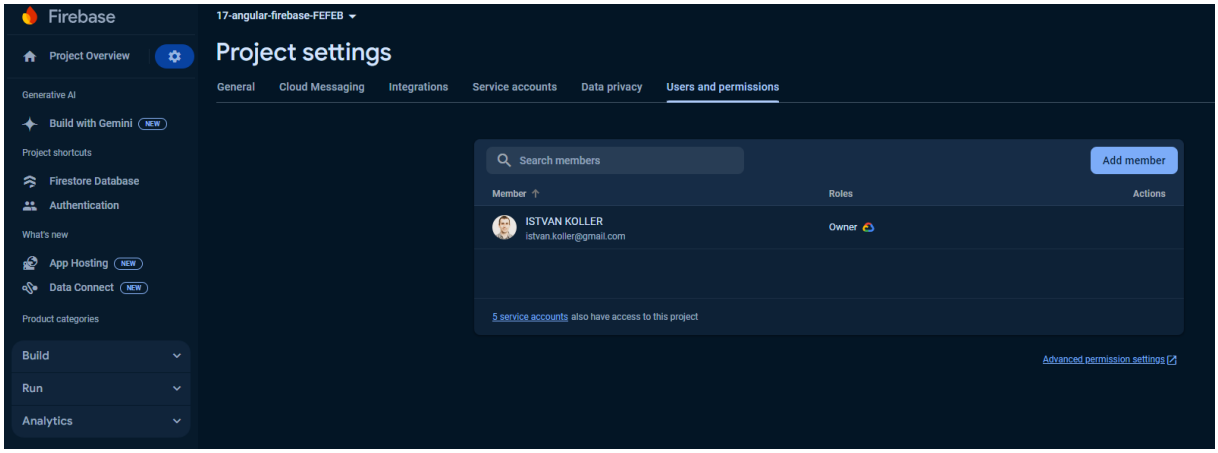
Angular – Firebase – Projektmunka tippek

Firebase:

- 1 valaki a csapatból, hozzon létre egy új firebase projektet, lehet a neve pl Angular-firebase-project

- Ha kész legyen bekapcsolva a cloud firestore, és az authentikációban az email/password és a google opció

- ha kész a project settingben a Users and permission tabon meg tudod hívni a csapatból a többi embert email cím alapján és közösen tudtok dolgozni a projekten



Add member résznél… adminként érdemes a csapattagokat hozzáadni…

Kiindulási állapot:

Angular 18-as változat.
SCSS-t választottam, mert jobb kiindulás szerintem, mint a CSS.

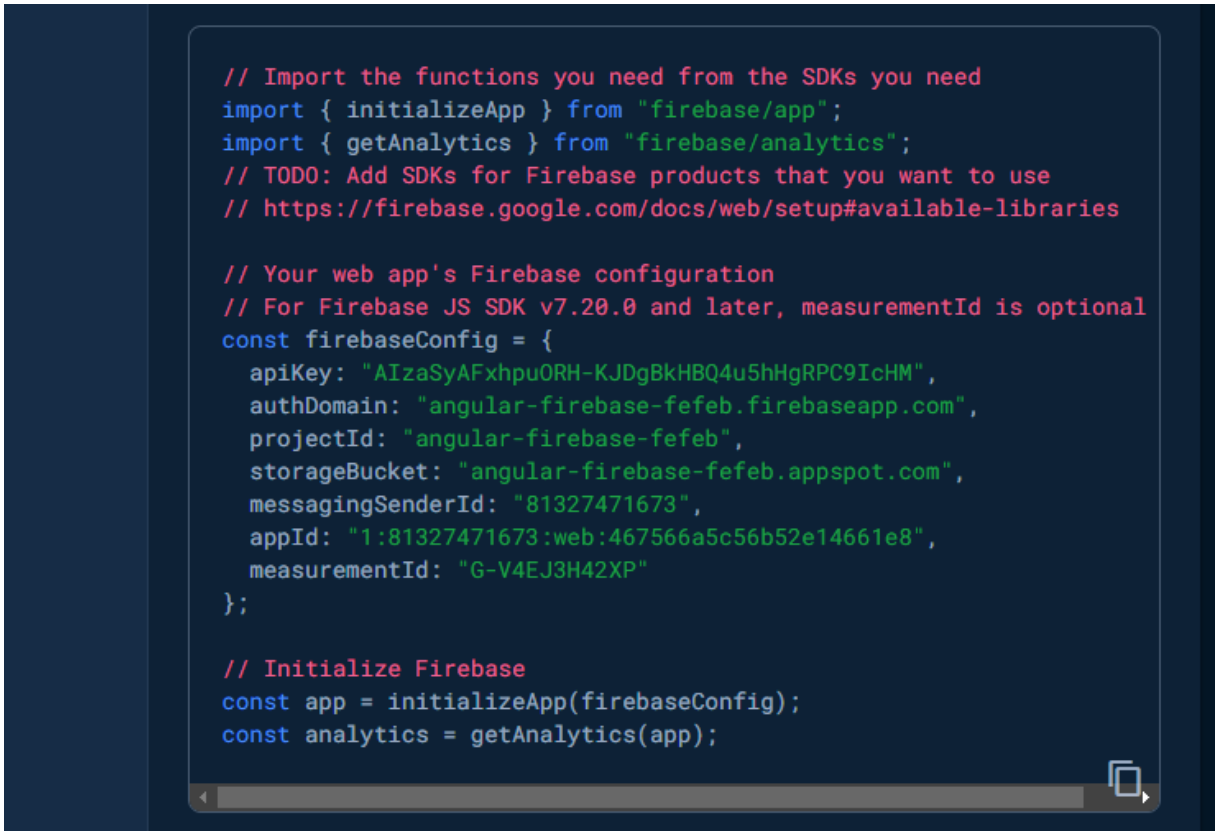
A Bootstrap nincs bekötve.

Angular – firebase összekötése:

```
npm i
```

majd

```
ng add @angular/fire
```



```
// Import the functions you need from the SDKs you need
import { initializeApp } from "firebase/app";
import { getAnalytics } from "firebase/analytics";
// TODO: Add SDKs for Firebase products that you want to use
// https://firebase.google.com/docs/web/setup#available-libraries

// Your web app's Firebase configuration
// For Firebase JS SDK v7.20.0 and later, measurementId is optional
const firebaseConfig = {
  apiKey: "AIzaSyAFxhpuORH-KJDgBkHBQ4u5hHgRPC9IcHM",
  authDomain: "angular-firebase-fefeb.firebaseapp.com",
  projectId: "angular-firebase-fefeb",
  storageBucket: "angular-firebase-fefeb.appspot.com",
  messagingSenderId: "81327471673",
  appId: "1:81327471673:web:467566a5c56b52e14661e8",
  measurementId: "G-V4EJ3H42XP"
};

// Initialize Firebase
const app = initializeApp(firebaseConfig);
const analytics = getAnalytics(app);
```

A firebase configot kell bemásolni. Zölddel jelöltem, hogy hova… (app.modulet.ts)

```
  providers: [

    provideFirebaseApp(() =>
      initializeApp({
        projectId: 'angular-firebase-fefeb',
        appId: '1:81327471673:web:467566a5c56b52e14661e8',
        storageBucket: 'angular-firebase-fefeb.appspot.com',
        apiKey: 'AIzaSyAFxhpuORH-KJDgBkHBQ4u5hHgRPC9IcHM',
        authDomain: 'angular-firebase-fefeb.firebaseapp.com',
        messagingSenderId: '81327471673',
        measurementId: 'G-V4EJ3H42XP',
      })
    ),
    provideAuth(() => getAuth()),
    provideFirestore(() => getFirestore()),
  ],
```

Github

- érdemes sok branchet létrehozni és gyakran mergelni, vagy merge requestel dolgozni, hogy elkerüljétek a sok merge conflictot

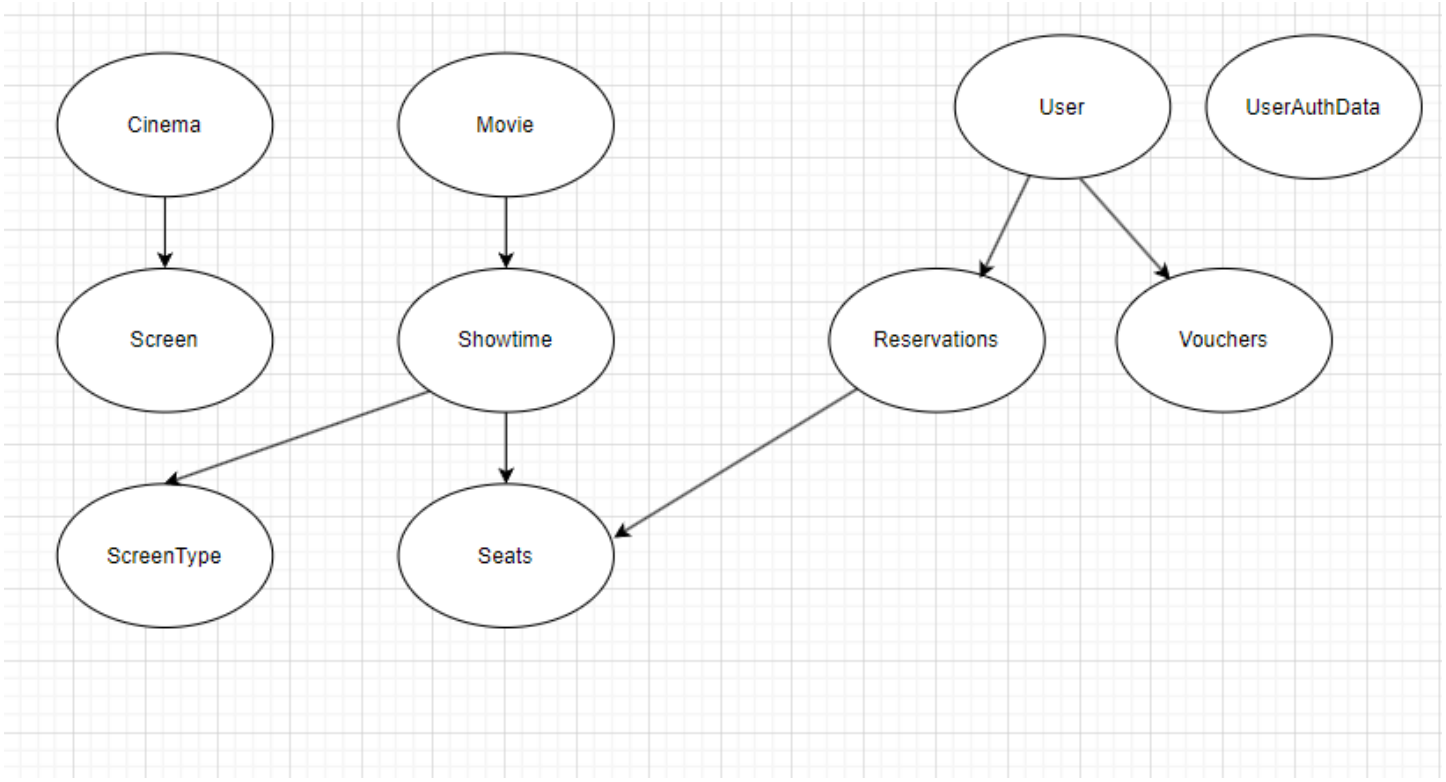
- érdemes a feladatokat úgy megbeszélni / elosztani, hogy kevés ütközés legyen, de egy idő után nem lesz elkerülhető, de ezt is gyakorolni kell

- nem kötelező de érdemes egy development branchet létrehozni, abba betenni az új fejlesztéseket, és ha minden működik akkor mehet a main-be

- a végső állapot a mainba legyen

Modellek

Küldök egy lehetséges egyszerűsített példát, hogy el tudjátok képzelni pl hogy érdemes felépíteni az adatbázist, a modelleket:



cinema.model.ts

```
import { Screen } from "./screen.model";

export interface Cinema {
  id?: string; //egyedi azonosító az adatbázisban
  name: string; //A mozi neve
  zipCode: number; //irányítószám alapján történő keresést teszi lehetővé
  location: string; //A mozi pontos címe pl "Aréna Mall, Budapest"
  city: string;
  mapUrl: string; //Egy Google Maps vagy más térképszolgáltató URL-je a mozi helyének
megjelenítésére.
  screens: Screen[]; //minden vetítőteremhez tartalmazza a szükséges adatokat: melyik mozihoz
tartozik, neve, befogadóképessége, VIP elérhetőség
  movies?: string[]; // Filmek azonosítóinak listája
}
```

movie.model.ts

```
import { ShowTime } from "./showtime.model";

export interface Movie {
  id?: string; //auto-generált id
  title: string;
  description: string;
```

```
  duration: number;
  releaseDate: Date;
  genres: string[]; //több műfaj felsorolható a tömbben
  posterUrl: string; //poszter URL
  trailerUrl?: string; //előzetes URL
  rating: number; //értékelés (1-10)
  showtimes?: ShowTime[]; //vetítési idők
  vipAvailable: boolean; //Vip szolgáltatás elérhető-e
  isActive: boolean; //filmek archiválására admin módban. TRUE = aktív, FALSE = archívált
}
```

reservation.model.ts

```
import { Seat } from "./seat.model";

export interface Reservation {
  id?: string;
  userId: string; //felhasználó azonosító
  showtimeId: string; //vetítés azonosító
  seats: Seat[]; //foglalt helyek
  totalPrice: number; //végösszeg
  status: "pending" | "confirmed" | "cancelled"; //A foglalási folyamat alatt pending,
megerősítés után confirmed, törlés után cancelled.
}
```

screen.model.ts

```
export interface Screen {
  id?: string; //auto-generált id
  cinemaId: string; //mozi azonosítója
  name: string; //A terem neve vagy száma
  capacity: number; //A terem befogadóképessége
  vipAvailable: boolean; //Elérhető-e VIP szolgáltatás
}
```

seat.model.ts

```
export interface Seat {
  row: string; //pl. A, B, C...
  column: number; //pl. 1, 2, 3...
  seatNumber: string; //pl A1, A2..., B1...
  isBooked: boolean; //TRUE = Foglalt, FALSE = szabad
  price: number; //Az adott hely ára
}
```

showtime.model.ts

```
import { Seat } from "./seat.model";
```

```ts
export interface ShowTime {
  id?: string; //auto-generált id
  movieId: string; //film azonosítója, amelyet a vetítésben játszanak
  cinemaId: string; //mozi azonosítója, amelyben a vetítés történik
  screenId: string; //vetítőterem azonosítója
  startTime: Date; //A vetítés időpontja
  seats: Seat[]; //Az összes ülőhely listája ebben a vetítésben, foglalási állapottal.
}
```

user.model.ts

```ts
import { Reservation } from "./reservation.model";
import { Voucher } from "./voucher.model";

export interface User {
  id?: string;
  userName: string;
  firstName: string;
  lastName: string;
  email: string;
  role: "guest" | "user" | "admin"; //Vendégként nem foglalhat, vásárolhat, csak
bejelentkezett userként. Adminként kezelheti a felületet is.
  reservations?: Reservation[]; //Foglalások
  vouchers: Voucher[]; //Kuponok, ajándékok
}
```

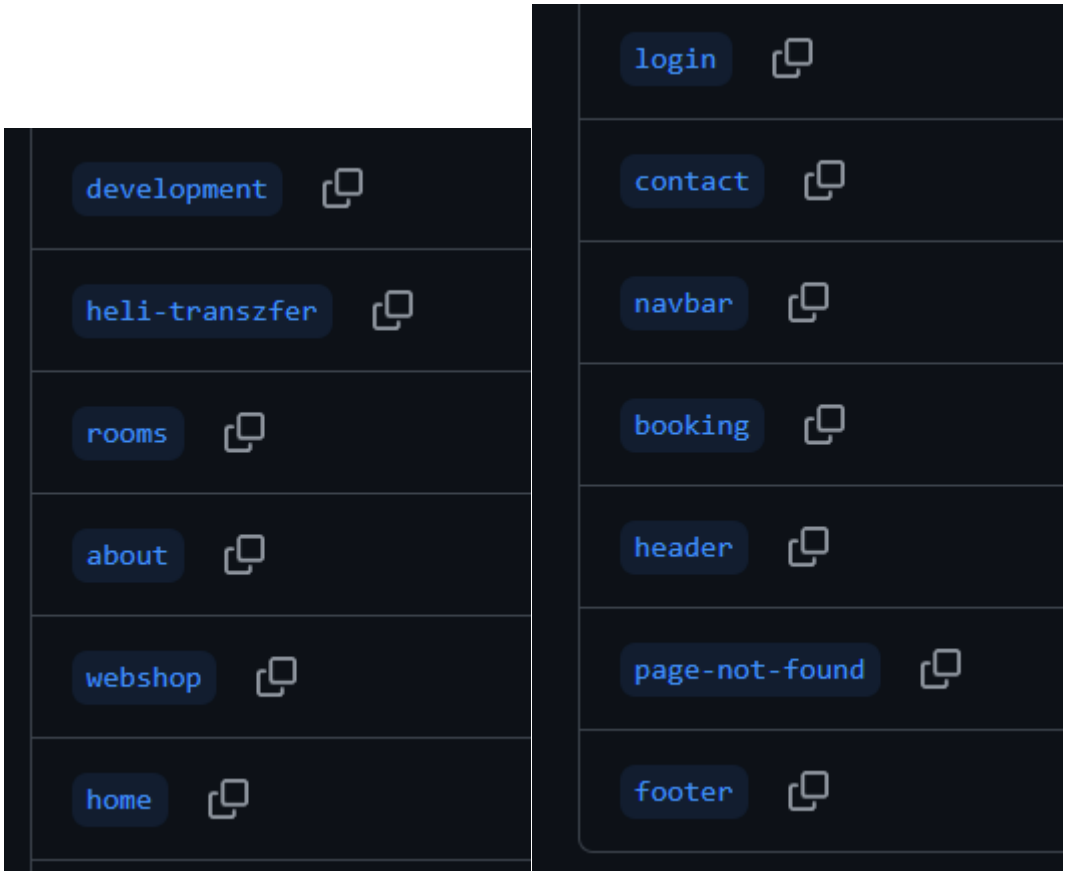
userAuthData.ts

```ts
export interface UserAuthData {
  email: string;
  password: string;
}
```

voucher.model.ts

```ts
export interface Voucher {
  id?: string;
  code: string; //kuponkód
  discount: number; // kedvezmény %-ban
  validUntil: Date; //érvényességi idő
  type: "gift" | "discount"; //ajándék, vagy kedvezmény
}
```

Barches példa



Shared components példa

Újra felhasználható input component

input.component.ts

```
import { Component, Input } from "@angular/core";
import { FormGroup } from "@angular/forms";
import { Error } from "../../../models/error";

@Component({
  selector: "app-input",
  templateUrl: "./input.component.html",
  styleUrl: "./input.component.scss",
})
export class InputComponent {
  @Input() formGroup!: FormGroup;
  @Input() componentClass = "col-3 mb-3";
  @Input() label = "";
  @Input() forId = "";
  @Input() controllerName = "";
  @Input() inputType = "";
  @Input() placeholder = "";
  @Input() inputClass = "form-control";
  @Input() labelClass = "form-label";
```

```
  @Input() errorArray: Error[] = [];
}
```

input.component.html

```
<div [formGroup]="formGroup" [class]="componentClass">
  <label *ngIf="label" [for]="forId" [class]="labelClass">{{ label }}</label>
  <input
    [type]="inputType"
    [class]="inputClass"
    [id]="forId"
    [formControlName]="controllerName"
    [placeholder]="placeholder"
  />

  <ng-container *ngFor="let error of errorArray">
    <small *ngIf="error.ngIfCondition" class="text-danger">{{
      error.messageShown
    }}</small></ng-container
  >
</div>
```

error.model.ts

```
export interface Error {
  ngIfCondition: boolean;
  messageShown: string;
}
```

Felhasználása:

```
form [formGroup]="testForm" (ngSubmit)="submitForm()" class="p-3">
  <app-input
    [formGroup]="testForm"
    [controllerName]="'name'"
    [label]="'Your name:'"
    [forId]="'name'"
    [inputType]="'text'"
    [placeholder]="'Enter your name here...'"
    [errorArray]="[{ ngIfCondition: name?.dirty && name?.errors?.['required'],
messageShown: 'required*'}, { ngIfCondition: name?.dirty &&
name?.errors?.['pattern'], messageShown: 'must be John*'}]"
  ></app-input>
  <app-input
```

```
    [formGroup]="testForm"
    [componentClass]="'col-2 mb-3'"
    [controllerName]="'salary'"
    [forId]="'salary'"
    [inputType]="'number'"
    [placeholder]="'Salary in $...'"
  ></app-input>
```

Újrafelhasználható select componens

select.component.ts

```
import { Component, Input } from "@angular/core";
import { FormGroup } from "@angular/forms";
import { Option } from "../../../models/option";

@Component({
  selector: "app-select",
  templateUrl: "./select.component.html",
  styleUrl: "./select.component.scss",
})
export class SelectComponent {
  @Input() formGroup!: FormGroup;
  @Input() componentClass = "col-3 mb-3";
  @Input() label = "";
  @Input() forId = "";
  @Input() controllerName!: string;
  @Input() disabledOption = "Please select one... ";
  @Input() options: Option[] = [];
}
```

select.component.html

```
<div [class]="componentClass" *ngIf="formGroup" [formGroup]="formGroup">
  <label *ngIf="label" [for]="forId" class="form-label">{{ label }}</label>
  <select class="form-select" [id]="forId" [formControlName]="controllerName">
    <option *ngIf="disabledOption" value="" selected disabled hidden>
      {{ disabledOption }}
    </option>
    <option *ngFor="let option of options" [value]="option.value">
      {{ option.text }}
    </option>
  </select>
</div>
```

Felhasználása:

```
  <app-select
    [formGroup]="testForm"
    [controllerName]="'country'"
    [options]="countryOptions"
  ></app-select>
```

Role lekérése példa

```
import { Injectable } from "@angular/core";
import {
  Auth,
  createUserWithEmailAndPassword,
  getAuth,
  GoogleAuthProvider,
  signInWithEmailAndPassword,
  signInWithPopup,
  User,
  UserCredential,
} from "@angular/fire/auth";
import { Router } from "@angular/router";
import { ToastrService } from "ngx-toastr";
import { BehaviorSubject, catchError, from, map, Observable, tap } from "rxjs";
import { UserAuthData } from "../../models/userAuthData";
import {
  collection,
  doc,
  Firestore,
  getDoc,
  getDocs,
  query,
  where,
} from "@angular/fire/firestore";
import { UserService } from "../user/user.service";

@Injectable({
  providedIn: "root",
})
export class AuthService {
  constructor(
    private router: Router,
    private auth: Auth,
    private toastr: ToastrService,
    private firestore: Firestore,
    private userService: UserService
  ) {}

  private loggedInStatus: BehaviorSubject<boolean | null> = new BehaviorSubject<
```

```
  boolean | null
>(null);

private userEmail: BehaviorSubject<string | null> = new BehaviorSubject<
  string | null
>(null);

private userIsAdmin: BehaviorSubject<boolean> = new BehaviorSubject<boolean>(
  false
);

public get userIsAdmin$(): Observable<boolean> {
  return this.userIsAdmin.asObservable();
}

public get loggedInStatus$(): Observable<boolean | null> {
  return this.loggedInStatus.asObservable();
}

get loggedInStatusValue(): boolean | null {
  return this.loggedInStatus.value;
}

isAdmin(): boolean {
  return this.userIsAdmin.value;
}

checkedLoggedInStatus(): boolean {
  if (!this.loggedInStatusValue) {
    this.router.navigate(["sign-in"]);
  }
  return true;
}

public get userEmail$(): Observable<string | null> {
  return this.userEmail.asObservable();
}

private googleAuthProvider = new GoogleAuthProvider();

registration(regData: UserAuthData): Observable<UserCredential> {
  return from(
    createUserWithEmailAndPassword(this.auth, regData.email, regData.password)
  ).pipe(
    tap((userCredential) => {
      console.log("user adatok: ", userCredential);
      this.loggedInStatus.next(true);
      this.toastr.success("Registration sucessfully and logged in!");
      this.router.navigate([""]);
```

```
    }),
    catchError((error) => {
      this.toastr.error(error.message);
      return error;
    })
  ) as Observable<UserCredential>;
}

login(loginData: UserAuthData): Observable<UserCredential> {
  return from(
    signInWithEmailAndPassword(this.auth, loginData.email, loginData.password)
  ).pipe(
    tap(async (userCredential) => {
      console.log("user adatok: ", userCredential);
      this.loggedInStatus.next(true);
      this.toastr.success("You have logged in successfully");
      this.router.navigate([""]);
      this.checkAuthState();
    }),
    catchError((error) => {
      this.toastr.error(error.message);
      return error;
    })
  ) as Observable<UserCredential>;
}

async logout(): Promise<void> {
  await this.auth.signOut();
  this.loggedInStatus.next(false);
  this.userIsAdmin.next(false);
  this.userEmail.next(null);
}

checkAuthState(): void {
  this.auth.onAuthStateChanged({
    next: async (user) => {
      if (user) {
        console.log("Van user initkor: ", user);
        this.loggedInStatus.next(true);
        this.userEmail.next(user.email);
        const email = user.email;
        try {
          const usersCollection = collection(this.firestore, "users");
          const q = query(usersCollection, where("email", "==", email));
          const querySnapshot = await getDocs(q);

          if (!querySnapshot.empty) {
            const docData = querySnapshot.docs[0].data(); // Az első találat
            console.log("User document:", querySnapshot.docs[0].id, docData);
```

```
            const role = docData["role"];
            this.userIsAdmin.next(role === "admin");
          } else {
            console.log("No user found with the given email.");
            this.userIsAdmin.next(false);
          }
        } catch (error) {
          console.error("Error fetching user by email:", error);
          this.userIsAdmin.next(false);
        }
      } else {
        console.log("No user is logged in.");
        this.loggedInStatus.next(false);
        this.userIsAdmin.next(false);
        this.userEmail.next(null);
      }
    },
    error: (error) => console.log("Auth state error:", error),
    complete: () => {},
  });
}

async loginWithGoogle(): Promise<void> {
  const user = await signInWithPopup(this.auth, this.googleAuthProvider);
  this.toastr.success("You logged in seccesfully!");
  console.log(user);
  this.router.navigate([""]);
}
}
```