ДОДАТОК А

Програмний код сервісів додатку:

```
export class AuthService {
public currentUser: User = null;
 constructor(private afAuth: AngularFireAuth, private afs: AngularFirestore) {
   this.afAuth.onAuthStateChanged(user => {
     this.currentUser = user;
   });
 }
 public async signUp({ email, password}) {
   const credential = await this.afAuth.createUserWithEmailAndPassword(
     email,
     password
   );
   const uid = credential.user.uid;
   return this.afs.doc(
     `users/${uid}`
     ).set({
       uid,
       email: credential.user.email,
       name: credential.user.email.split('@')[0],
       image: 'notdefined'
     });
 }
 signIn({ email, password}) {
   return this.afAuth.signInWithEmailAndPassword(email, password);
 }
 signOut() {
   return this.afAuth.signOut();
 }
 async getCurrentUser() {
   return (await this.afAuth.currentUser).uid;
 }
 getUserProfile()  {
   const observable = new Observable((observer) => {
     this.getCurrentUser().then( uid => {
```

```
      this.afs.collection('users').doc(uid).valueChanges().subscribe( (profile: User) => {
       if ( profile.image === 'notdefined') {
        profile.image = 'assets/undraw_male_avatar_323b.svg';
       }
       observer.next(profile);
       observer.complete();
      });
     });
   });
   return observable;
  }
export class ChatService {
  currentUser: User = null;
  constructor(private afAuth: AngularFireAuth,
              private afs: AngularFirestore) {
    this.afAuth.onAuthStateChanged(user => {
      this.currentUser = user;
    });
  }
  addChatMessage(msg) {
    return this.afs.collection('messages').add({
      msg: {msg}.msg,
      from: this.currentUser.uid,
      createdAt: firebase.default.firestore.FieldValue.serverTimestamp()
    });
  }
  getChatMessage() {
    let users = [];
    return this.getUsers().pipe(
      switchMap(res => {
        users = res;
        return this.afs.collection('messages', ref => ref.orderBy('createdAt')).valueChanges({ idField: 'id'}) as
Observable<Message[]>;
      }),
      map(messages => {
        for (const m of messages) {
          m.fromName = this.getUserProfile(m.from, users).name;
          m.user = this.getUserProfile(m.from, users);
```

```
        m.myMsg = this.currentUser.uid === m.from;
      }
      return messages;
    })
  );
 }
 getUsers() {
  return this.afs.collection('users').valueChanges({ idField: 'uid' }) as Observable<User[]>;
 }
 getUserProfile(msgFromId, users: User[]): User {
  for (const usr of users) {
   if (usr.uid === msgFromId) {
    if ( usr.image === 'notdefined' ) {
     usr.image = 'assets/undraw_male_avatar_323b.svg';
    }
    return usr;
   }
  }
  return null;
 }
}
export class PostService {
 currentUser: User = null;
 typeOfPost: string = null;
 users: User[];
 constructor(private afAuth: AngularFireAuth,
             private afs: AngularFirestore) {
              this.afAuth.onAuthStateChanged(user => {
               this.currentUser = user;
              });
 }

 addPostMessage(msg, type) {
  return this.afs.collection('posts').add({
   msg: {msg}.msg,
   from: this.currentUser.uid,
   type: {type}.type,
   createdAt: firebase.default.firestore.FieldValue.serverTimestamp()
```

```
    });
  }
  addPostComment(msg, postId) {
    return this.afs.collection('posts').doc(postId).collection('comments').add({
      msg: {msg}.msg,
      from: this.currentUser.uid,
      createdAt: firebase.default.firestore.FieldValue.serverTimestamp()
    });
  }
  getPostMessage() {
    let users = [];
    return this.getUsers().pipe(
      switchMap(res => {
        users = res;
        return this.afs.collection('posts', ref => ref.orderBy('createdAt', 'desc')).valueChanges({ idField: 'id'})
as Observable<Post[]>;
      }),
      map(posts => {
        for (const m of posts) {
          m.from = this.getUserProfile(m.from, users).name;
          m.userProfile = this.getUserProfile(m.from, users);
          if ( m.likes !== undefined && m.likes !== null ) {
            m.countLikes = m.likes.length;
            // check if uder uid is in the array of likes
            if (m.likes.includes(this.currentUser.uid)) {
              m.userLikes = true;
            } else {
              m.userLikes = false;
            }
          } else {
            m.countLikes = 0;
          }

          this.getPostComments(m.id).subscribe( comments => {
            m.comments = comments;
            m.countComments = comments.length;
          });
        }
```

```
      return posts;
    })
  );
}
getPostComments(messageId) {
  let users = [];
  return this.getUsers().pipe(
    switchMap(res => {
      users = res;
      return this.afs.collection('posts').doc(messageId)
      .collection('comments', ref => ref.orderBy('createdAt', 'desc'))
      .valueChanges({ idField: 'id' }) as Observable<Comment[]>;
    }),
    map(comments => {
      for (const c of comments) {
        c.fromName = this.getUserProfile(c.from, users).name;
      }
      return comments;
    })
  );
}
getUsers() {
  return this.afs.collection('users').valueChanges({ idField: 'uid' }) as Observable<User[]>;
}
getUserProfile(msgFromId, users: User[]): User {
  for (const usr of users) {
    if (usr.uid === msgFromId) {
      return usr;
    }
  }
  return null;
}

updatePostLike(post: Post) {
  let tempLikesArray = [];
  if ( post.likes !== undefined && post.likes !== null ) {
    tempLikesArray = post.likes;
    if (tempLikesArray.includes(this.currentUser.uid)){
```

```
      const index = tempLikesArray.indexOf(this.currentUser.uid);
      tempLikesArray.splice(index, 1);
     } else {
      tempLikesArray.push(this.currentUser.uid);
     }
   } else {
    tempLikesArray.push(this.currentUser.uid);
   }    return this.afs.collection('posts').doc(post.id).update({
    likes: tempLikesArray,
   });
  }
}
```