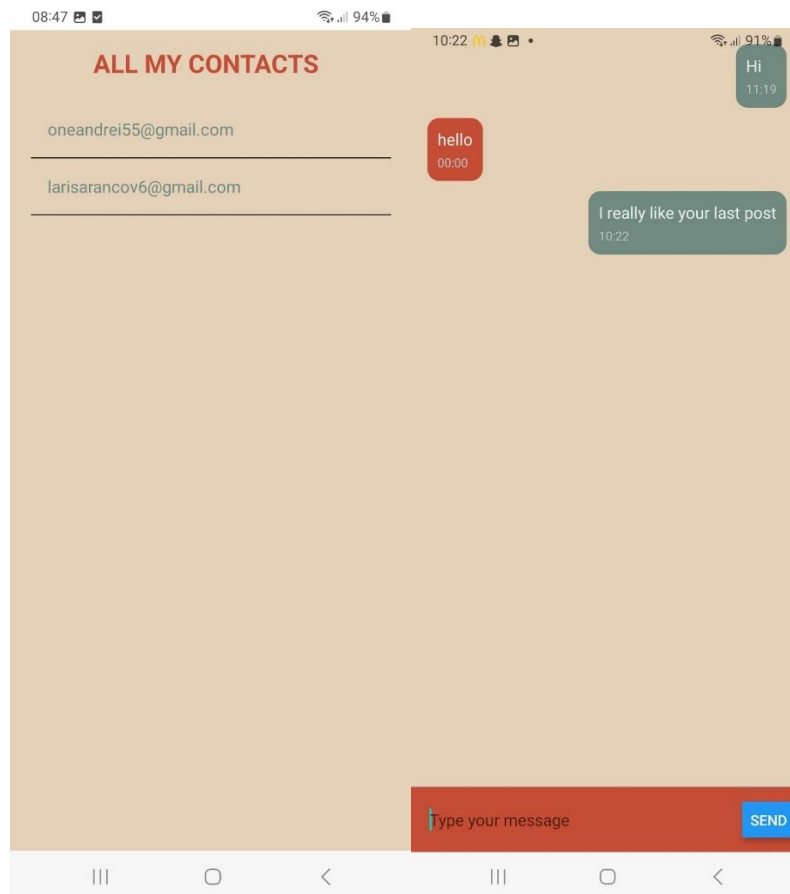
```
const updateDescription = async () => {
  try {
    const userDocRef = doc(db, 'users', userProfile.id);
    await setDoc(userDocRef, { description: userProfile.description }, { merge: true });

    // Optionally, you can also update the local state after a successful update
    setUserProfile((prevProfile) => ({ ...prevProfile, description: userProfile.description }));
  } catch (error) {
    console.error('Error updating description:', error);
  }
};
```



## Chat Screen

In ecranul Home exista posibilitatea “Keep in touch with others” care permite utilizatorului sa converseze cu alti utilizatori. De asemenea, acest buton este regasit si in pagina de detalii ale utilizatorului, unde utilizatorul este direct redirectionat catre conversatia cu scriitorul.



```
const fetchConversationId = async () => {
  try {
    // Query conversations where the user is a participant
    const conversationsQuery = query(
      collection(db, 'conversations'),
      where('participants', 'array-contains', currentUser.uid)
    );

    const conversationsSnapshot = await getDocs(conversationsQuery);

    // Find the conversation with the specified writer
    const matchingConversation = conversationsSnapshot.docs.find(doc =>
      doc.data().participants.includes(writerId)
    );
  }
}
```

```

    );

    if (matchingConversation) {
      setConversationId(matchingConversation.id);
    } else {
      // If no conversation exists, create a new one
      const newConversation = await addDoc(collection(db, 'conversations'), {
        participants: [currentUser.uid, writerId],
      });

      console.log('New Conversation ID:', newConversation.id);

      setConversationId(newConversation.id);
    }
  } catch (error) {
    console.error('Error fetching conversation:', error);
  }
};

fetchConversationId();
}, [currentUser.uid, writerId]));

```

Aceste informatii sunt salvate in tabela “conversations”. Exista un atribut numit participants care contine id-urile utilizatorilor pentru o conversatie si o subcolectie “messages” care contine id-ul utilizatorului care a trimis mesajul, continutul mesajului si data-ora acestuia.

<a href="#">Home</a> > <a href="#">conversations</a> > <a href="#">VlJVL8LFtbpICil...</a> > <a href="#">messages</a> > <a href="#">LvYPrgwiIVsHFs</a> <a href="#">More in Google Cloud</a>		
<div> <div>VlJVL8LFtbpICil7sVmX</div> <div> <div>+ Start collection</div> <div>messages</div> </div> </div>	<div> <div>messages</div> <div> <div>+ Add document</div> <div> <div>LvYPrgwiIVsHFsQ2aibT</div> <div> <div>mor1DdaAIJazSRHaxRr0</div> <div>v7xvyn7WnxCsv0u5wKKT</div> </div> </div> </div> </div>	<div> <div>LvYPrgwiIVsHFsQ2aibT</div> <div> <div>+ Start collection</div> <div> <div>+ Add field</div> <div> <div>content: "I really like your last post"</div> <div>senderId: "vrzHrHjDSmN2U46zy0c9Em5LyWX2"</div> <div>timestamp: December 9, 2023 at 10:22:00 AM UTC+2</div> </div> </div> </div> </div>
<div> <div>+ Add field</div> <div> <div>participants</div> <div> <div>0 "vrzHrHjDSmN2U46zy0c9Em5LyWX2"</div> <div>1 "1G1XWvpX6qfRhGe4O9s2egKdZmc"</div> </div> </div> </div>		