myCondominiums

Fejlesztői dokumentáció

2024

<https://mycondo.hu>

**A fejlesztés célja**

Egy olyan társasháznyilvántartó weboldal készítése, amelyen keresztül a lakók leadhatják a mérőóráik állását online felületen.

Az alkalmazás ötletét az adta, hogy a készítők maguk is laktak társasházban és tapasztalatuk szerint még mindig a faliújságra kiírós illetve a közös képviselő ládájába bedobott cetlik formájában kerültek leadásra a lakáson belüli mérőórák havi állásai.

A mai világban szinte mindent online el lehet intézni ezért merült fel az ötlet, hogy aki szeretné az akár kényelmesen az otthonából intézhesse ezt az ügyét is, az ódivatú módszerek helyett.

A webalkalmazás emellett alkalmas a lakók adatainak nyilvántartására is, a kezelője közzé tehet információkat, ami a lakók számára fontos lehet. Feltölthet fájlokat, például meghatalmazás mintákat, közgyűlési beszámolókat és egyéb közérdekű dokumentumokat.

**Felhasznált technológiák**

**Frontend**

Angular 16 (Typescript alapú keretrendszer)

-Ingyenes, nyílt forráskódú rendszer, a Google fejlesztése ezért valószínű, hogy sokáig lesz hozzá támogatás.

**Backend**

Google Firebase

-Szintén Google fejlesztés, a legtöbb szolgáltatása az alkalmazás számára ingyen elérhető. Segítségével el lehet hagyni a szerver oldali programozási feladatokat, mivel ez a szolgáltatás lesz a szerverünk.

Php

-A felhasználók részére kiküldött email-ek miatt szükséges. Azért esett erre a választás mert Magyarországon kevés szolgáltató kínál elérhető áron olyan webtárhelyet ami kiszolgál Java vagy Node Js rendszereket.

**A készítéshez használt szoftverek**

**Visual Studio Code 1.87.2**

-Szinte minden nyelvet tud, ami egy átlagos fejlesztőnek szükséges, számtalan ingyenes bővítmény tölthető le hozzá.

**XAMPP**

-Az Apache szerver, a PHP futtatásáshoz szükséges.

**Google Chrome**

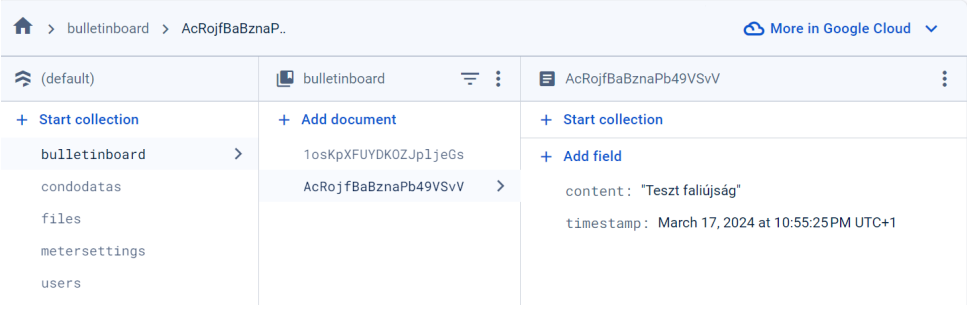
-Mivel ez a felhasználók körében a legnépszerűbb böngésző, a program futásának az ellenőrzése főleg ezen zajlott a Microsoft Edge és a Firefox mellett.

**Adatbázis szerkezet**

A felhasználók azonosítása a Firebase Authentication szolgáltatásával lett megvalósítva. A belépéshez az Email/Password kombináció felelt meg a céloknak mivel az alkalmazásba csak az adminisztrátor által felvett személyek jelentkezhetnek be így a többi bejelentkeztetési szolgáltatások /Facebook, Google, GitHub stb./ nem feleltek meg.

Az adatok kezelésére szükség volt még 5 *collection*-ra, amik a következők:

-**bulletinboard** /faliújság, a lakóknak szóló hírek tárolására/



mentés:

firestore.collection('bulletinboard').add({

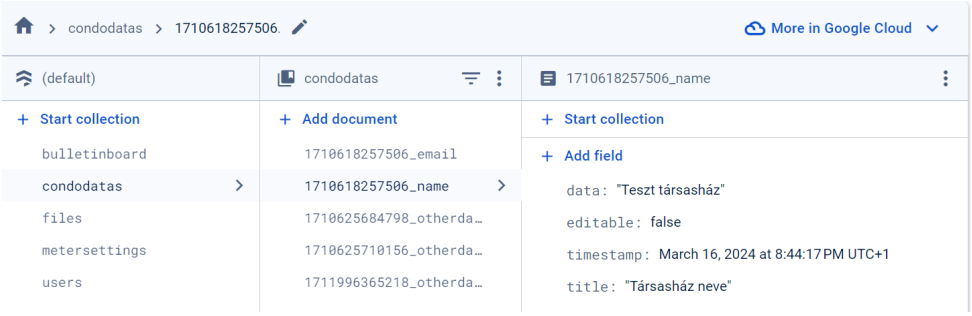
content: string,

timestamp: timestamp

}

);

-**condodatas** /a társasház adatainak a tárolására/



Az első két *document*-et az alkalmazás hozza létre, mivel ezeknek a ’title’ értéke nem módosítható. A *document* id-ket a *timestamp.getTime().toString()* függvények alakítják át hogy a létrehozás sorrendjében szerepeljenek a *collection*-ban.

mentés:

firestore.collection('condodatas').doc(docName).set({

        title: string,

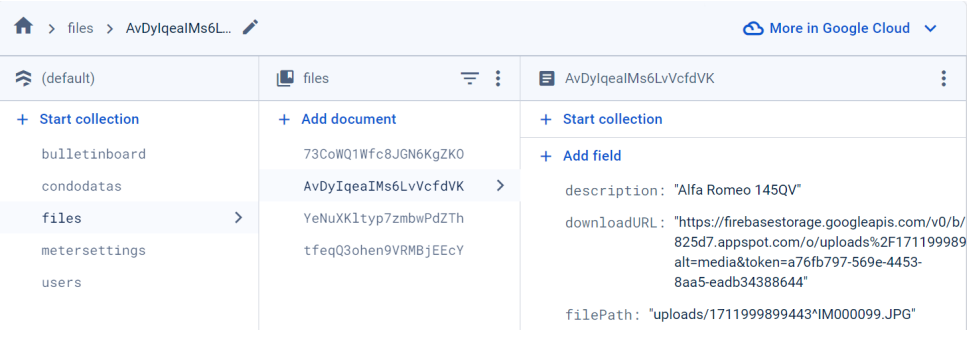
        data: string,

        editable: boolean,

        timestamp: timestamp

      });

-**files** /fájlok tárolására/

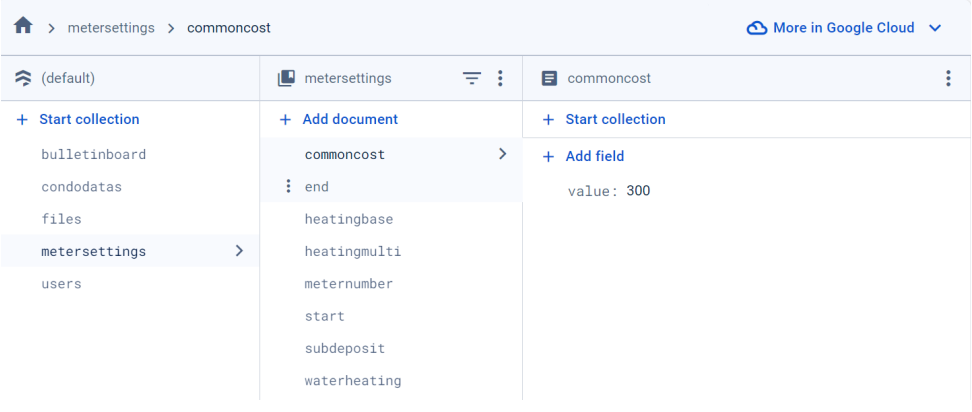


A fájlok tárolására a Firebase Storage build-ben történik.

mentés:

firestore.collection('files').add({ downloadURL, filePath, description })

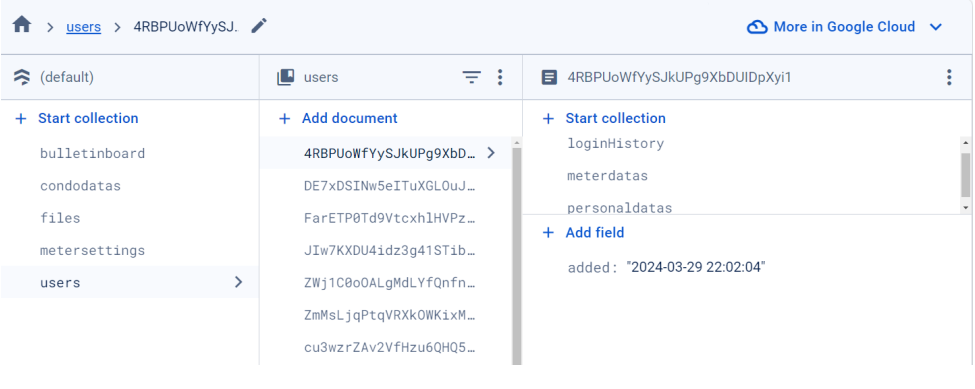
**metersettings** /a mérőórák és költségadatok tárolására/



mentés:

firestore.collection('metersettings').doc(docName).set({ value: number });

**users** /felhasználók adatainak tárolására/



Ebben a *collection*-ban ki lett használva a Firestore azon lehetősége, hogy egy *collection*-on belül létre lehet hozni további *collection*-okat, így ugyanazon a *document* alatt lett tárolva a felhasználó bejelentkezési időpontjai, a leadott óraállásai és a személyes adatai.

mentés új lakó esetén:

firestore.collection('users').doc(uid).collection('personaldatas').doc('datas');

set({ name, email, phone, building, floor, door, squaremeter, balance, hasMeter, isAdmin });

string: name, email, phone, building, floor, door

num: squaremeter, balance

boolean:hasMeter, isAdmin

firestore.collection('users').doc(uid).collection('loginHistory').doc('datas');

set({});

firestore.collection('users').doc(uid);

set({ added: currentDateAndTime });

mapData = { meterColdOne: 0, meterColdTwo: 0, meterHotOne: 0, meterHotTwo: 0, amountOfHeat: 0 };

doc = this.firestore.collection('users').doc(uid).collection('meterdatas').doc('datas');

set({ ['2000-01']: mapData });

ezzel létrejön egy új lakó az alapadatokkal.

**Könyvtár és fájl szerkezet**

Az Angular lehetőségeit kihasználva a kód szerkezete úgy lett kialakítva, hogy az *src/app* mappán belül három modul lett létrehozva:

-admin

-shared

-user

Ezeknek külön *routing*-ja van, ami csak a modulon belüli átirányításokért felel.

Az *app-routing.modules.ts* fájl csak a modulok közti szétosztást segíti.

const routes: Routes = [

  { path: '', redirectTo: 'login', pathMatch: 'full' },

  { path: 'admin', loadChildren: () => import('./admin/admin.module').then(m => m.AdminModule) },

  { path: 'lako', loadChildren: () => import('./user/user.module').then(m => m.UserModule) },

  { path: 'login', component: LoginComponent }

];

A fenti kódból látható, hogy a programnak két fő része van, az adminisztrációs felület és a felhasználói, azaz a lakók által használt felület.

Azért, hogy kizárjuk azt a lehetőséget, hogy az user az url módosításával elérje az admin felületet, ún. *guard* lett alkalmazva. A *guard*-ok a modulok *routing*jain keresztül kerülnek meghívásra, az alábbi kódrészlet szerint:

const adminRoutes: Routes = [

    {

        path: 'admin',

        canActivate: [AdminGuard],

        children: [

            { path: 'home', component: AdminhomeComponent },

            { path: 'bboard', component: BboardComponent },

            { path: 'users', component: UsersComponent },

            { path: 'meters', component: MetersComponent },

            { path: 'settings', component: SettingsComponent },

            { path: 'uploadfiles', component: UploadfileComponent },

            { path: 'update/:id', component: UpdateuserComponent },

            { path: 'add', component: AdduserComponent },

            { path: '\*\*', component: NotfoundComponent },

        ]

    },

];

A *guard* ellenőrzi a bejelentkezett felhasználó státuszát és ha nincs jogosultsága elérni az adott url-t akkor átirányítja a bejelentkezési felületre.

A modulokon belül további mappák vannak annak érdekében, hogy az azonos szerepű fájlokat el lehessen különíteni. pl. *components, models* és *services*.

Ha az MVC mintával hasonlítjuk össze akkor a *components* tartalmazza a *kontrollereket,* amik a felhasználói interakciókért a *view*-eken keresztül felelősek.

Az adatok feldolgozását a *services* mappában található typescript fájlok végzik.

**A felület kinézete/formázása**

A kimeneti HTML kód formázása a *Boostrap 5.3* segítségével történt, sok segítséget nyújtott a beépített lehetőségeivel. Például a *grid system* és a pagination használatával. Az oldal kinézete saját *css* kóddal készült, ha több komponens használja ugyanazt a formázást akkor az adott komponens *styleUrls* importja a *shared/css* mappában található css fájlokat importálja.

Szintén hasznos volt a *modal* funkció, amit több komponens használ, illetve a *navbar* ami segítette a reszponzivitását a menünek.

**Algoritmusok/kódok**

**Admin felület**

Az adminisztrációs felület összes menüpontja úgy lett megvalósítva, hogy a kijelző felső részén rögzítve van a menüsor, mobilnézetben pedig „hamburger menü”. A lap alján szintén rögzítve a *footer* komponens.

<app-adminmenu></app-adminmenu>

…

tartalom

…

<app-adminfooter></app-adminfooter>

A „**Főoldal**” /*adminhome*/ információkkal szolgál a további menűpontok lehetőségeiről.

A következő menü elem a „**Faliújság**” /*bboard*/ ami a megosztandó tartalmak létrehozásáért és szerkesztéséért fele.

Az Angular lehetőségeit kihasználva a *bboard.component.ts* fájl használja az Angular Editor-t ami egy WYSIWYG szerkesztő sokféle beállítási lehetőséggel.

editorConfig: AngularEditorConfig = {

    toolbarHiddenButtons: [

      [

      ],

      [

        'insertImage',

        'insertVideo',

        'insertHorizontalRule',

        'toggleEditorMode'

      ]

    ],

    editable: true,

    spellcheck: true,

    height: 'auto',

    minHeight: '200px',

    placeholder: 'Ide írd a tartalmat...',

    translate: 'no',

    sanitize: false,

  };

Itt az admin által bevitt adatokat jelenítjük meg, ezért szükség volt egy függvényre, ami biztonságosan jeleníti meg a HTML tartalmat.

 sanitizeHTML(html: string): SafeHtml {

    return this.sanitizer.bypassSecurityTrustHtml(html);

  }

<div [innerHTML]="sanitizeHTML(bulletin.content)"></div>

A *DomSanitizer* szolgáltatás segítségével jelzi az Angularnak hogy a bemenet biztonságos és ne vegye figyelembe a beépített szűrőket.

A *formatTimestamp* metódus egy *timestamp*-ot kap paraméretként amit átalakít

’yyyy MM-dd HH:mm’ formátummá, ami a faliújság létrehozásának a dátuma.

Ebben a fájlban található még a mentés, módosítás, módosításból kilépő és a *modal*-t kezelő metódusok. Az adatokkal való manipulációs eszközök a *shared/services/* mappában levő *bboard.service.ts* fájlban vannak.

***getAllBulletinBoardData()***-> lekéri a Firestoreból az összes adatot, a létrehozás dátuma szerint, fordított sorrenben.

Visszatér: (tömb elem száma)[{content: „Faluújság tartalom”, id: „valamiId”, timestamp{seconds: s, nanoseconds: ns}}]

***saveToDatabase(selectedBulletin:any, editorContent:any)*** -> ha a *selectedBulletin* érték nem üres akkor módosítja az adatokat, ha üres akkor új faliújságként menti el az adatbázisba.

***deleteBulletin(bulletinId:any)*** -> törli az adatbázisból a paraméterként kapott id-jű dokumentumot.

„**Lakók**” menüpont

A lakók adatainak kilistázása és manipulálása

***users.component.ts*** ->

fő változók:

error:any; -> hibaüzenet tárolása

users: any[] = []; -> lakók adatai

filteredUsers[]: any[] = []; -> keresés szerint szűrt lakók

 modalRefDelete: MdbModalRef<UserdelconfirmComponent> | null = null;

  modalRefEdit: MdbModalRef<UpdateuserComponent> | null = null;

  modalRefNew: MdbModalRef<AdduserComponent> | null = null;

-> a megfelelő komponensek referencia objektumai.

fő metódusok:

getUsers() -> a sevice-n keresztül lekéri a lakók adatait, meghívja a névsorba rendező metódust. Nincs visszatérési értéke.

onPageSizeChange() -> ha az egy oldalon megjeleníthető lakók számában változás történik akkor beállítja az első oldalra.

sortUsersByName() -> névsorba rendezi az users tömb tartalmát.

visszatérési értéke: users[];

filterUsers() -> ha a kereső mezőbe érték kerül akkor az alapján szűri az users tömböt. Nincs visszatérési érték.

openDeleteModal(username: any, userid:any) -> átadja a törlést megerősítő modalnak az értékeket.

openEditModal(user:any) -> átadja a szerkesző modalnak az adott user adatait. openNewModal() -> egy új lakó létrehozásához nyit egy modalt

**adduser.component.ts** /listusers modal/

fő változók:

formFields:any; -> a form-fields.data.ts tartalma kerül bele, ami az új lakó formjának a formázását tartalmazza, ami alapján renderel a HTML.

nameError = false; -> hibaüzenet üres névmező esetén

emailError = false; -> hibaüzenet érvénytelen email cím esetén

fő metódus:

onSave() -> ha ki van töltve a név mező és érvényes az email cím akkor elküldi a service-nek az adatokat. Utána beállítja a lakót értesítő email adatait, amit átad az *emailService*-nek.

**email.service.ts**

fő változó:

url -> az email küldő php fájl elérési útja a szerveren

fő metódus:

sendMail(emailData:any) ->az angular HttpClient eszközét használja:

sendEmail(emailData: any) {

    const headers = new HttpHeaders({

      'Content-Type': 'application/x-www-form-urlencoded'

    });

    this.http.post(this.url, 'emailData=' + encodeURIComponent(JSON.stringify(emailData)), { headers }).subscribe(response => {

      console.log('Email küldése sikeres:', response);

    }, error => {

      console.error('Hiba történt az email küldése során:', error);

    });

  }

emailData: json {recipient: „címzett email címe”,

recipientName: „címzett neve”,

subject: „tárgy”,

message: „üzenet tartalma”}

A sendmail.php így postként fogadja az adatokat.

**updateuser.component.ts** /listusers modal/

fő változók:

formFields, nameError, emailError -> ua. mint az addusernél

fő metódus:

onUpdate() -> működése hasonló mint az onSave-nek,

változóinak, metódusainak a paraméterei ugyanazok.

***users.service.ts***

fő metódusok:

addUser(userData:any) -> paraméterként megkapja az űrlap adatait, vizsgálja hogy létezik-e már az adott email címmel felhasználó, létrehoz egy új dokumentumot a firebase  *users* kollekciójában, és beállítja a három alkollekciót /personaldatas, loginhistory, meterdatas/

userData: name, email, phone, building, floor, door, squaremeter, balance, hasMeter (boolean), isAdmin(boolean)

getAllData():observable<any> -> lekéri az *users* kollekcióból a *personaldatas* és *meterdatas* értékeket és egyesíti egy tömbben azokat.

upDateUser(userData:any,userId\_string) -> frissíti a felhasználó adatait, ha változás van bennük

userData: ugyanaz mint az addUser metódusnál

deleteUser(userId:string) -> törli az *userId* dokumentumot és az alkollekcióit. A firestore nem törli automatikusan az alkollekciót ezért végigmegy rajtuk és egyenként törli.

generatePassword(length:number) -> a kapott paraméter hosszúságú véletlen jelszóval(string) tér vissza.

„**Óraállások**” menüpont

Az óraállások rögzítése és kilistázása

**meters.component.ts**

fő változók:

meterusers:any[] = []; -> felhasználók és óraállásaik tömbje

fő metódusok:

initializeData() -> összes adat lekérése, sorba rendezése, nincs visszatérési értéke, a változókat tölti fel értékkel

openPreviousData(content:any,userId:string,userName:string) -> a lakó előző óraállásainak a modalját nyitja meg és ad át adatokat (sortedMeterData és userName).

sortMetersData(data:any[]) -> az előző állások sorba rendezése dátum alapján, visszatér a rendezett tömbbel.

getHungarianMonthName(month:number) -> az adott hónapszám magyar nevével tér vissza

saveMeter(userId:string) -> elmenti ill. módosítja a lakó mérőóra adatait, a service *saveMeter* metódusának küldi az *userId* és *meterData* változókat.

„**Beállítások**” menüpont

Az óradiktálás kezdetét, végét, vízórák számát, valamint a költségadatokat lehet beállítani. Itt rögzíthetjük a társasház adatait is.

**settings.component.ts**

fő változók:

measureRecStart:any; -> diktálás kezdete

  measureRecEnd:any; -> diktálás vége

  meterNumber: number = 2 -> vízórák száma /alapesetben 2/

  commonCost:any; -> közös költség

  heatingBase:any; -> fűtés alapdíj

  heatingMulti:any; -> fűtés szorzó

  waterHeating:any; -> vízfelmelegítési díj

  subdeposit:any; -> vízóra nélküli lakás albetéti díja

fő metódusok:

initBaseDatas() -> lekéri az adatokat és elmenti a fenti változókba

saveRecDate() -> a diktálási időszak mentése, validálást végez, ha rendben minden akkor elküldi az adatokat a *service*-nek, ellenkező esetben figyelmeztet a hibás adatokra.

saveMeterNumber() -> elküldi a *service*-nek a vízórák számát, nem validál mert csak két értéket kaphat a HTML form radio gombjaitól.

saveCostDatas() -> a költségadatokat küldi a servicenek, müködése megegyezik a saveRecDate metódussal

A HTML-ben ez a kód rendereli le a társasház adatait:

<div class="mt-3" \*ngFor="let data of condoDatas">

            <form #dynamicForm="ngForm">

                <div class="row row-flex" data-id="{{data.id}}">

                  <div class="col-md-5 mb-2">

                    <input type="text"

                           class="form-control"

                           id="title"

                           name="title"

                           [(ngModel)]="data.title"

                           required [attr.readonly]="data.editable ? null : 'true'"

                           placeholder="Cimke:">

                  </div>

                  <div class="col-md-5 mb-2">

                    <input type="text"

                           class="form-control"

                           id="data"

                           name="data"

                           [(ngModel)]="data.data" required

                           placeholder="Adat:">

                  </div>

                  <div class="col-md-2">

                    <div class="editor-buttons">

                      <a (click)="saveDataRow(data.id, data.title, data.data)" class="buttons green">

                        Mentés

                      </a>

                    </div>

                    </div>

                </div>

            </form>

          </div>

A következő metódusok ezt manipulálják:

addNewRow() -> ha az admin új adatot rögzít, akkor hozzáad két üres input mezőt a beviteli formhoz. {id: newId, title:’’,data:’’, editable:true}

saveDataRow(id:any,title:any,data:any) -> vizsgálja ha nincs *title* érték és van *id* akkor törli az adott sort, ha van *id* akkor módosít, ha nincs *id* de van *title* érték azaz a felhasználó új adatot rögzít akkor új adatként ment.

**settings.service.ts**

fő metódusok:

getSettingDatas(docName:string):Promise<any> és

saveSettingData(docName:string,value:any):Promise<any>

-> a nevüknek megfelelően lekér ill. ment adatokat a firestoreba.

createCollection() -> vizsgálja hogy létezik-e a *condodatas* kollekció, ha nem akkor létrehozza és elment két dokumentumot ’Társasház neve:’ és ’Társasház email címe’ értékekkel. Erre azért van szükség mert ezek fix adatok, minden társasháznak van és így előre kerülnek a listában.

{title: ’Társasház neve’,data:’’,editable:false,timestamp:timestamp}

„**Fájlok**” menüpont

Fájlok tölthetőek fel a firebase *storage* buildjébe.

**uploadfile.component.ts**

fő változók:

selectedFile:File;

description:string = ’’; -> a fájlhoz tartozó leírás

uploadedFiles:any; -> a feltöltött fájlok adatai

downloadUrl: Observable<string>;

uploadProgress: number = 0; -> a feltöltés indikátorának kezdő értéke

fő metódusok:

uploadFile() -> a formból kapott adatokat összegzi és továbbítja a *service* felé.

cancel() -> ha van kiválasztva fájl de mégsem akarja feltölteni az admin akkor törli a form tartalmát. A *selectedFile* és a *description* értékét null-ra állítja.

**uploadfile.service.ts**

változók:

selectedFile: File; -> a kiválasztott fájl tárolása

uploadProgress: Observable<number>; -> a feltöltés százalékos értéke

downloadURL: Observable<string>; -> a fájl url-je

fő metódusok:

uploadFile(selectedFile:File, description:string) -> a kapott adatokkal együtt feltölti a *storage* ’uploads’ mappájába a fájlt.

Visszatér: uploadTask {filePath,selectedFile}

saveFileData(downloadUrl:string, filePath:string, description:string) -> a firestore ’files’ kollekciójába létrehoz egy új dokumentumot a fájl adataival.

getUploadedFiles():Observable<any> -> az előző metódus által elmentett fájlok adatait kéri le és rendezi sorba időrendben.

deleteFile(fileId:string, filePath:string) -> törli a *storage*-ből és a firestore kollekcióból is a fájlt és adatait.

**Shared module**

Itt vannak azok a komponensek css fájlok, amit megosztva használ az admin és az user felület. Használatával elkerülhető a redundancia és ha módosítani kell a kódon akkor elég egy helyen megtenni.

**login.component.ts**

fő változók:

getEmail=””,

getPassword=”” -> a HTML formból kapott értékek tárolásásra.

fő metódus:

login() -> az authService részére küld adatokat. (this.getEmail, this.getPassword)

**passwordreset.component.ts**

fő változó:

email:any -> a jelszómódosítás linkje erre az email címre fog menni.

fő metódus:

resetPassword(email:String) -> szintén az authService részére továbbít adatot.

**passwordreset.service.ts**

fő metódus:

resetPassword(email:string) -> a firestore *sendPasswordResetEmail* szolgáltatásának átadva a kapott email címet, linket küld a kapott email címre.

**auth.service.ts**

fő változók:

user:any,

userId:any;

fő metódusok:

authSateChange() -> figyeli ha változik a bejelenkező személye akkor elmenti az adatokat hogy más komponensek is hozzáférjenek:

 localStorage.setItem('userEmail', user.email);

 localStorage.setItem('userId', user.uid);

login(email:string, password:string) -> a kapott adatokkal megpróbál bejelentkezni a firestore authentication-be, ha sikeres akkor a jogosultságnak megfelelő oldalra irányít. /admin vagy user/

getUserData(uid:string) -> lekéri az aktuális felhasználó adatait az adatbázisból.

visszatér: data()

redirectByRole(isAdmin:any) -> ez valósítja meg a jogosultság szerinti routingot.

    isAdmin ? this.redirectToAdminPage() : this.redirectToUserPage();

saveLoginDate(userId:any) -> a bejelentkezés időpontját menti az *users* kollekció *userId* dokumentumának *loginHistory* kollekciójába.

**shared.service.ts**

fő változó:

userId:string;

fő metódusok:

getMeterDates(userId:string) -> elmenti egy tömbbe, hogy a felhasználó mely hónapokban adta le az óraállásait. Visszatér: dataArray

saveMeter(userId:string, meterData:any) -> a leadott óraállásokat menti a felhasználó *meterdatas* kolelkciójába.

getMetersData(doc:string):Promise<any> -> a megadott dokumentum alapján lekéri annak a *value* értékét. Visszatér: snapshot.data();

getCondoName() -> a társasház nevét kéri le az adatbázisból.

Visszatér: data

getCondoDatas():Observable<any[]> -> lekéri a társasház adatait /email cím, gondnok …/ Visszatér: {id, …data};

**User module**

Ezen a felületen tudják a lakók elérni a szolgáltatásokat. Itt nem kerültek alkalmazásra hagyományos menüpontok, hanem lenyíló elemek vannak. Mivel az egyes elemek tartalma nem foglal sok helyet célszerűnek tűnt inkább ezt a megoldást választani. A *home* komponens gyerek elemei lettek az egyes lenyíló részek.

**home.components.ts**

változók:

bulletinBoardData: any[] | undefined;

  metersData: any;

  currentDay: any;

  currentYearAndMonth: any;

  meterDates: any[] | undefined;

  userId = localStorage.getItem('userId') || '';

  userEmail = localStorage.getItem('userEmail') || '';

  isDictate = false;

  enableDictate: any;

  startDictate: any;

  endDictate: boolean = false;

  condoName: any;

  lastLogin: any;

  newBulletin = 0;

  uploadedFiles: any;

  newFiles = 0;

fő metódusok:

checkEnableDictate() -> megvizsgálja hogy a diktálás aktuális-e.

getLoginDates() -> a *homeservice*getLoginDates metódusát meghívva a lastLogin változóba menti az utolsó bejelentkezés időpontját.

getBbData() -> lekéri a faliújság tartalmát és menti a bulletinBoardData változóba.

getFiles() -> a fájlok listáját menti az uploadedFiles változóba.

isFileNewer(timestamp:any) ->

return this.lastLogin <= timestamp;

<div \*ngIf="isFileNewer(file.timestamp)">

          <span class="red">Új fájl!</span>

        </div>

Ez az \*ngIf hívja meg minden egyes elemén a fájloknak. Amennyiben a visszatérési érték *true*, akkor a fájlt új fájlként jelöli.

isEntryNew() -> hasonlóan működik mint az előző, csak ez a faliújság bejegyzéseit vizsgálja.

**home.service.ts**

változók:

meterData:any;

userId:string;

metódus:

getLoginDates(userId:any):Promise<any> -> lekéri a felhasználó bejelentkezési időpontjait, sorbarendezi csökkenő sorrendbe és az utolsó előttit elmenti egy változóba.

„**Adatok**” elem

**condodatas.component.ts**

változó:

condoDatas:any[] = [];

metódus:

getCondoDatas():Promise<void> -> lekéri a társasház adatait és elmenti egy tömbbe, amit a HTML feldolgoz.

„**Költségek**” elem

**costs.component.ts**

változók:

meterDatas:any={}; -> a mérőórákkal kapcsolatos adatokat tárolja.

datas: any = []; -> a HTML számára tárolja a megjelenítési adatokat.

metódusok:

initMeterDatas():Promise<void> -> feltölti a meterDatas tömböt, az adatbázisból lekért értékekkel.

createArray() -> a meterDatas tömbből kinyert adatokkal létrehozza azt a tömböt amin a HTML ciklusa végigmegy és megjelenít.

„**Diktálás**” elem

**savemeters.component.ts**

változók:

userId = localStorage.getItem('userId') || '';

  metersData: any;

  currentDay: any;

  currentYearAndMonth: any;

  startDictate: any;

  endDictate: any;

  meterNum: any;

  fieldNames: any[] | undefined;

  isDictate = false;

  enableDictate: any;

  meterDates: any[] | undefined;

  cold1: any;

  hot1: any;

  cold2: any;

  hot2: any;

  heat: any;

  errorMessage: any;

  message: any;

metódusok:

initMetersData() -> elmenti a változókba a diktálási időpont kezdetét, végét és a mérőórák számát.

save() -> elmenti az adatbázisba a leadott óraállásokat, a *message* változóba üzenetet ír ami megjelenik a lakónak.

checkInputValues() -> ellenőrzi hogy a beviteli mezőknek van-e értéke.

Visszatér: boolean

„**Fájlok**” elem

**files.component.ts**

változók:

uploadedFiles: any;

  lastLogin: any;

  userId = localStorage.getItem('userId') || '';

metódus:

getLoginDates() -> az utolsó bejelentkezés lekérése.

„**Profilom**” elem

A lakó saját adatait jeleníti meg és itt változtathatja meg a jelszavát.

**myprofile.component.ts**

változók:

userId -> a localStorage.getItem segítségével kap értéket.

userData: any = {}; -> a lakó adatait tartalmazza.

datas: any = [];

message = false; -> a lakó számára megjeleníthető üzenetet tartalmazza.

metódusok:

getUserData() -> a service segítségével feltölti az userData tömböt.

resetPassword() -> a firebase resetpassword szolgáltatását hívja meg.

**További fejlesztési lehetőségek/tervek**

1. szavazás komponens létrehozása. Az admin szavazást indíthat különböző a társasházat érintő témákban.
2. ha a lakó leadta az óraállását, akkor a program kiszámolja a költségadatok alapján a havi rezsit és emailben el is küldi neki.
3. közös költséget ne csak négyzetméter alapján lehessen számolni, vannak olyan társasházak is ahol csak 3 féle lakás van, ott fix a közös költség lakástípusonként.
4. szerveroldalon létrehozni egy CRON-t ami a diktálás kezdetekor és a vége előtt x nappal értesítést küld a lakóknak.
5. log kollekciót létrehozni az adatbázisban, ami segítségével az esetleges hibákat nyomon lehetne követni.
6. email küldése a lakóknak abban az esetben ha új hír vagy fájl kerül feltöltésre

**Felhasználói dokumentáció**

A mycondo.hu egy weboldal, ami többféle céllal készült.

-a társasház lakónyilvántartást tud vele vezetni

-híreket információkat oszt meg a lakókkal, online felületen

-fájlokat, például meghatalmazás nyomtatványt, jegyzőkönyveket, dokumentációkat stb. tölt fel az oldalra, amit a lakók bármikor letölthetnek

-a lakók a mérőórák adatait leadhatják az oldalon, ezáltal nem kell cetliken bedobálni, vagy a faliújságra kiírni

Az oldal használatához elég egy számítógép, tablet vagy okostelefon.

**Használat (lakók részére):**

**Bejelentkezés**

Amikor a társasház adminisztrátora regisztrálja Önt a mycondo.hu-n, automatikusan kap egy levelet az e-mail fiókjába. A levél megerősíti, hogy sikeresen regisztrálva lett a társasház rendszerében, egyúttal kap egy nyolc karakterből álló jelszót, amivel a bejelentkezési oldalon be tud lépi.

A belépéshez írja be az email címét és a kapott jelszót és nyomja meg a ’Bejelentkezés’ gombot.

*Fontos! Az adatai védelmében az első bejelentkezéskor változtassa meg a jelszavát a ’Profilom’ menüpont alatt!*

Abban az esetben, ha véletlen törölte a regisztrációs emailt vagy elfelejtette a jelszavát, akkor a ’Bejelentkezés’ gomb alatt található ’Elfelejtette a jelszavát’ feliratra kattintva egy másik oldal jelenik meg, ahova beírva az email címét és a ’Küldés’ gomra kattintva egy hivatkozást küld a rendszer az email címére.

A kapott hivatkozásra kattintva megnyílik egy oldal, ahol adja meg az új jelszavát kétszer. A ’Mentés’ gomb lenyomása után már be tud jelentkezni az oldalra.

**Főoldal**

Sikeres bejelentkezés után a honlap kezdőoldala jelenik meg. A felső részén olvasható a társasház neve, a bejelentkezett személy email címe és a kilépés gomb.

A kijelentkezéshez nyomja meg ezt a gombot és amikor a rendszer rákérdez hogy ’Biztosan kijelentkezik’ akkor nyomja meg az igen gombot.

Alatta jelenik meg az ’Óraállások leadhatóak’ felirat amennyiben még nem adat le és a dátum a diktálás kezdete és vége közé esik.

Szokatlan lehet, hogy a főoldalon nincsenek hagyományos navigációs gombok. A feliratokra kattintva lenyíló formában jelenik meg az adott elem tartalma.

Hat felirat, illetve ikon található az oldalon: Hírek, Adatok, Költségek, Diktálás, Fájlok, Profilom

A Hírek és a Fájlok sorban látható, hogy a legutolsó bejelentkezése óta hány frissítés történt. pl. 5 új. Ha nem történt változás akkor a ’Nincs új’ felirat látható.

A Diktálás sorban a diktálási időszak kezdete és vége jelenik meg, ha Ön már leadta akkor a ’Leadva’ felirat látható.

**Hírek**

Itt olvashatóak a társasház új hírei, információi és felhívásai.

**Adatok**

A társasház adatai találhatóak ebben a részben. pl.: elérhetőségek, telefonszámok, nevek

**Költségek**

A ház szolgáltatásainak költségeit tartalmazza. pl.: közös költség, fűtés stb.

**Diktálás**

A mérőórák (víz, fűtés) aktuális havi állásait adhatja le. Csak akkor aktív, ha a két diktálási dátum között van és még nem adta le az adatait.

**Fájlok**

A feltöltött fájlok listája itt található. A fájl neve után levő ’Letöltés’ gombot lenyomva tölthető le.

**Profilom**

Az Ön adatait tartalmazó mező. Itt változtathatja meg a jelszavát a ’Jelszó módosítása’ gombra kattintva.

**Használat (admin részére)**

Bejelentkezés után a főoldalra kerül. Itt a ’Menüpontok használata’ felirat alatt található gombok, tájékoztatást nyújtanak az egyes menüpontok szerepéről.

**Faliújság**

Itt hozhatja létre azokat a tartalmakat, amiket a lakókkal meg szeretne osztani. A felső szerkesztő mező úgy működik, mint egy szövegszerkesztő. A lakó oldalon úgy fogják látni a tartalmat ahogy azt megszerkesztette.

Alatta a korábban feltöltött tartalmak vannak. A ’módosít’ gomra kattintva az adott tartalom a szerkesztő mezőbe kerül, ahol módosíthatja azt. Itt a ’Mégse’ gombot megnyomva visszavonhatja a módosítást.

**Lakók**

A rögzített lakók listája jelenik meg. A névre kattintva több információ jelenik meg róluk.

A kereső mező segíthet egy adott lakót megtalálni.

A ’Lakó hozzáadása’ gombra kattintva egy felugró ablak jelenik meg, ahol az adatok kitöltése után a ’Mentés’ gombot kell megnyomni a rögzítéshez. A név és az email cím kötelező mezők.

A ’Módosít’ gomb szintén egy felugró ablakot hoz létre, ahol módosíthatja az adatokat. A ’Mégse’ gomb visszavonja ezeket.

**Óraállások**

Az aktuális hónapban leadott óraállások listája. Ha a mezőben 0 szerepel az azt jelenti, hogy még nem adták le. A leadott óraállást lehet módosítani hiba esetén.

Az ’Előző’ gomb kilistázza az adott lakó utolsó hat havi állásait.

**Beállítások**

A diktálási időszak, vízórák száma és a költségek beállítása: írja be az értéket és nyomja meg a mentés gombot.

A társasház adatai:

Az első mező tartalmazza az információ típusát. pl.: Közös lépviselő, Th telefonszáma stb.

A második mező az információ. pl.: Kovács Béla, +36 20 123 45 67

Ha törölni akar egy mezőt akkor törölje ki a ’címke’ mezőt vagy mind a kettőt és nyomja meg a mentés gombot.

**Fájlok**

Fájl feltöltéséhez kattintson a ’Fájl kiválasztása’ gombra. A leírás mezőben megjegyzést fűzhet a fájlhoz. pl. Meghatalmazás, közgyűlési jegyzőkönyv stb.

Törléshez nyomja meg a fájl mellett levő ’Törlés’ gombot.

**Tesztelés**

A tesztelés a három legnépszerűbb böngésző a Google Chrome, Mozilla Firefox és a Microsoft Edge felületén zajlott. A reszponzivitást a böngésző fejlesztői környezetében különböző mobiltelefonok szimulációjával, illetve a böngészőablak átméretezésével lett ellenőrizve. Amikor a projekt felkerült a webtárhelyre akkor a tesztelés mobiltelefonon is megtörtént.

**Egységteszt**

A fejlesztés során minden metódus kimeneti értéke először a fejlesztői konzolra lett kiíratva, amennyiben a kapott visszatérési értékek megfelelőek voltak és nem jelzett hibát a konzol akkor az adott metódus használatba lett véve.

A kódban még sok helyen megtalálhatóak a konzolos kiírató parancsok, mivel a projekt fejlesztése még nem zárult le teljesen. A későbbi tesztek futtatásához hasznos lesz.

A kódban található *service* fájlok, amik főleg az adatbázis kapcsolatokért felelősek szintén fejlesztői konzolon voltak tesztelve és csak helyes működés esetén írhattak az adatbázisba.

**Integrációs teszt**

A teszt folyamán vizsgálva lett a különböző komponensek közötti helyes adatátadás és interakció.

Az admin felület tesztelése kiegészült olyan tesztel is amely során hibás adatok lettek beírva a beviteli mezőkbe. Ez a teszt főleg a lakók menüpont alatt volt fontos, mert itt van csak lehetőség hibás adatot megadni. A *lakó neve* opció kötelező mező, amit a kódban a

      name: ['', Validators.required],

sor valósít meg.

Az email mező ellenőrzése szintén szükséges a megfelelő formátum miatt, amit a

      email: ['', [Validators.required, Validators.email]],

sor ellenőriz.

Amennyiben a bevitt értékek nem megfelelőek akkor az admin egy-egy figyelmeztető üzenetet kap és a mentési folyamat megszakad.

Az alkalmazás első indulásakor még több kollekció nincs létrehozva az adatbázisban, ezért fontos, hogy a kód vizsgálja ezek meglétét. Amennyiben még nincsenek létrehozva akkor létrehozza azokat.

A teszt során igénybe lett véve harmadik fél is, aki „szimulálta” az átlagos felhasználót. A visszajelzéseket követve a kód felhasználóbarátabb lett.

**Összefoglalás**

A fejlesztés célja véleményünk szerint megvalósult. Az előre kitűzött célok megvalósításra kerültek, kivéve azok, amik opcionálisnak lettek a projekt kezdetén jelölve.

Került bele viszont olyan opció, amiről a fejlesztés közben derült ki, hogy hasznos funkció lenne a weboldal működéséhez.

Mi a véleményünk az elkészült munkáról?

Az, hogy most rögtön újra kellene írni 😊.

Azért jön ez a gondolat mert a fejlesztés során tanultuk a legtöbbet a programozás rejtelmeiből. Ahogy haladtunk előre a fejlesztésben a tudásunk és a tapasztalatunk is folyamatosan gyarapodott. Így visszanézve a projekt elejére már nem tűnik olyan tökéletesnek, mint akkor. Természetesen működik minden funkciója, de a kódon még lehetne csiszolni.

Mit tanultunk a fejlesztés során?

Kihívás volt megoldani a közös fejlesztést. Ebből sokat tanultunk. Az önálló fejlesztés, oktatói segítség nélkül szintén fontos tapasztalat volt.

Az elkészült szoftvert szeretnénk továbbfejleszteni, hozzáadni azokat a funkciókat amire nem volt idő/energia a fejlesztés során. Optimalizálni a kódot és a működését.

A végső cél, hogy használatba kerüljön és segítséget nyújtson a felhasználóknak.