Németh Sára, Kobida Krisztián 2024

Dokumentáció

Üdvözöljük a projektünk dokumentációjában! Ez a dokumentáció a fejlesztői csapatok számára készült, és célja, hogy segítse az alkalmazás működésének megértését, valamint a fejlesztési folyamatot elősegítse.

A megalkotott alkalmazásunk a mindennapi metróközlekedést hivatott könnyebbé tenni. Célcsoportunk minden metrót használó, aki időt szeretne spórolni miközben utazik.

Felhasznált technológiák:

Alkalmazásunkat Angular (17.2.0) keretrendszerben TypeScript nyelven, HTML és CSS, bootstrap (3.4.1), node.js, node package manager (továbbiakban: npm) segítségével hoztuk létre Visual Studio Code IDE-ben. Adatbáziskezelő rendszerként Firestore Databaset használtunk a projekt létrehozása során. Verziókövetés GIT alatt történt.

Előfeltételek a projekt telepítésénél:

Node.js és **npm** telepítése szükséges a projekt lokális futtatásához.
Git clone [repository címünk] futtatása és a projekt mappájába való belépés után (**cd [Mappa] parancs**) az **npm install** parancs segítségével tudjuk telepíteni a szükséges rendszereket, függőségeket, amelyek a package.json fileban vannak felsorolva. Ezután **ng serve** parancs futtatása következik és így alapesetben az alkalmazás a **'http://localhost:4200'** címen elérhetővé válik bármely támogatott böngésző alatt.

Projekt felépítése, alapkönyvtárak:

A projekt egy kifejezett mappastruktúrát követ, így hatékonyabbá téve az elrendezést. Alább található egy áttekintés a főkönyvtárakról:

src: Ez a mappa tartalmazza az összes forráskódját a projektnek.

environments: Konfigurációs fájlokat tartalmaz (environments.ts)

assets: Az alkalmazásban használt képek, specifikus CSS-fileok kerülnek ide

app: A fő alkalmazáskód ebben a mappában található

about, dashboard, home, lines, login, password-recovery, signup, auth.service, auth-guard.service, firebase.service, megallok.service, megallok.model mind itt találhatóak.

Komponensek áttekintés:

A projekt modulok és komponensek segítségével jött létre, amelyek akár újrafelhasználhatóak önállóan is, így könnyen karbantartható, cserélhető, javítható az alkalmazás. Itt egy áttekintés mindegyik komponens céljáról:

'Home' komponens: amely az üdvözlő oldalt foglalja magában

'Lines' komponens: amely a metróvonalakat foglalja magában, itt található a fő célja alkalmazásunknak, a felhasználó itt választ metróvonalat, irányt, valamint úticélt

'Dashboard' komponens: az admin felületet foglalja magában a CRUD elvnek megfelelően

'Login' komponens: a bejelentkezést és hitelesítést segítő komponens

'Password Recovery' komponens: elfelejtett jelszó visszaállítását segítő komponens

'Signup' komponens: az új felhasználók itt tudnak beregisztrálni az alkalmazásba

'About' komponens: egy rövid összefoglaló, itt található az alkalmazásunk 'Rólunk' része

Service-ek áttekintés:

Az említett komponenseken kívül még ezek a **Service-**ek találhatóak alkalmazásunkban:

’auth.service’: kezeli a felhasználói beléptetést és hitelesítést

’auth-guard.service’: felhasználói hitelesítés alapján engedélyezi az elérési útvonalak működését

’firebase.service’: együttműködik a Firebase backend-el az adatok tárolása és lekérdezése érdekében

’megallok.service’: a metrómegállókkal kapcsolatos funkciókat szolgáltatja

Valamint:

’megallok.model’: leírja a megállók adatmodelljét

Ezek a komponensek és service-ek járulnak hozzá a projekt funkcióinak működéséhez és a felhasználói élmény megalkotásához, legyen szó hitelesítésről, adatlekérésről vagy általános felhasználói interakciókról.

Környezeti változók

A környezeti változók biztosítják az alkalmazás és a különböző környezetek együttműködését egy megadott konfiguráció alapján. A mi projektünk az ’environments.ts’ fájlt használja e célra. Ez a konfigurációs fájl általában API-végpontokat, Firebase hitelesítési adatokat és egyéb hitelesítéssel, bizalmas információkkal kapcsolatos beállításokat tartalmaz és ezeket ezen fájlon keresztül kezelik egymás között a rendszerek.

Harmadik féllel kapcsolatos integrációk

Firebase: Az alkalmazásunk a Firebase-t olyan háttérszolgáltatásokhoz használja, mint pl hitelesítés, adatbáziskezelés, ezek felhőalapú tárolása. A Firebase konfigurációs beállításai, beleértve az API-kulcsokat és a hitelesítési adatokat, a Firebase-konzolon kezelhetők, és a firebase.service és a kapcsolódó fájlok használatával integrálhatók a projektbe.

Fejlesztési munkafolyamat

A verziókövetést projektünkben a Git támogatta. A Git verziókezelésre szolgál, lehetővé téve a fejlesztők számára a változások nyomon követését, a hatékony együttműködést. A projekt az alapbeállításokat követve **main** ágon található.

Alkalmazásunkban clean code elvek mellett a TypeScript saját irányelvei is segítenek az átláthatóság érdekében. Ilyenek például a változók elnevezésére utaló elvek, típusjegyzetek, behúzások viselkedésének elve stb.

Emellett továbbra is ajánljuk az Angular-rendszer alapvető felépítési architektúrájának követését, legyen szó modulokról, komponensekről, vagy minták/sablonok szintaxisáról.

Projekt komponensei és ezek funkciói

’Lines.component’

Célja:

Az egyik fő komponens, a lines.component megjelenítési funkciója lehetővé teszi a felhasználók számára a különböző metróvonalak információinak megtekintését, vonal kiválasztását, úti cél kiválasztását és a kiválasztott célhoz kapcsolódó képek megtekintését.

Használat:

A felhasználók a megjelenített metróvonalakkal interakcióba léphetnek, a képekre való kattintással egy metróvonalat választanak, majd egy célt, ezután még több információhoz és a célhoz kapcsolódó képekhez férnek hozzá.

Megvalósítás:

A metróvonalak komponens felelős a rendelkezésre álló metróvonalak megjelenítéséért, és lehetővé teszi a felhasználók számára a vonal kiválasztását.
onSelectLine(): Amikor a felhasználó egy metróvonal képére kattint, az onSelectLine() függvény aktiválódik a további választás kezelésére. A metróvonal kiválasztása után megjelenik a ’megallok-container’, amely a kiválasztott vonalhoz tartozó és választható irányokat mutatja.
onSelectDestination(): A felhasználók ezután kiválaszthatnak egy célt, amivel elindítják az onSelectDestination() függvényt a kiválasztás kezelésére. Frissíti a selectedDestination változót és a megfelelő képeket lekéri a destinationPictures tömbben meghatározottak alapján, azaz miután kiválasztottak egy célállomást, megjelennek a célhoz kapcsolódó képek.

Változók:

selectedLine: Egy string típusú változó, amely az aktuálisan kiválasztott metróvonalat tárolja. Ez akkor frissül, amikor a felhasználó a kívánt metróvonal képére kattint.

destinations: Egy tömb, amely információkat tartalmaz a kiválasztott metróvonalhoz társított, elérhető úti célokról. Ezeket az információkat a Firebase-ből kérjük le majd jelenítjük meg a felhasználó számára.

selectedDestination: Egy string típusú változó, amely az aktuálisan kiválasztott célállomást tárolja a kiválasztott metróvonalon belül. Akkor frissül, amikor a felhasználó egy célgombra kattint.

selectedPictures: A kiválasztott célhoz kapcsolódó képek fájlneveit tartalmazó tömb. Ezek a képek jelennek meg a felhasználó számára, hogy vizuális információt is nyújtsanak az úti célról.

destinationPictures: Egy objektum tömb, amely a célhelyek neveit a megfelelő képfájlnevek tömbjére képezi le. Ez lehetővé teszi a képek dinamikus megjelenítését a kiválasztott célhely alapján.

’Dashboard.component’

Célja:

Egyfajta központi vezérlőpultként szolgál a metrómegállók és felhasználók kezeléséhez az alkalmazáson belül.

Használat:

A DashboardComponent egy megfelelő felületet biztosít a metrómegállók kezeléséhez, beleértve a megtekintési, szerkesztési, törlési és új megállók létrehozásának lehetőségeit. A felhasználók oldalszámozási vezérlőkkel navigálhatnak a metrómegállók listájában, és különféle műveleteket hajthatnak végre az egyes megállókon.

Megvalósítás:

ngOnInit(): metódus, amelyet az komponens inicializálásakor hívunk meg. Lekéri egy adott metrómegálló adatait, ha az útvonal paraméterei között szerepel egy azonosító, és lekéri az összes metrómegállót a Firestore-ból.

nextPage(): Növeli az aktuális oldalszámot, hogy a metrómegállók következő oldalára navigáljon.

prevPage(): Csökkenti az aktuális oldalszámot, hogy a metrómegállók előző oldalára navigáljon.

populateEditForm(selectedStop): Feltölti a szerkesztési űrlapot a kiválasztott metrómegálló adataival szerkesztés céljából.

onEditClick(selectedStop): Bekapcsolja a szerkesztési űrlap megjelenítését, és feltölti azt a kiválasztott metrómegálló adataival, amikor a felhasználó a szerkesztés gombra kattint.

RemoveMegallo(megallok): Eltávolítja a kiválasztott metrómegállót az adatbázisból, miután a user megerősítette a törlést egy párbeszédpanelen keresztül.

onSubmitUpdate(): A metrómegálló frissített adatait a szerkesztési lapról elküldi a Firestore-ba.

onSubmitCreate(): Elküldi egy új metrómegálló adatait a létrehozási lapról a Firestore-ba.

toggleCreateForm(): Bekapcsolja a létrehozási űrlap megjelenítését, így hozzáadható új metrómegálló.

signUp(email, password, name): Aszinkron módon regisztrál egy új felhasználót a megadott e-mail címmel, jelszóval és névvel az AuthService segítségével.

Változók:

selectedStopData: Az aktuálisan kiválasztott metrómegálló adatait tárolja szerkesztés céljából.

megallokRef: Hivatkozás a kiválasztott metrómegálló Firestore dokumentumára.

Megallok: A Firestore-ból letöltött összes metrómegállóról szóló információkat tartalmazó tömb.

megallokForm: Új metrómegállók létrehozásának űrlapját képviselő űrlapcsoport.

showEditForm: Logikai változó, amely jelzi, hogy a metrómegállók szerkesztési űrlapja jelenleg látható-e.

showCreateForm: Logikai változó, amely jelzi, hogy az új metrómegállók létrehozási űrlapja jelenleg megjelenik-e.

currentPage: Tárolja az aktuális oldalszámot az oldalszámozáshoz.

pageSize: Megadja az oldalanként megjelenítendő metrómegállók számát.

’Home.component’

Célja:

A HomeComponent az alkalmazás nyitóoldala, amely üdvözli a felhasználókat, és lehetőséget biztosít számukra a metróvonalak felfedezésére és Budapest térképének megtekintésére.

Használat:

Indításkor a HomeComponent üdvözli a felhasználókat, és két fő lehetőséget kínál számukra: Metróvonalakból való választás (amely átirányít a LinesComponent oldalra) vagy budapesti metróvonalakról szóló térkép tanulmányozása. A Budapest térkép képére kattintva az egy modális átfedésben kinagyítható, így a felhasználók részletesebben is felfedezhetik a térképet. Ha a kinagyított térképen vagy a fedvényen kívül bárhová kattint, bezárja a modált, és visszatér a kezdőlapra.

Megvalósítás:

openModal(): true-ra állítja az isModalOpen jelzőt, megjelenítve a térképet egy modális felületen.

closeModal(): Az isModalOpen jelzőt false értékre állítja, így elrejtve a térkép modális felületét.

Változók:

isModalOpen: Logikai változó, amely a térkép láthatóságát szabályozza.

’Login.component’

Célja:

A LoginComponent bejelentkezési funkcionalitást biztosít a felhasználók számára, e-mail-cím és jelszavukkal, vagy Google hitelesítéssel.

Használat:

A felhasználók megadhatják e-mail címüket és jelszavukat a bejelentkezéshez. A "Bejelentkezés" gombra kattintva elindítja a login() metódust, amely a megadott hitelesítő adatokkal hitelesíti a felhasználót. A "Bejelentkezés Google-lal" gombra kattintva elindítja a loginWithGoogle() módszert a Google hitelesítéshez. Sikeres hitelesítés esetén a felhasználók a kezdőlapra kerülnek. A felhasználók az "Elfelejtetted a jelszavad?" gombra kattintva a jelszó-visszaállító oldalra, az „Új fiók létrehozása” gombra kattintva pedig a regisztrációs oldalra léphetnek.

Megvalósítás:

login(): Meghívja a hitelesítés login() metódusát a megadott e-mail címmel és jelszóval az e-mail/jelszó hitelesítéshez.

loginWithGoogle(): Elindítja a hitelesítési szolgáltatás loginWithGoogle() metódusát a Google hitelesítéshez.

logout(): Elindítja a logout() metódust, így a felhasználó kijelentkezik az alkalmazásból.

Változók:

email: a felhasználó által megadott e-mail címet tárolja

password: a felhasználó által megadott jelszó értékét tárolja

’password-recovery.component’

Célja:

A jelszóvisszaállítási komponensen a felhasználóknak lehetőségük adódik elfelejtett jelszó esetén annak visszaállítására, amiről emailt kapnak.

Használat:

A felhasználók be tudják írni email címüket, majd a Jelszó visszaállítása gombra kattintva meghívják a resetPassword() metódust, így elindítva a visszaállítási folyamatot. Sikeres lefutás után a kapott emailben található instrukciók vezetik végig a felhasználót a sikeres jelszóvisszaállításon.

Megvalósítás:

resetPassword(): Elindít egy jelszóvisszaállítási folyamatot a megadott email címre való visszaállítási email küldése által.

’about.component’

Célja:

A rólunk rész rövid információt oszt meg a készítőkről és a projektről, készítés évéről. Különösebb backend nem dolgozik mögötte.

’signup.component’

Célja:

A komponens célja, hogy regisztrációs felületet biztosítson a felhasználók számára.

Használat:

A név mező egy kötelező mező, ahol a felhasználók regisztrációhoz használni kívánt nevüket adják meg. Ezután kell megadniuk az email címüket, ami csak email formátum lehet (minta@minta.com). Legvégül egy jelszó választása következik, minimum 6 karakterből kell álljon. Amennyiben minden feltételnek eleget tesz a felhasználó, a regisztráció gomb aktívvá válik és rákattintva tudjuk elindítani a regisztrációs metódust.

Megvalósítás:

validateEmail(): validálja a megadott email cím helyességét (minta@minta.com)

signUp(): elindítja a regisztrációs folyamatot a megadott adatok alapján. Átirányítja a főoldalra a sikeresen regisztrált felhasználót. Tájékoztat az esetlegesen előforduló hibákról.

onFormChange(): frissíti az űrlap érvényességi állapotát, amikor a mezők megváltoznak.

Változók:

Email: a felhasználó által megadott email címet tárolja

Password: a felhasználó által megadott jelszót tárolja

Name: a felhasználó által megadott nevet tárolja

isFormValid: az űrlap általános helyességére vonatkozó logikai változót tárolja

signInForm: az űrlap érvényesítéséhez és kezeléséhez használt NgForm példányt jelöli

’app.component’

Célja:

Az app komponens az alkalmazás úgynevezett gyökérkomponenseként szolgál, biztosítva az alap elrendezést és a navigációt.

Használat:

Ez a komponens automatikusan betöltésre kerül az alkalmazás indításakor. Beállítja a fő elrendezési struktúrát és a navigációs linkeket, hivatkozásokat.

Megvalósítás:

Megjelenít egy navigációs sávot az alkalmazás különböző részeire mutató hivatkozásokkal. Kijelentkezési gombot biztosít, ha a felhasználó be van jelentkezve. Tartalmaz egy router outlet-et a különböző komponensek betöltéséhez.

Változók:

authService: AuthService: Az AuthService egy példánya, a felhasználói hitelesítéshez fontos

isLoggedIn$: Egy observable az AuthService-ből, amely jelzi, hogy a felhasználó jelenleg be van-e jelentkezve.

Modulok, Modellek, Service-ek

’app.module’

Célja:

Az AppModule az Angular alkalmazás gyökérmoduljaként szolgál, amely konfigurációt biztosít például a routinghoz is.

Használat:

Ez a modul automatikusan betöltődik az alkalmazás indításakor. Konfigurálja az alkalmazás összetevőit, útvonalait, hitelesítését és a Firestore adatbázist.

Megvalósítás:

Az app.module deklarálja az alkalmazásban használt komponenseket.Importálja a szükséges Angular modulokat az formokhoz, az routinghoz és a Firebase-integrációhoz. Útvonalakat határoz meg a különböző komponensek közötti navigációhoz. Konfigurálja a Firebase hitelesítést és a Firestore adatbázist. Authguard-ot biztosít a hitelesítést igénylő útvonalak védelmére.

Útvonalak konfigurációja az app.module-ban:

Célja:

Az routeok konfigurációja meghatározza az útvonalakat és a megfelelő összetevőket az alkalmazáson belüli navigációhoz.

Használat:

A RouterModule.forRoot() metódus használja a Routes Configuration-t az AppModule importálási tömbjében az alkalmazás útvonalainak konfigurálásához.

Funkcionalitás:

Útvonalakat határoz meg az alkalmazás egyes összetevőihez, megadva az URL elérési utat és a megfelelő megjelenítendő komponenst. Az Angular útválasztási funkcióját használja a különböző összetevők közötti navigáláshoz az URL-útvonalak alapján.

Változók:

appRoutes: Útvonal-konfigurációk tömbje, amely megadja az URL elérési utat és a megfelelő megjelenítendő komponenst.

Firebase Integráció az app.module-ban

Célja:

A Firebase integráció konfigurációt biztosít a Firebase szolgáltatások, például a Firestore és a hitelesítés Angular alkalmazásba történő integrálásához.

Használat:

A Firebase szolgáltatások inicializálására és konfigurálására szolgál az Angular alkalmazáson belül.

Funkcionalitás:

Importálja az AngularFireModule és AngularFirestoreModule modulokat a Firestore integráció engedélyezéséhez.Inicializálja a Firebase-t a környezeti fájl megadott konfigurációjával. A Firestore-t és a hitelesítési szolgáltatásokat a provideFirestore() és a provideAuth() metódusok használatával konfigurálja.

Változók:

Environments.firebaseConfig: A Firebase hitelesítő adatait és beállításait tartalmazó konfigurációs objektum az environments.ts fileból.

AuthGuard az app.module-ban és ’auth-guard.service’

Célja:

Az Authentication Guard (AuthGuard) a felhasználó hitelesítési állapotának ellenőrzésével megakadályozza az illetéktelen hozzáférést bizonyos útvonalakhoz.

Használat:

A hitelesítést igénylő útvonalak védelmére szolgál, biztosítva, hogy csak hitelesített felhasználók férhessenek hozzájuk.

Funkcionalitás:

Megvalósítja a canActivate felületet annak meghatározására, hogy a felhasználó hitelesített-e. A Firebase hitelesítési szolgáltatást használja a felhasználó hitelesítési állapotának ellenőrzésére. RxJS operátorokat használ az aszinkron hitelesítési ellenőrzések kezelésére. Observable<boolean> értéket ad vissza, amely jelzi, hogy a felhasználó hitelesített-e vagy sem. A nem hitelesített felhasználókat átirányítja a bejelentkezési oldalra, ha védett útvonalakat próbálnak elérni.

Változók:

AuthGuard: Hitelesítési szolgáltatás, amely a hitelesítést igénylő útvonalak védelméért felelős.

’auth.service’

Célja:

Az AuthService hitelesítési funkciókat biztosít az alkalmazás számára, beleértve a bejelentkezést, a kijelentkezést és a felhasználói regisztrációt. A Firebase Authentication szolgáltatással együttműködve kezeli a felhasználói hitelesítést.

Használat:

Olyan összetevőkbe és szolgáltatásokba van beillesztve, ahol hitelesítéssel kapcsolatos feladatokat kell végrehajtani, például bejelentkezést, kijelentkezést vagy felhasználói regisztrációt.

Funkcionalitás:

Inicializálja a Firebase hitelesítést, és figyeli a hitelesítési állapot változásait. Metódusokat biztosít a felhasználók hitelesítéséhez, beleértve az e-mail-címmel/jelszóval történő bejelentkezést, a Google-lal való bejelentkezést és a felhasználói regisztrációt. Kezeli a felhasználó hitelesítési állapotát, és Observable-t hoz létre, hogy értesítse az ezt figyelőket a hitelesítési állapot változásairól. Sikeres hitelesítés vagy kijelentkezés esetén átirányítja a felhasználókat a megfelelő útvonalakra.

Változók:

auth: Firebase Auth példány a hitelesítési műveletek kezelésére.

user: Az aktuálisan hitelesített felhasználó adatait tárolja.

isLoggedInSubject: A hitelesítési állapot belső nyomon követésére használt ún. BehaviorSubject.

isLoggedIn$: Observable, amely kiadja a hitelesítési állapotot az azt figyelőknek.

’firebase.service’

Célja:

A FirebaseService megkönnyíti a kommunikációt a Firebase Firestore-ral a metrómegállókkal és célállomásokkal kapcsolatos adatok lekérése érdekében. Metódusokat biztosít egy adott metróvonal úticéljainak lekérésére, egy adott vonal irányainak lekérésére, és egy adott irány megállóinak lekérésére.

Használat:

Szúrja be a FirebaseService-t olyan összetevőkbe vagy szolgáltatásokba, ahol a metrómegállókkal és célállomásokkal kapcsolatos adatokat kell lekérni és felhasználni.

Funkcionalitás:

getDestinationsForLine(line: string): Observable<any[]>: Lekéri egy megadott metróvonal célállomásait a Firestore-ból. A célhelyek tömbjét adja vissza.

getDirectionsForLine(line: string): Observable<string[]>: Irányokat kér egy megadott metróvonalhoz a Firestore-ból. Egyedi irányok tömbjét adja vissza

getStopsForDirection(irány: string): Observable<string[]>: Lekéri a megállókat egy meghatározott irányba a Firestore-ból. Megállónevek tömbjét adja vissza.

Változók:

firestore: Az AngularFirestore példánya, amelyet a Firestore-ral való interakcióhoz használunk.

’megallok.service’

Célja:

A MegallokService függvényeket, metódusokat biztosít a Firestore-ral való interakcióhoz a metrómegállókhoz kapcsolódó CRUD (Create, Read, Update, Delete) műveletek végrehajtásához.

Használat:

Injektálható a MegallokService olyan komponensekbe vagy service-ekbe, ahol a metrómegállókkal kapcsolatos CRUD műveleteket kell végrehajtani.

Funkcionalitás:

getMegalloDoc(id): Egyetlen metrómegálló dokumentumot kér le a Firestore-ból a megadott azonosító alapján. A dokumentum értékének megfelelő Observable-t ad vissza.

getMegalloList(): Lekéri az összes metrómegálló dokumentum listáját a Firestore-ból. Pillanatkép-változásokat ad vissza Observable formájában.

createMegallo(megallok: Megallok): Új metrómegálló dokumentumot hoz létre a Firestore-ban a megadott Megallok objektum segítségével. Resolved promiset ad vissza, ha a művelet sikeres.

deleteMegallo(megallok): Töröl egy metrómegálló dokumentumot a Firestore-ból a megadott Megallok objektum azonosítója alapján. Resolved promiset ad vissza, ha a törlés sikeres.

updateMegallo(megallok: Megallok, id): Frissít egy meglévő metrómegálló dokumentumot a Firestore-ban a Megallok objektum megadott adataival és a megadott azonosítóval.

Változók:

angularFirestore: Az AngularFirestore példánya, amelyet a Firestore-ral való interakcióhoz használnak.

’megallok.model’

Célja:

A Megallok modell egy-egy metrómegálló adatbázisban szereplő modelljét jelenti az alkalmazásban. Meghatározza a metrómegálló objektumának szerkezetét különféle tulajdonságokkal, mint például id, line, irány, név, doorInfo és kijarat.

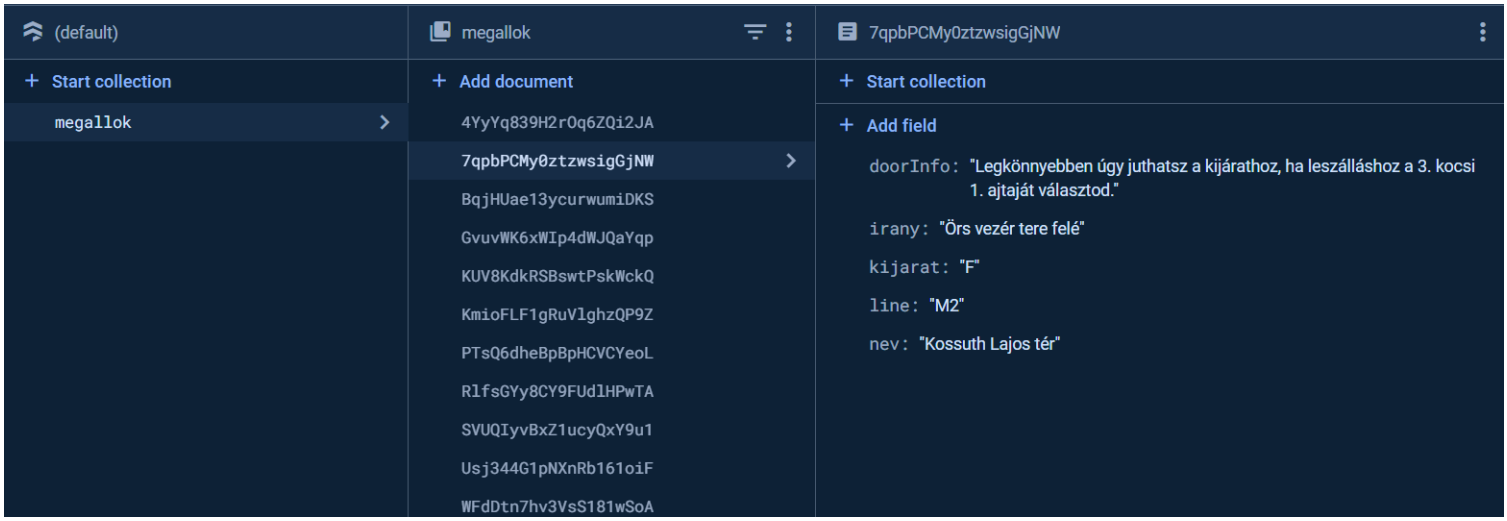
```
export class Megallok{
    id:string;
        line: string;
        irany: string;
        nev: string;
        doorInfo: string;
        kijarat:string;
}
```

Használat:

A Megallok modellt metrómegálló objektumok példányainak létrehozására használják az alkalmazásban. Szabványos struktúrát biztosít a metrómegállók adatainak tárolására és kezelésére.

Adatbázis modell

Az alábbi adatbázist használjuk, ennek a rendszerét a Firebase szolgáltatja:



A ’megallok’ elnevezésű collection tartalmazza a különböző autoID-val ellátott megállókat, célokat. Minden megálló, úti cél különböző field tulajdonságokkal rendelkezik. Ez a dokumentáció ennek megértésére szolgál.

AutoID struktúra:

Az autoID segít, hogy minden dokumentumnak automatikusan egyedi azonosítója legyen. Ez leegyszerűsíti az adatokhoz való hozzáférést. Az autoID-val ellátott dokumentumok struktúrája az alábbi mezőket tartalmazza:

nev(string): Megálló vagy cél elnevezése. (pl. ’Oktogon’)

irany(string): Az utazásra használt metró iránya, végállomása. (pl. ’Mexikói út’)

line(string): A metróvonal számozása (pl. ’M3’)

doorInfo(string): Leszálláshoz legmegfelelőbb ajtó és ezzel kapcsolatos információkat tartalmazza (pl. ’Az átszálláshoz legelőnyösebb, ha a 3. kocsi 1. ajtaját választod!’)

kijarat(string): Ajánlott feljáró betűjele az aluljáróból (pl. ’A’)

Célja:

Az adatbázisrendszer célja, hogy egy átfogó, könnyen elérhető, könnyen szerkeszthető, skálázható adatbázisban tartsuk a projekthez tartozó adatokat. Egyszerűségének köszönhetően később is bővíthető funkciókkal, adatokkal, táblákkal.

Végpontok

Végpont	Metódus	Azonosítás	Leírás
/megallok/{id}	get	igen	Megálló lekérdezése
/megallok	get	Igen	Megállók lekérdezése
/megallok	post	Igen	Megállók hozzáadása
/megallok	put	Igen	Megállók szerkesztése
/megallok	delete	Igen	Megállók törlése
/register	post	Nem	Felhasználó regisztrációja
/login	post	Nem	Felhasználó bejelentkezése
/logout	post	Igen	Felhasználó kijelentkezése

Általános működés

A fenti táblázat áttekintést nyújt a végpontokról és azok rendszeren belüli működéséről. A kényes műveletek, információk megszerzéséhez, használatukhoz autentikáció szükséges. Ilyen például a megállók szerkesztése, lekérdezése, hozzáadása, törlése, felhasználó kijelentkezése. Nem szükséges autentikáció a regisztrációhoz és a bejelentkezéshez. Későbbi projektverzióval kapcsolatos tervekben átdolgozásra kerülnek a végpontok is. Ezen adatok küldését fogadását, service-ek és metódusok biztosítják a projektek során (lásd: fentebb Komponensek rész).

’Tesztelés’

Teszt 1. Felhasználói regisztráció

Cél:

A regisztrációs űrlapon keresztül felhasználó érvényes, működő hozzáadása történik.

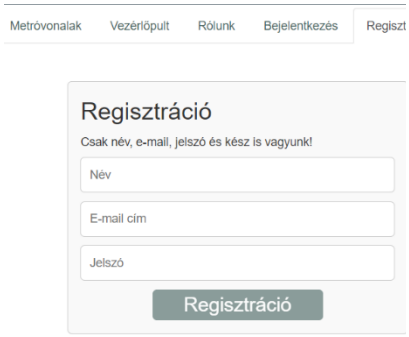
Teszt lépései:

1. A Regisztráció gombra kattintva navigáljon az alkalmazás az űrlaphoz. Ez sikeresen megtörtént.
2. Az alkalmazás sikeresen észlelje és kitöltésre kerüljenek a mezők (név, email, jelszó). Ez a pont is sikeresen megtörtént.
3. Regisztráció gombra kattintva sikeresen hozzáadásra kerül a felhasználó a rendszerhez, és átirányítódik a routing által megadott megfelelő oldalra. Ez a pont is sikeresen megtörtént.

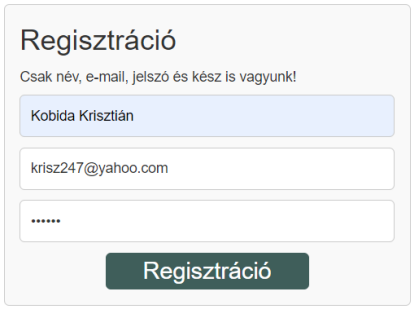
Teszt adatok:
Név: Kobida Krisztián
Email: krisz247@yahoo.com
Jelszó: 123456

Tesztkörnyezet:
Böngésző: Google Chrome 123.0.6312.122 (Hivatalos verzió) (64 bites)
Eszköz: Asztali PC Win 11

Sikeres navigálás (teszt 1. pont)**:**

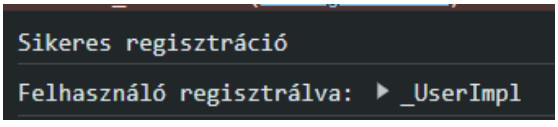


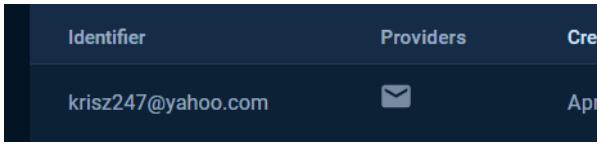
Kitöltés (teszt 2. pont):



Átirányítás (teszt 3.pont):

Tesztünk sikerességét a console log is alátámasztja, valamint a firebase konzoljában megtalálható user is:





Teszt 2. Metrómegállók szerkesztése

Célja:

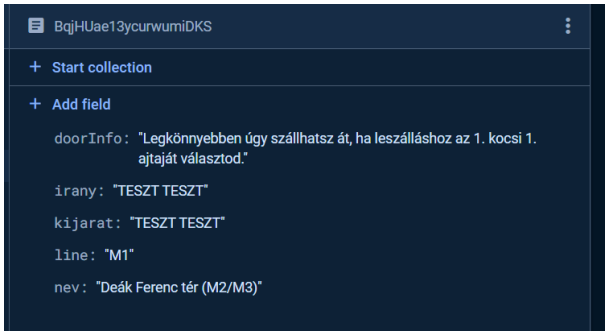
A metrómegállók szerkesztési metódusainak és adatbázissal való együttműködésének tesztelése.

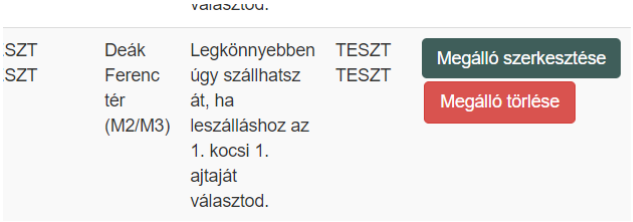
1. Navigálás a vezérlőpultra. **Sikeresen megtörtént.**

2	M2	Örs vezér tere felé	Kossuth Lajos tér	Legkönnyebben úgy juthatsz a kijárathoz, ha leszálláshoz a 3. kocsi 1. ajtaját választod.	F	Megálló szerkesztése Megálló törlése
3	M1	Mexikói út	Deák Ferenc tér (M2/M3)	Legkönnyebben úgy szállhatsz át, ha leszálláshoz az 1. kocsi 1. ajtaját választod.		Megálló szerkesztése Megálló törlése

2. Szerkeszteni kívánt metrómegálló melletti „Megálló szerkesztése” gombra való kattintás és megjelenő űrlap kitöltése. **Sikeresen megtörtént**

3. „Megálló szerkesztése” gomb után a frontenden és az adatbázisban is megváltozik az átírt mező. **Sikeresen megtörtént.**





Teszt adatok:
Metró iránya: TESZT TESZT
Kijárat betűjele: TESZT TESZT

Tesztkörnyezet:
Böngésző: Google Chrome 123.0.6312.122 (Hivatalos verzió) (64 bites)
Eszköz: Asztali PC Win 11

Ismert hibák, fejlesztésük folyamatban

Admin – User, korrekt management:
A jelenlegi megvalósításból hiányzik a teljesen működőképes admin-user felügyeleti rendszer, ami korlátozza a felhasználói hozzáférés-szabályozást és az adminok által ellátott feladatok összefüggését.

Elfelejtett jelszó komponens:
Az elfelejtett jelszó komponens nincs teljesen integrálva, így nem teljesen működőképes.

Adatbázis-állomány:
Az adatbázisban nincs elegendő adatállomány, ami korlátozott vagy hiányos információkat eredményezhet tesztelési vagy demonstrációs helyzetekben.

Automatikus üdvözlő e-mail:
A regisztrációs rendszerbe implementálni egy üdvözlő emailt, amely lényeges információkat vagy esetleg utasításokat tartalmaz.

Összefoglalás

A projektünk megvalósításával kapcsolatban kiemelendő, hogy bár nem sikerült az elképzeléseinket megvalósítani teljes mértékben, így is sok olyan dolgot tanultunk és olyan dolgokkal találkoztunk, amelyek hasznos tapasztalatokkal és tudással bővítették a már megszerzett ismereteinket. Legnagyobb kihívásként arra tekintünk, hogy az ötlet mint maga, nem hordoz magában sokféle, a projekttel kapcsolatos implementálható funkciót, így nehéz lett volna túlbonyolítani az elképzelésünket. Ez köszönhető a többszöri ötletváltásnak, csoportstruktúránk változásának, idő-management hiányának. Projekttel kapcsolatos jövőbeli tervek között van az adatbázis feltöltése, egy átdolgozott, modernebb, felhasználóbarátabb, letisztult frontend design, komplexebb adatbázismodell kiépítése, több, hasznos funkció implementálása az alkalmazásba.

Köszönet

Ezúton szeretnénk köszönetet mondani

Bálint Dezső, Jáger Attila és Józsa Béla

tanárainknak a rengeteg segítségért, türelemért, megértésért, szakmai rálátásukért és tanácsaikért.