



EÖTVÖS LORÁND TUDOMÁNYEGYETEM

Informatikai Kar

Média- és Oktatásinformatikai Tanszék

Optimalizált érdekképviselési rendszerek webes felületeken a reaktív paradigmákat követve

Témavezető

Dr. Abonyi-Tóth Andor

Okleveles informatika tanár

Szerző

Deák Dalma Annamária

Programtervező informatikus

Budapest, 2021.

„Ki mint élte tavaszát, úgy fizet annak az ősz.”

- Árkossy Károly

EREDETISÉGI NYILATKOZAT

Alulírott **Deák Dalma Annamária** (Neptun-kód: IFTH4E) ezennel kijelentem és aláírással megerősítem, hogy az ELTE Programtervező informatikus alapszakon írt jelen szakdolgozatom saját szellemi termékem, melyet korábban más szakon még nem nyújtottam be szakdolgozatként és amelybe mások munkáját (könyv, tanulmány, kézirat, internetes forrás, személyes közlés stb.) idézőjel és pontos hivatkozások nélkül nem építettem be.

Budapest, 2021. november 30.



Deák Dalma Annamária

TARTALOMJEGYZÉK

Eredetiségi nyilatkozat.....	3
1. Bevezetés.....	7
1.1 Motiváció.....	7
1.2 Használt technológiák.....	7
1.3 Látványtervek, elképzelések.....	9
2. Felhasználói dokumentáció	13
2.1. Rendszerigények	13
2.2 A program telepítése, használata.....	13
2.2.1 A program kitelepítése AWS Elastic Beanstalk szolgáltatásra.....	14
2.2.2 A program telepítése saját eszközre	16
2.3 Az oldal felépítése bejelentkezés nélkül	18
2.3.1 A Híroldal	18
2.3.2 Az Ülések oldal	19
2.3.3 A Szervezet oldal	20
2.3.4 Az Átláthatóság oldal	21
2.3.5 Az Ösztöndíjak oldal.....	22
2.3.6 A Szolgáltatások oldal	23
2.3.7. Egyéb látogatói funkcionalitások.....	24
2.4 Az oldal felépítése bejelentkezéssel	24
2.4.1 A Profil oldal	25
2.4.2 Az Új bejegyzés oldal	26
2.4.3 A Saját beszámolók oldal.....	27
2.4.4 A Beszámolók összegzése oldal.....	27
2.4.5 A Határozatok tára oldal.....	28

2.4.6 A Sorompó adminisztráció oldal	28
2.4.7 A Belépőkártya adminisztráció oldal	29
2.4.8 Az Admin panel oldal	30
2.4.9. Egyéb felhasználói funkcionalitások	31
2.5 Hozzáférés vezérlés, jogosultsági körök	31
3. Fejlesztői dokumentáció	33
3.1 A feladat leírása	33
3.2 A felhasználói esetek	33
3.3 Alkalmazott technológiák, használt programcsomagok	35
3.3.1 Optimalizált applikáció	35
3.3.2 Typescript	36
3.3.3. Single Page Application	36
3.3.4 RESTful API	37
3.3.5 Angular Forms	37
3.3.6 Angular Animations	37
3.3.7 Angular Router	37
3.3.8 NgImageSlider	37
3.3.9 NgxBootstrap	38
3.3.10 NgxPagination	38
3.3.11. Http	38
3.3.12 JsonWebToken	38
3.3.13. Bcrypt	38
3.3.14. Multer	38
3.4 Az adatbázis felépítése	38
3.5 A backend felépítése	42
3.5.1 Routes	42
3.5.2 Az app.js	46

3.5.3 A server.js.....	47
3.5.4 Képek és fájlok tárolása	48
3.5.5 Az angular mappa és a package.json	50
3.6 A frontend felépítése.....	50
3.6.1 Komponens diagram.....	51
3.6.2 Az új bejegyzéseket létrehozó komponensek	52
3.6.3 Az bejegyzéseket tartalmazó komponensek	56
3.6.4 A bejelentkezés.....	58
3.6.5 Egyedi függvények.....	61
4. Tesztelés	62
4.1 Tesztelési jegyzőkönyv.....	62
5. Fejlesztési lehetőségek.....	68
5.1 Új funkcionalitások	68
5.1.1 Levelezéssel való összekötés	68
5.1.2 Beszámolók összegzése, azok generálása dokumentumba	68
5.2 Meglévő funkcionalitások fejlesztése.....	68
5.2.1 A Híroldal képnézegetője	68
6. Összefoglalás	69
7. Köszönetnyilvánítás.....	70
8. Ábrajegyzék.....	71

1. BEVEZETÉS

1.1 Motiváció

Egyetemi tanulmányaim során lehetőséget kaptam a hallgatói érdekképviselésben való részvételre az ELTE IK Hallgatói Önkormányzatában, és az itt eltöltött időm során – kifejezetten most a járványhelyzet idején is – be kellett látnom, hogy a belső szervezeti, valamint a külső, hallgatóság irányába nyújtott kommunikáció elengedhetetlen egy önkormányzat kiegyensúlyozott működéséhez, melynek egyik legcélravezetőbb formája a közösségi médiák mellett a HÖK honlapja.

Mivel képzésem során az ízlésemhez és képességeimhez a webprogramozás állt a legközelebb, így személyes célomnak tűztem ki, hogy önkormányzatunk honlapjának nem csak egy új, igényesebb külsőt adjak, hanem fejleszthessek a meglévő és tervben lévő funkcionalitásokon, igényeken. Ennek okai a jelenlegi honlapunk hiányosságai, a kihasználatlan, szükségtelen technológiák és a már nem működő szolgáltatások, melyek mihamarabbi kijavítását szükségesnek érzek.

Nemcsak egyetemi tanulmányaim, hanem szakmai gyakorlatom során is megtapasztalhattam a webfejlesztés sajátosságait. Frontend fejlesztő munkám során elsődlegesen Vue.js¹ keretrendszerben dolgozhattam, mely mellett többször is lehetőségem volt kipróbálni magam Angular²-ben is. Bár egészen 2021 nyaráig nem volt szakmai tapasztalatom backend alkalmazások elkészítésében, nem gondoltam azt, hogy ez megakadályozna egy ilyen volumenű applikáció elkészítésében, hiszen biztos vagyok benne, hogy ez a fejlesztés nagymértékben előre fogja lendíteni a Hallgatói Önkormányzat mindennapi működését.

1.2 Használt technológiák

Jelenlegi honlapunk egy PHP³ alapokon futó oldal, mely beágyazott CSS és Javascript fájlokkal dolgozik, és HTML kimenetet ad vissza. Bár szerveroldalon sok esetben dinamikusan funkcionál, több adathalmaz is statikusan került elmentésre, mely megnehezíti ezek módosítását. A honlap nem kezeli AJAX lekérdezéseket, hanem pusztán adatbázis

¹ <https://vuejs.org/>

² <https://angular.io/>

³ <https://www.php.net/>

lekérdezésekből, azon belül is az ELTE Informatikai Igazgatóság által kezelt Caesar PDO⁴ segítségével dolgozik, emellett egyes funkciókat – ilyen például az autentikáció és a hozzáférések vezérlése – LDAP⁵ protokoll segítségével valósít meg.

Mivel az önkormányzati honlapot első sorban hallgatók kezelik és fejlesztik, mindenképpen fontosnak tartottam egy egyszerűbb, korszerű technológia bevezetését, melyet már egy harmadik féléves hallgató is könnyen használni tud. Az ELTE IK képzéseinek felépítésének köszönhetően képzéstől és szakiránytól függetlenül minden hallgató lehetőséget kap a webprogramozás alapú tárgyak elvégzésére, melyeken olyan keretrendszereket, könyvtárakat ismerhetnek meg, amelyek használatát én is szorgalmaztam szakdolgozatom írása közben.

Frontend oldalról az applikáció Angular 12 keretrendszerben készült. Számos előnye közé tartozik az, hogy nem csak könnyen kezelhető és megtanulható a hallgatók által, hanem lehetőséget biztosít a reaktív felhasználói élményhez is egyaránt. Utóbbi értelmében a felhasználó mindig azonnal látja a honlapon folytatott tevékenységeinek eredményét, emellett mindig tisztában van az adatainak állapotával. Az Angular keretrendszerben a további fejlesztések is könnyen implementálhatóak a komponens alapú felépítésének köszönhetően, és az esetleges verzióváltások is egyszerűen abszolválhatóak. Mindezek segítségével az IK HÖK honlapjának elkészítéséhez könnyedén készíthetünk Single Page Application-t (SPA-t), melynek működését és fontosságát dolgozatomban a későbbiekben fogok kifejteni.

Az applikáció backendjének elkészítéséhez a Node.js⁶ szoftverrendszert és az Express.js⁷ keretrendszert választottam. Mivel a Node.js az Angular-hez hasonlóan Javascript alapú, illetve magának az Angular keretrendszernek is szüksége van a Node.js egyes funkcionalitásaira, ezért kézenfekvőnek tűnt ezen választásom. Utóbbinak oka, hogy az Angular app elkészítése során Typescript⁸-et használok, mely a Javascript egy szigorúan szintaktikus felülhalmaza, és ennek Javascript-re való fordítását a Node.js végzi annak érdekében, hogy a böngésző az általunk írt kódot értelmezni tudja. Emellett magának az Angular applikációnknak is szüksége van az összecsomagolására és optimalizálására, ezzel elérve, hogy minimalizálhassuk kódunk méretét – ezt szintén a Node.js szoftverrendszer végzi el.

⁴ <https://www.php.net/manual/en/book.pdo.php>

⁵ <https://ldap.com/>

⁶ <https://nodejs.org/>

⁷ <https://expressjs.com/>

⁸ <https://www.typescriptlang.org/>

Az Express.js használatával célom, hogy a backend-en menedzselhessük a kéréseket és az általunk létrehozott útvonalakat, emellett a HTTP⁹ modul használatát és a REST API¹⁰ szoftverarchitektúra létrehozását nagymértékben leegyszerűsíthessük. Bár alapvetően nem szükséges egy full-stack applikáció elkészítéséhez, az egyszerű kezelhetőség és átláthatóság érdekében nagy segítséget nyújt az önkormányzat számára.

Adataink tárolása céljából a MongoDB¹¹ NoSql adatbázist választottam, melyben a megszokott táblák és rekordok helyett kollekciókat és dokumentumokat kezelünk. A MongoDB nem erőlteti az adatok közötti szoros kapcsolatokat vagy adatsémákat, emellett könnyen integrálható Node.js projektekhez is egyaránt. Ahhoz, hogy ezen applikációkhoz könnyedén hozzákapcsolhassuk adatbázisunkat, egy harmadik féltől származó csomagot használtam, mely a Mongoose¹² névre hallgat.

Ezen MEAN Stack architektúra, melynek kialakítását szorgalmaztam szakdolgozatomban, egy dinamikus webalkalmazás építését segíti elő. Nem csak egyszerű kezelhetősége és open-source elérhetősége miatt tetszett meg ez a szoftverköteg, hanem mert a jelenlegi trendeknek megfelelően kifejezetten elterjedt technológiának számít, így az Önkormányzat bármely tagja könnyedén elsajátíthatja a szükséges tudást annak érdekében, hogy a későbbi igényeknek megfelelően fejleszthessék az általam elkészített applikációt.

1.3 Látványtervek, elképzelések

A szoftver elkészítése előtt kiemelten fontosnak tartottam a drótváz- és látványtervek elkészítését. Mivel nem csak a funkcionalításokon szerettem volna fejleszteni, hanem a honlap arculatán is, ezért a jelenlegi neves oldalak dizájnjaiból ihletet merítve alakítottam ki az IK HÖK honlapjának kinézetét.

⁹ <https://nodejs.org/api/http.html>

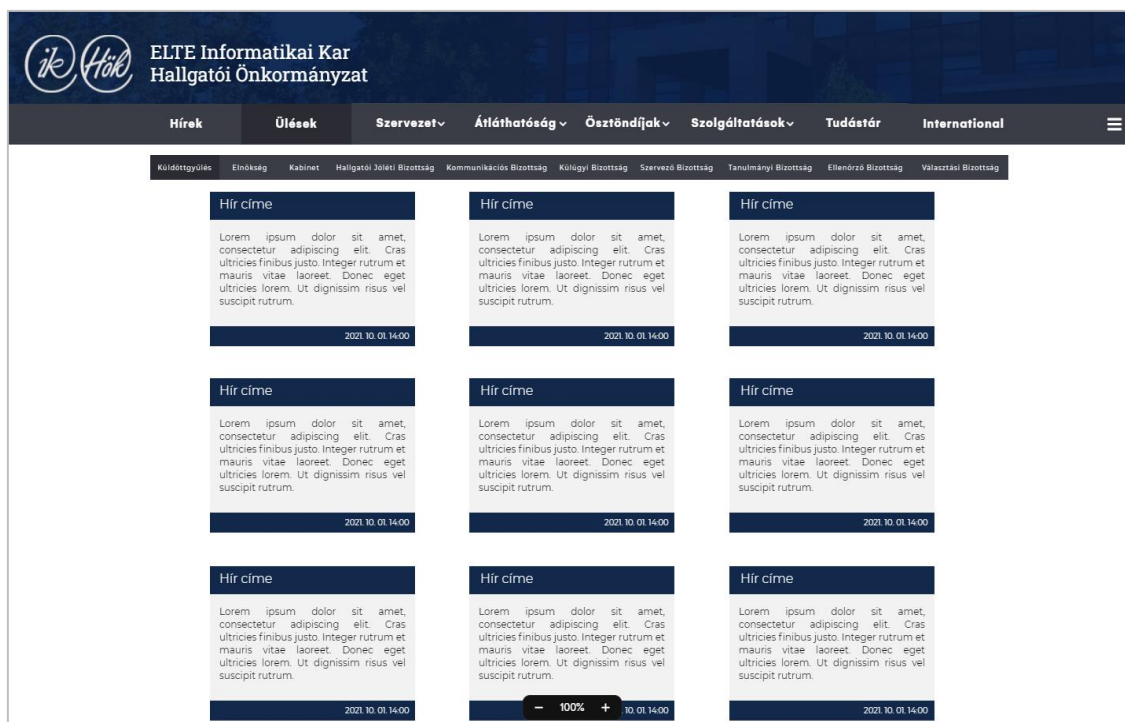
¹⁰ <https://restfulapi.net/>

¹¹ <https://www.mongodb.com/>

¹² <https://mongoosejs.com/>

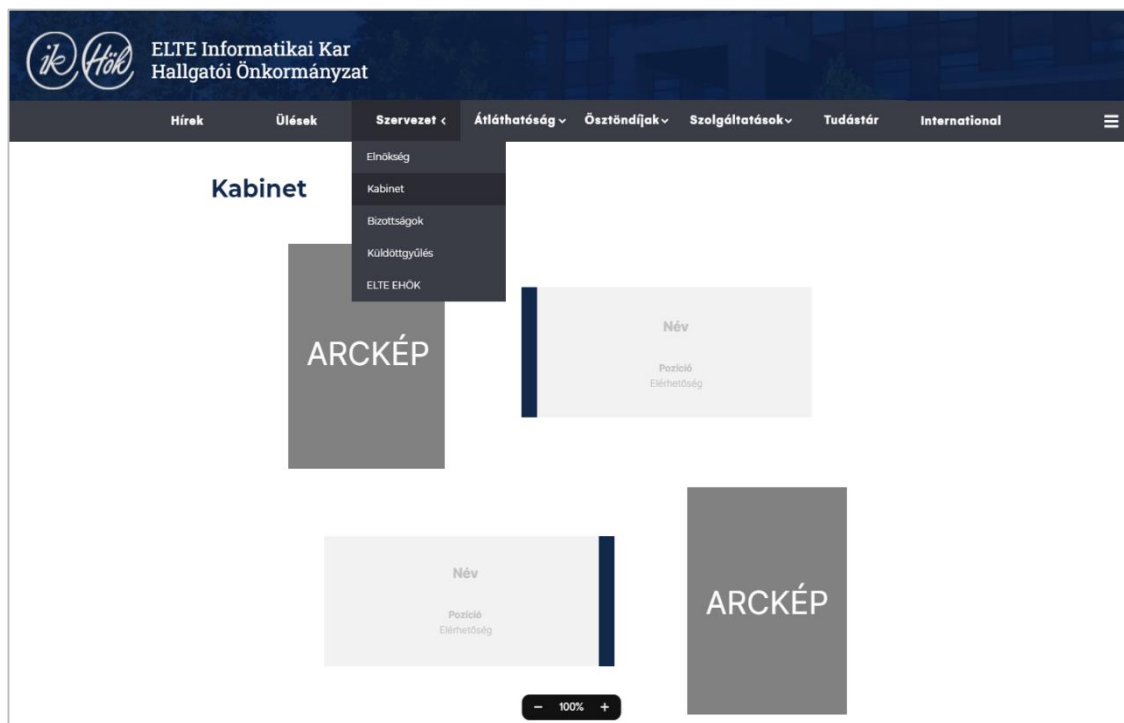


1.3.1. ábra - Hírek főoldal látványterve



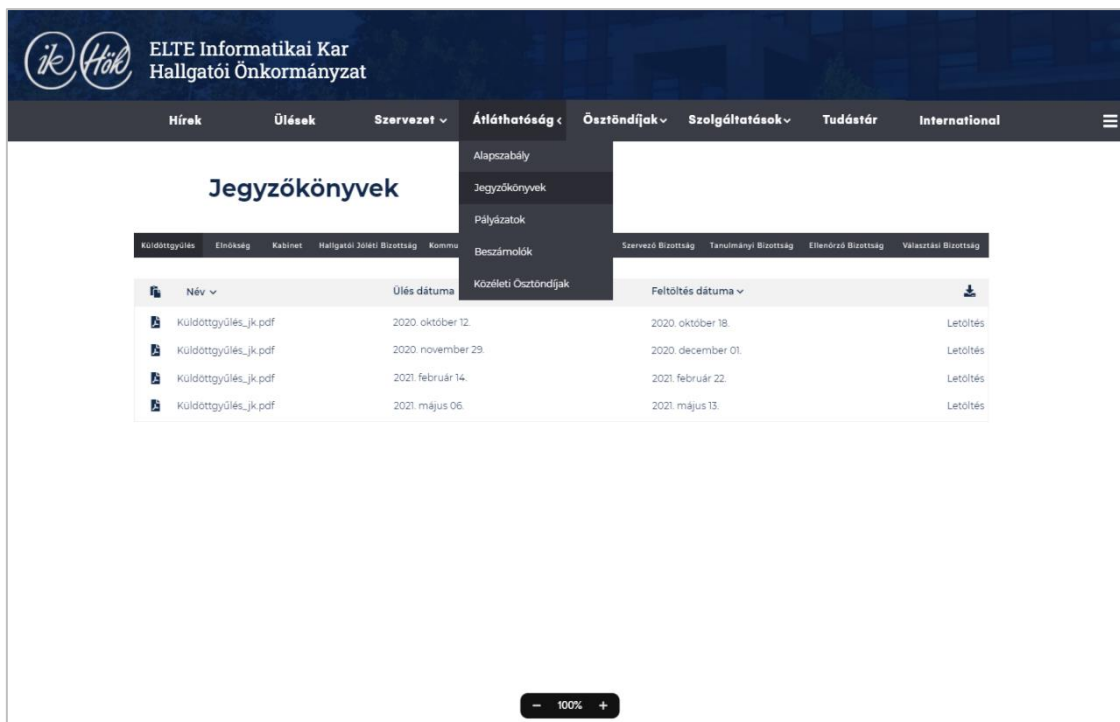
1.3.2. ábra – Ülések oldal látványterve

Az 1.3.1. és az 1.3.2. ábrán látható, hogy előtérbe helyeztem az ELTE IK HÖK arculati kézikönyvének alapelveit a tervezet kialakítása során, ezáltal törekedtem egy letisztult, felhasználóbarát dizájn kialakítására.

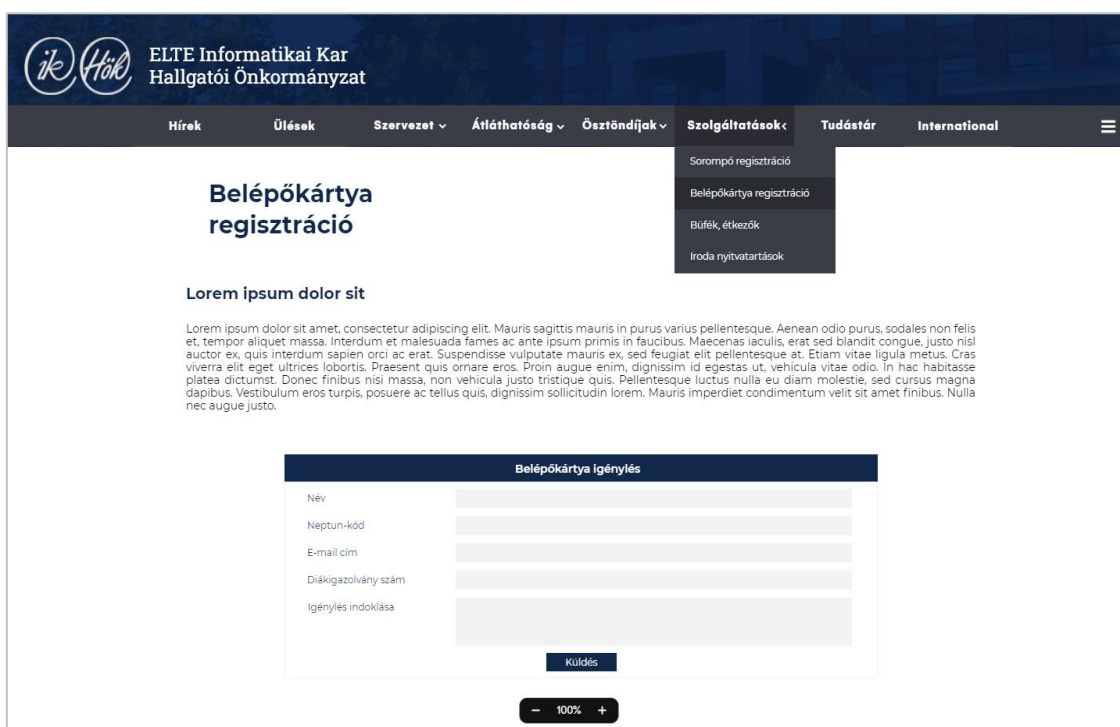


1.3.3. ábra – Szervezet oldal látványterve

A dinamikusan elhelyezett adatok stilizálásakor első sorban fehéres árnyalatokat, valamint az ELTE IK HÖK egyik hivatalos színét, a **#13294B** HEX kódú sötétkékét használtam. A felület könnyen értelmezhető, mellyel az volt a célom, hogy bármelyik oldalról könnyedén eljuthassunk azonnal egy másikra. Éppen ezért a honlap minden egyes aloldalán megtalálható a fejléc és a navigációs sáv is.



1.3.4. ábra – Átláthatóság oldal látványterve



1.3.5. ábra – Szolgáltatások oldal látványterve

2. FELHASZNÁLÓI DOKUMENTÁCIÓ

Ezen applikáció célja, hogy mind karunk hallgatói, mind pedig az ELTE IK Hallgatói Önkormányzatának képviselői a megszokott tartalmak mellett új, hasznos információkhoz juthassanak a honlapon, és az ezekhez kötődő funkcionalitások is hiánytalanul működjenek. Ezen tartalmak közé tartozik a megszokott híroldal, a bizottsági ülések összehívó szövegei, a szervezet felépítése és a tisztségviselők elérhetőségeik, valamint a tudnivalók a HÖK és a Kar által nyújtott szolgáltatásokról.

Az önkormányzati tisztségviselők számára a belső rendszerben a továbbiakban is elérhetőnek kell maradnia a havi beszámolók folyamatos követésének, az üléseken meghozott döntések határozati javaslatainak vezetésének, illetve a szolgáltatások menedzselésének.

2.1. Rendszerigények

A szoftver már kitelepített verziójának eléréséhez az alábbi böngészőkövetelmények szükségesek:

- Chrome: legalább 63-as verzió,
- Firefox: legalább 58-as verzió,
- Safari: legalább 9-es verzió,
- Android: legalább Marshmallow 6.0 verzió.

Amennyiben az applikációt saját eszközünkről (saját számítógépünkön, operációs rendszerünkön) szeretnénk futtatni, úgy a Node.js szoftverrendszerhez tartozó rendszerkövetelmények az alábbiak:

- Windows 10 64-bit, Linux vagy MacOS Catalina,
- 10 GB szabad tárterület,
- 4 GB RAM,
- Az előbb említett böngészők egyike a specifikált verzióknak megfelelően.

2.2 A program telepítése, használata

Az alkalmazás használatának megkezdését kétféleképpen kezdeményezhetjük. Lehetőségünk van egy általunk preferált Cloud Service szolgáltatás igénybevételére és az alkalmazás kitelepítésére, de saját eszközünkön is elindíthatjuk a szoftvert.

2.2.1 A program kitelepítése AWS Elastic Beanstalk szolgáltatásra

Az első lehetőség választása esetén választanunk kell egy megbízható, ingyenesen elérhető Cloud Service szolgáltatást, melynek egyik példája az Amazon Web Services Elastic Beanstalk, de megfelelőek erre a célra a Heroku szolgáltatásai is.

A Beanstalk honlapján történt regisztrációt követően létre kell hoznunk egy új környezetet (Environment), illetve egy applikációt is (Application), melyekre a szoftvert ki fogjuk telepíteni. A telepítés során az integrált megközelítést fogom alkalmazni, melynek értelmében az applikáció frontend és backend oldalát nem két külön app-ként fogom feltölteni, hanem egyként.



Create new application

Application information

Application name

thesis-app-node-angular

Maximum length of 100 characters, not including forward slash (/).

Description

Szakdolgozat kitelepítése Cloud Service-re.

2.2.1. ábra – Új applikáció létrehozásának menete az AWS Elastic Beanstalk szolgáltatásban

A környezet létrehozásakor egyes beállításokat mindenképpen el kell mentenünk annak érdekében, hogy a szoftver feltöltésekor problémamentesen meg tudjon kezdődni az applikáció használata.

Platform gyanánt a Node.js opciót kell választanunk, az ágat, illetve a verziót pedig hagyjuk az automatikusan kiválasztott válaszokon – ez jelen esetünkben a „Node.js 14 running on 64bit Amazon Linux 2”, illetve az 5.4.8-as verzió.

Platform

☒ **Managed platform**
Platforms published and maintained by Amazon Elastic Beanstalk. [Learn more](#)

☐ **Custom platform**
Platforms created and owned by you.

Platform
Node.js ▼

Platform branch
Node.js 14 running on 64bit Amazon Linux 2 ▼

Platform version
5.4.8 (Recommended) ▼

2.2.2. ábra – AWS Elastic Beanstalk platform beállításai

A következő beállítási szükséglet a kód feltöltése. Ehhez a letöltött zip állományban található egy `deploy.zip` nevű összecsomagolt mappa, melyben az elkészített webalkalmazás kitelepített verziója található meg – ennek kibontása nem szükséges, ebben a formában töltsük fel a felhőbe.

Application code

☐ **Sample application**
Get started right away with sample code.

☐ **Existing version**
Application versions that you have uploaded for `thesis-app-node-angular`.
-- Choose a version -- ▼

☒ **Upload your code**
Upload a source bundle from your computer or copy one from Amazon S3.

Version label
Unique name for this version of your application code.
thesis-app-node-angular-source

Source code origin
Maximum size 512 MB

☒ **Local file**

☐ **Public S3 URL**

File name : **deploy.zip**

☒ File successfully uploaded

► Application code tags


2.2.3. ábra – AWS Elastic Beanstalk feltöltése beállítások

Ezen beállítások elmentésével hozzuk létre új környezetünket. A szerver konfigurálása pár perccel fog tartani, azonban amint elkészül, a szolgáltatás oldalán található „Environments” menüpontra kattintva kiválaszthatjuk újonnan létrehozott környezetünket. Amennyiben a feltöltés sikeres volt, az oldal tetején található linken el is érhetjük kitelepített applikációnkat.


Thesisappnodeangular-env
Thesisappnodeangular-env.eba-idxqxrzk.eu-central-1.elasticbeanstalk.com (e-gvnx3gfjeh)
Application name: thesis-app-node-angular

Refresh

Actions ▼

Health

 Ok
 Causes

Running version
 thesis-app-node-angular-source
 Upload and deploy

Platform

 Node.js 14 running on 64bit
 Amazon Linux 2/5.4.8
 Change

Recent events

Show all

< 1 >

Time	Type	Details
2021-11-30 12:11:31 UTC+0100	INFO	Added instance [i-03460082e0a527c4d] to your environment.
2021-11-30 12:11:31 UTC+0100	INFO	Environment health has transitioned from Pending to Ok. Initialization completed 25 seconds ago and took 4 minutes.
2021-11-30 12:11:31 UTC+0100	INFO	Successfully launched environment: Thesisappnodeangular-env
2021-11-30 12:11:31 UTC+0100	INFO	Application available at Thesisappnodeangular-env.eba-idxqxrzk.eu-central-1.elasticbeanstalk.com.
2021-11-30 12:10:59 UTC+0100	INFO	Instance deployment completed successfully.

2.2.4. ábra – AWS Elastic Beanstalk elkészített környezet elérése, adatai

2.2.2 A program telepítése saját eszközre

A szoftver megfelelő használata érdekében elsősorban meg kell bizonyosodnunk arról, hogy saját eszközünkön megtalálható a legfrissebb Node.js verzió, hiszen az npm-et fogjuk használni a szükséges csomagok telepítésére.

A letöltött szakdolgozat mappájának kicsomagolását követően az általunk preferált módon navigáljunk az applikáció mappájába. Ezt megtehetjük parancssor segítségével is, de használhatunk integrált fejlesztői környezetet (IDE-t), vagy kódszerkesztő alkalmazást is, mint a Visual Studio Code.

A lokális szerverünk elindítása előtt installálnunk kell a szükséges csomagokat az applikáció működéséhez. Ehhez használjuk az alábbi parancsot, mely a projektben lévő package-lock.json tartalma alapján installálni fogja a package-eket.

```
npm ci
```

Amint ezt megtörtént, Visual Studio Code használata esetén érdemes az ablak frissítése, mivel az installációt követően egyes csomagok települését nem ismeri fel azonnal – ilyen például a tslib package is, melynek hiányában a Typescript fájlok tartalma hibát fog adni. Ehhez nyissuk meg a program parancs palettáját (Ctrl + Shift + P), és írjuk be az alábbi parancsot:

```
Reload Window
```

A fejlesztői módban való elindításhoz két terminálra lesz szükségünk – az egyiken az frontend Angular applikációt fogjuk futtatni, míg a másikon a backendet. Előbbi elindításához nyissunk egy új terminált, majd az Angular CLI segítségével indítsunk el az applikációt:

```
ng serve
```

Ekkor az alkalmazást az automatikusan generált 4200-as porton, a http://localhost:4200 URL címen el is érhetjük, azonban ez még nem egy teljesértékű szoftver önmagában. A backend elindításához és az adatbázishoz való csatlakozáshoz egy külön terminálba írjuk be az alábbi parancsot:

```
node backend/server.js
```

Ekkor a terminálban láthatjuk, hogy a szerveroldal sikeresen elindításra került, és az adatbázisra való csatlakozás is megtörtént. Előfordulhat, hogy az automatikusan választott port (ami jelen esetünkben a 3000-es) már egy másik alkalmazás által foglalt. Ekkor az alábbi hibaüzenettel fogunk találkozni:

```
> port 3000 is already in use!
```

Emellett, amennyiben olyan portot próbálunk választani, melyhez nem rendelkezünk megfelelő jogosultsági körökkel eszközünkön, akkor pedig az alábbi hibaüzenetet fogjuk terminálunkban látni:

```
> port 3000 requires elevated privileges!
```

A szoftver nem csak fejlesztői, hanem bemutatási (production) módban is elérhető. Ekkor mindösszesen egy terminálra lesz szükségünk az általunk választott IDE-ben vagy parancssorban, melyben az alábbi parancs begépelésével el is indul a szerver:

```
npm run start:server
```

Ekkor a kitelepített alkalmazás a `http://localhost:3000` URL címen elérhetővé is válik, és azonnal használható is.

2.3 Az oldal felépítése bejelentkezés nélkül

A honlap elérését követően szembe találkozunk az alapvető funkcióinkkal – A híroldallal, az ülésekkel, a szervezettel, az átláthatósággal, az ösztöndíjakkal, a szolgáltatásokkal és a tudástárral. Ezen funkciók elérése minden látogató számára elérhető bejelentkezés nélkül, és a tartalmak szabadon böngészhetők.

2.3.1 A Híroldal

A híroldal főoldalként funkcionál oldalunkon. Ezen elem két részre bontható fel – a tájékoztató csúszkára és magára a hírekre.

A tájékoztató csúszka elemei egyszerű szövegekből állnak, emellett mindegyik rendelkezik egy háttérképpel. Ez a funkcionalitás azt a célt szolgálja, hogy az oldalra először látogató érdeklődők tisztában legyen a honlap szolgáltatásaival. A képek automatikusan mozognak megadott időközönként, de lehetőségünk van a képek között manuálisan is lépegetni a nyilak segítségével.



2.3.1. ábra – A tájékoztató csúszka a Hírek oldalon

A hír elemek a HÖK tisztségviselői (tehát a bejelentkezett felhasználók) által közzétett tájékoztató üzeneteket jelentik. Minden egyes hír rendelkezik egy címmel, egy törzsszöveggel, illetve egy közzététel időpontjával. A hírek közzétételi sorrend szerint helyezkednek el az oldalon, és a legfrissebb hír az oldal legtetején lesz elérhető. A hírek egymás mellett és alatt helyezkednek el csempés megoldással, mely a képernyő szélességének arányában változtatja az egy sorban elférő hír elemek számát.



2.3.2. ábra – Egy hír elem felépítése

2.3.2 Az Ülések oldal

Az ülések oldal felépítése hasonló a híroldaléhoz. Két fő elemből áll, melynek az egyike az önkormányzati bizottsági ülések összehívó tájékoztatója, valamint a szűrő panel a bizottságok neveire.

A szűrő célja, hogy az érdeklődők az általuk keresett bizottságok üléseinek időpontjairól és témáiról szerezhessenek tudomást. Az önkormányzat bármely testületére rászűrhetnek, melynek következtében csak az adott bizottság ülései kerülnek felsorolásra közzététel időpontja szerint rendezve.

Az ülések elemei hasonlóan az ülés címéből, az ülésösszehívó szöveg tartalmából, valamint a közzététel időpontjából épülnek fel. Az ülésösszehívó szövege egy megadott sablonszöveg, melyből egyszerűen megtudhatják a felhasználók, hogy pontosan mikor és hol kerül megtartásra az ülés, illetve kihez fordulhatnak kérdés esetén.

**ELTE IK HÖK Küldöttgyűlés rendes ülése - 2021.
12. 06. 18:00**

Tisztelt Küldöttek, Tisztségviselők! Kedves Hallgatók!

Összehívom az ELTE IK HÖK Küldöttgyűlés rendes ülését.
Időpont: 2021. 12. 06. 18:00
Helyszín: ELTE IK HÖK iroda (1117 Budapest, Pázmány Péter sétány 1/A, -1.66/B).

Az előzetes napirendi pontok a következők:

1. Tájékoztató
2. Feladatok
3. Egyebek

Az esetleges kimentéseket, illetve napirendipont-módosító javaslatokat az elnok@ikhok.elte.hu e-mail címre várom.

Tisztelettel,
Deák Dalma
Elnök
ELTE IK HÖK
1117 Budapest, Pázmány Péter sétány 1/A -1.66B
tel: +36 1 372 25 20
e-mail: elnok@ikhok.elte.hu
web: <http://ikhok.elte.hu>

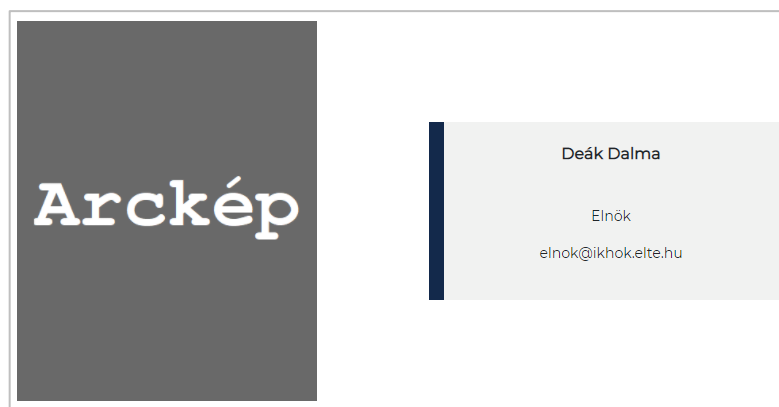
2021-11-26 00:09

2.3.3. ábra – Egy ülés elem felépítése

2.3.3 A Szervezet oldal

A HÖK honlapjának egyik új eleme az önkormányzati felépítést tartalmazó szervezet oldal. Nem csak a testület képviselőit tekinthetik meg az érdeklődők, hanem betekintést nyerhetnek a HÖK Elnökségének, Kabinetének, valamint bizottságainkat felépítésébe.

Az Elnökség és Kabinet almenü alatt minden tisztségviselőhöz tartozik egy arckép, a tisztségük neve és elérhetősége is a könnyű elérések biztosítása érdekében. Az elemek egymás alatt helyezkednek el úgy, hogy minden második elem jobbra kötött.



2.3.4. ábra – Az Elnökség almenü egyik eleme

A Bizottságok almenü alatt a bizottság nevei szerinti felbontásban tekinthetők meg a tagok, a Küldöttgyűlés alatt pedig a képviselők listája ABC sorrendben, megnevezve az egy vagy két bizottsági neveket is, melyeknek tagjaik az illetők.

Kommunikációs Bizottság	
Babos Zselyke Enikő	Tag
Balázs Kristóf	Tag
Bujáki Patrik Ferenc	Tag
Dobi Viktor	Tag

2.3.5. ábra – A Bizottságok almenü egyik eleme

2.3.4 Az Átláthatóság oldal

Szintén egy új elem az önkormányzat honlapján az Átláthatóság menüpont. A szervezeten belül történő döntésekről, pályázati lehetőségekről és ülésekről készült jegyzőkönyvekről kaphatnak részletes tájékoztatást az érdeklődők. Minden közzétett dokumentum letölthető bejelentkezés nélkül is a látogatók számára, emellett megnézhetik, mikor született meg a döntés, és az erről készült dokumentum mikor került feltöltésre a honlapra.

Név ▾	Feltöltés dátuma ▾	Letöltés
Animátorkoordinátori pályázat	2021-11-28 19:44	Letöltés

2.3.6. ábra – A Pályázatok almenü tartalma

Mivel a Hallgatói Önkormányzat kifejezetten fontosnak tartja, hogy a közéleti ösztöndíjaik mértékéről is feketén-fehéren legyenek tájékoztatva a hallgatók, ezért az átláthatóság menüpont alatt a közéleti ösztöndíjak oldalon ezeket közzé is tettem. Az itt található táblázatban a tisztség megnevezése mellett a pontos összeg is megjelenik.

Közéleti ösztöndíjak	
Elnök	10.000 Ft
Gazdasági és pályázati ügyekért felelős alelnök	10.000 Ft
Stratégiai és innovációs alelnök	10.000 Ft

2.3.7. ábra – A Közéleti ösztöndíjak almenü tartalma

2.3.5 Az Ösztöndíjak oldal

Az újonnan megjelenő Ösztöndíjak oldal célja, hogy a Kar és az Egyetem által nyújtott ösztöndíj lehetőségekről tájékoztatva legyenek a hallgatók. Ezek közé tartoznak a szociális alapú támogatások, a rendszeres és egyszeri kari ösztöndíjak, valamint a tanulmányi ösztöndíj is. Az Ösztöndíjak menü alatt található almenük egyszerű szöveges tartalmakat jelentenek, melyeken a pályázati kiírások külső oldalra mutatnak – ennek oka, hogy ezen ösztöndíjakról leghamarabb az ELTE HÖK oldalán, illetve az ELTE IK HÖK Confluence oldalán találhatóak meg a tájékoztatások, ezáltal nem szükséges új fájlok feltöltése az oldalra minden félévben.

Egyszeri Kari Ösztöndíjak

Mik az Egyszeri kari ösztöndíjak?

Az **Egyszeri kari ösztöndíjak** (ISZTK ösztöndíjak) egyösszegű ösztöndíjakat jelentenek, melyek pályázat útján igényelhetők. Ezen teljesítmény alapú ösztöndíjakat kari szinten határozzák meg, ezáltal az Informatikai Karon az alábbi támogatások léteznek:

- Egyszeri sport és kulturális ösztöndíj,
- Egyszeri tudományos és szakmai ösztöndíj
- Egyszeri közéleti ösztöndíj

2.3.8. ábra – Részlet az Egyszeri kari ösztöndíjak almenü tartalmából

2.3.6 A Szolgáltatások oldal

Az Informatikai Kar hallgatói számára különféle szolgáltatások vehetőek igénybe, melyeket a Szolgáltatások menüpont foglal össze. Ezek közé tartoznak az egyes regisztrációs lehetőségek, valamint a Kar környéki lehetőségek összefoglalói is.

A kampusz melletti sorompó rendszerbe, illetve a kari épületekbe való belépéshez szükséges belépőkártya rendszerbe van lehetőségük regisztrálni a hallgatóknak, mely egészen eddig manuálisan történt az IK HÖK Gazdasági és pályázati ügyekért felelős alelnöke által. Az új honlap lehetőséget biztosít ezek online lebonyolítására, melyhez a Sorompó regisztráció, illetve a Belépőkártya regisztráció almenü került implementálásra.

Miért szükséges a sorompó regisztráció?

Autóval, esetleg motorral szeretnél bejárni az óráidra, és mindenképpen az egyetem épületei mellett parkolni? Erre csak akkor van lehetőség, ha regisztrálsz járművedet az épületeüzemeltetésnél.

A lent található regisztrációs űrlap megfelelő kitöltésével regisztrálhatod autódát vagy motorodát annak érdekében, hogy az egyetem területén belül is tudj parkolni. A belső parkolót sorompóval zárták el, így amennyiben bekerül rendszámod a rendszerbe, onnantól a portaszolgálat bármikor be fog téged engedni egy gombnyomásra. Amennyiben regisztrálsz, hamarosan megkeresünk Téged, amint sikeresen továbbítottuk igényedet az illetékesek felé!

Kérdés esetén keressetek minket a gazdasag@ikhok.elte.hu e-mail címen!

Sorompó Regisztráció

Név:	<input type="text"/>
Neptun-kód:	<input type="text"/>
E-mail cím:	<input type="text"/>
Telefonszám:	<input type="text" value="+36301234567"/>
Rendszám:	<input type="text"/>
Autó típusa:	<input type="text"/>
Belépőkártya száma:	<input type="text" value="Ha van"/>
Igénylés indoka:	<input type="text"/>
<input type="button" value="Küldés"/>	

2.3.9. ábra – A Sorompó regisztráció oldal tartalma

A hallgatóság számára biztosít a honlap egy tájékoztató felületet a Kampuszon, illetve a Kampusz környékén található étkezőkről, büfékről és éttermekről. Mindemellett megjelenik az oldalon a Hallgatói Önkormányzat két irodájának ügyfélfogadási ideje is, valamint a megközelítési lehetőségek is egyaránt.

Északi iroda
Északi irodánk az Északi épület déli kapuja felől közelíthető meg a -1. szinten.
Nyitvatartás:
Hétfő: 10:00 - 18:00
Kedd: 10:00 - 18:00
Szerda: 10:00 - 18:00
Csütörtök: 10:00 - 18:00
Péntek: 10:00 - 18:00

2.3.10. ábra – Az Iroda nyitvatartások almenü tartalmának részlete

2.3.7. Egyéb látogatói funkcionalitások

A korábban már emlegetett ELTE IK HÖK Confluence oldalra is biztosítottam átirányítási lehetőséget, mely a navigációs sávban található Tudástár gombbal érhető el. A szakdolgozatom írása alatt sajnos ezen oldal külső tényezőkből kifolyólag nem funkcionál, melynek következtében az oldalra való navigálás során hibaüzenettel találkozunk az érdeklődők – amint az ehhez csatolt virtuális szerver ismét elindul, a honlapon található Tudástár gomb ismételten megfelelően működni fog.

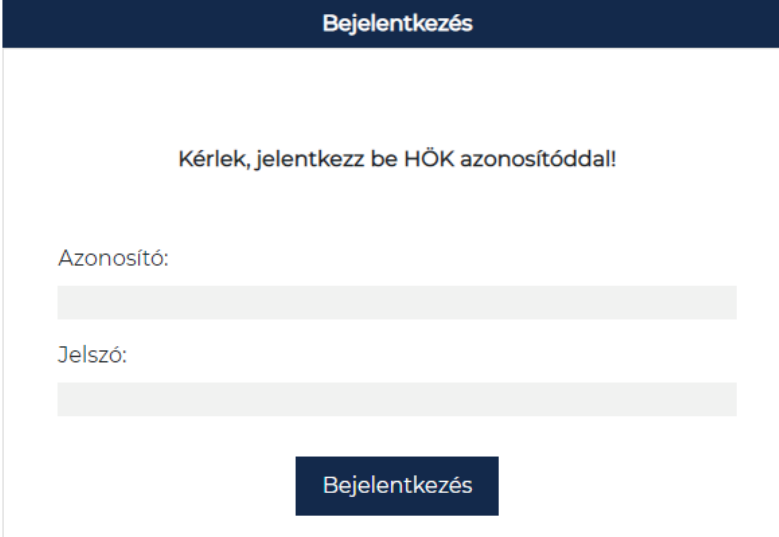
A honlap tetején található IK HÖK logóra kattintva bárhonnán vissza tudunk kerülni a főoldalra, azaz a Hírek oldalra.

A navigációs sáv legutolsó eleme egy ikon, melyre kurzorunkat helyezve láthatjuk, hogy az applikáció lehetőséget biztosít a bejelentkezésre, azonban a regisztrációra nem. Ennek oka, hogy a HÖK honlapjára csak a tisztségviselők és a küldöttgyűlési tagok kapnak hozzáférést annak érdekében, hogy a honlapon található információkat csak ők tudják javítani, pótolni és bővíteni, illetve csak a bejelentkezett tagok férhessenek hozzá a belső szervezeti funkcionalitásokra, melyeket a 2.4-es fejezetben fogok részletezni.

2.4 Az oldal felépítése bejelentkezéssel

Bár szakdolgozatomban látható, hogy megannyi funkcionalitás került bevezetésre, azonban adatik a kérdés – ezeket a tartalmakat ki és hogyan fogja feltölteni?

Ahogy az az előző fejezetben is említettem, a bejelentkezésre csak és kizárólag a Hallgatói Önkormányzat képviselői jogosultak. A bejelentkezés oldal a navigációs sávban található ikonnal érhető el.



The image shows a web form for login. At the top is a dark blue header with the text 'Bejelentkezés'. Below it, centered, is the text 'Kérlek, jelentkezz be HÖK azonosítóddal!'. There are two input fields: the first is labeled 'Azonosító:' and the second is labeled 'Jelszó:'. Below the input fields is a dark blue button with the text 'Bejelentkezés'.

2.4.1. ábra – A bejelentkezési felület tartalma

A bejelentkezést követően a navigációs sávban két ikon lesz látható – a Kijelentkezésé, valamint az Oldalmenüé. A belső funkciók az utóbbi elemmel érhetőek el, melyek közé tartozik a profil, az új bejegyzés létrehozása, a havi beszámolókkal kapcsolatos almenü, a Szolgáltatások oldalon található belépőkártya és sorompó regisztráció admin felülete, a határozatok tára, illetve az admin panel. Ezen tartalmak alapvetően nem jelennek meg minden tisztségviselő számára, hiszen egyes funkciókhoz (hozzáférési jogosultságokhoz) lesznek kötve, melyeket a dolgozatomban a későbbiekben fogok részletezni.

2.4.1 A Profil oldal

A bejelentkezett felhasználók a profil almenü pontban tekinthetik meg a hozzájuk tartozó információkat. Ezen oldalon lehetőséget kapnak a nevük, illetve tisztségviselői e-mail címük módosítására, emellett jelszavaik módosítására.

A jelszómódosítási panel szigorúan ellenőrzi a megadott jelenlegi jelszót, illetve összesíteni fogja a kétszer megadott új jelszó közötti különbségeket. Amennyiben ezek bármelyikével hiba történné, a felhasználó egy hiba modal-t fog látni a képernyőn. Hiányos kitöltés esetén az oldal nem engedélyezi az adatok mentését, és a mentés gomb inaktívvá válik.

Felhasználói adatok	Jelszómódosítás
Név: <input type="text" value="Deák Dalma"/>	Jelenlegi jelszó: <input type="password"/>
E-mail cím: <input type="text" value="elnok@ikhok.elte.hu"/>	Új jelszó: <input type="password"/>
<input type="button" value="Mentés"/>	Új jelszó ismét: <input type="password"/>
	<input type="button" value="Mentés"/>

2.4.2. és 2.4.3. ábra – A felhasználói adatok módosítási panelja, illetve a jelszómódosítási panel

2.4.2 Az Új bejegyzés oldal

Az Új bejegyzés almenü célja, hogy a megfelelő jogosultsági körökkel rendelkező bejelentkezett felhasználók létrehozassanak információkat az érdeklődők számára. Ezen bejegyzések közé tartoznak hírek, az ülésösszehívások, az átláthatóság menüpont alatt található beszámolók, pályázatok és jegyzőkönyvek, emellett a határozatok és a tisztségviselői havi beszámolók.

Új bejegyzés létrehozásakor a felhasználó egy legördülő menüből választhatja ki a neki szükséges bejegyzéstípust, melynek megfelelően a honlap dinamikusan megváltoztatja az ahhoz tartozó bemeneti mezőket. Meglévő bejegyzések szerkesztésekor erre az oldalra kerül átirányításra a felhasználó, a bemeneti mezők pedig automatikusan kitöltésre kerülnek a szerkesztendő elem adataival.



Bejegyzés létrehozásakor a bejegyzések nagy része rendelkezik „közzététel időpontja” bemeneti mezővel. Ezek automatikusan kitöltésre kerülnek a szerver dátumának és idejének megfelelően.

Bejegyzés létrehozása, szerkesztése	
Bejegyzés típusa:	<input type="text" value="Átláthatóság - Beszámoló"/>
Cím:	<input type="text"/>
Közzététel időpontja:	<input type="text" value="2021. 11. 30."/> <input type="button" value="📅"/>
	<input type="text" value="20:49"/> <input type="button" value="🕒"/>
Csatolmány:	<input type="button" value="Fájl kiválasztása"/> <input type="text" value="Nincs fájl kiválasztva"/>
<input type="button" value="Küldés"/>	

2.4.4. ábra – Az új bejegyzés menüpont új beszámoló esetén

2.4.3 A Saját beszámolók oldal

A Saját beszámolók almenü alatt a tisztségviselők lehetőséget kapnak, hogy a már leadott havi beszámolóikat megtekinthessék, szerkeszthessék, illetve törölhessék. Az oldal megjeleníti a beszámoló leadásának évét és hónapját is. A szerkesztés során az oldal átirányítja a felhasználót az Új bejegyzés almenübe, ahol a meglévő adatokat fel is tünteti.




Saját beszámolók		
Év	Hónap	Műveletek
2021	október	 
2021	november	 



2.4.5. ábra – A Saját beszámolók oldal tartalmának részlete

2.4.4 A Beszámolók összegzése oldal

A Beszámolók összegzése almenüben egyes tisztségviselők összesíthetik a már beérkezett havi beszámolókat. Ezen oldalon szintén évek és hónapok szerint kerülnek csoportosításra az elemek, és minden egyes sor végén található egy ikon, melyre kattintva megjeleníthetők az adott hónapra beérkezett beszámolók.

A beérkezett beszámolók tartalmazzák a leadó személy tisztségét, a leadás pontos időpontját és a beszámoló tartalmát. Ezen adatokat szintén lehet szerkeszteni, illetve törölni, emellett szerkesztés során az oldal átirányítja a felhasználót az Új bejegyzés almenübe, ahol a meglévő adatokat fel is tünteti.

Beszámolók összegzése				
Év	Hónap	Műveletek		
2021	december			
2021	november			
2021	október			

Beérkezett beszámolók				
Év	Hónap	Beküldő	Beküldés ideje	Műveletek
2021	december	Gazdasági és pályázati ügyekért felelős alelnök	2021-11-26 15:10	 



2.4.6. ábra – A Beszámolók összegzése menüpont tartalma, a decemberi hónapra szűrve

Bár ez a funkcionalitás eddig is létezett az IK HÖK korábbi honlapján, szükség volt a leadás időpontjának pontos követésére is. Ennek oka az, hogy a tisztségviselők beszámolóikat minden hónap ötödike éjfélig tölthetik fel, és ezzel kiküszöbölhetőek az esetleges visszaélések.

2.4.5 A Határozatok tára oldal

A Határozatok tára tartalmazza az Önkormányzat döntéseit. Az itt tárolt információk magukba foglalják a sorszámot, a döntés születésének időpontját, a határozati javaslat szövegét, a jelenlévő mandátummal rendelkező képviselők számát és döntési arányt, a csatolmányt és a feltöltés időpontját. Az adatok feltöltés szerint vannak rendezve, és hasonlóképpen, mint a jegyzőkönyveknél, itt is található bizottsági szűrő menü.

A bizottsági szűrő lényege, hogy az egyes bizottságokhoz tartozó döntéseket könnyedén elérhessék a honlap látogatói. Az itt található bejegyzéseket lehet szerkeszteni, illetve törölni, emellett szerkesztés során az oldal átirányítja a felhasználót az Új bejegyzés almenübe, ahol a meglévő adatokat fel is tünteti.



Sorszám	Dátum	Határozat	Mandátum	Szavazati arány	Csatolmányok	Feltöltve	Műveletek
1/2021/Kommunikációs Bizottság	2021-10-01 18:00	Az ELTE IK HÖK Kommunikációs Bizottsága a mellékletben található animátorkoordinátori pályázatot kiírja 2021. 10. 20., 18:00-s határidővel.	10	9/0/1	Letöltés	2021-10-02 17:21	 



2.4.7. ábra – A Határozatok tára menüpont tartalmának egy részlete

2.4.6 A Sorompó adminisztráció oldal

A Szolgáltatások menüpont alatt található Sorompó regisztráció során a látogatók jelentkezhetnek parkolási jogosultságokra a kampusz területén. Ennek adminisztrációjáért javarészt a HÖK felel, melyet ezen menüpont alatt lehet elérni.

Az adminisztráció két fő eleme a még függőben lévő jogosultsági kérések, valamint a már aktivált regisztrációk listája. Előbbiben, mint ahogyan az a 2.4.8-as ábrán is látható, egyes adatok hiányosak. Ezen információkkal a regisztráló hallgató még nem rendelkezik, ezért a kérelem aktiválása előtt a felelős tisztségviselőnek ebben a menüpontban módosítania kell a hiányzó részeket.

Sorompó jogosultság kérések									
Név	Neptun-kód	Rendszám	Típus	Belépőkártya száma	Telefonszám	E-mail cím	Regisztrált	Szemeszter	Műveletek
John Doe	NEPTUN	PLATE1	Suzuki Swift	-	+361111111	johndoe@gmail.com	2021-11-26 14:57	-	[M]  

Aktív sorompó jogosultságok listája									
Név	Neptun-kód	Rendszám	Típus	Belépőkártya száma	Telefonszám	E-mail cím	Regisztrált	Szemeszter	Műveletek
Jane Doe	NEPTUN	PLATE2	Toyota Yaris	22222222222	+36222222222	janedoe@gmail.com	2021-11-26 15:04	2021/2022/1	[M]  



2.4.8. ábra – A Sorompó regisztráció oldal tartalma



A hallgató által megadott adatok a név, a Neptun-kód, a jármű rendszáma és típusa, az esetleges belépőkártya száma, a telefonszáma, az e-mail címe és egy indoklás. Mivel ezen adatmennyiséget egyetlen táblázatban megjeleníteni nem előnyös, ezért a hallgatói indoklásokat a kurzorral a megfelelő sorban található [M] betűre lebegtetve érhetjük el. Az itt található bejegyzéseket lehet szerkeszteni és törölni, emellett szerkesztés során az oldal átirányítja a felhasználót az Admin panel almenübe, ahol a meglévő adatokat fel is tünteti.

2.4.7 A Belépőkártya adminisztráció oldal

A Szolgáltatások menüpont alatt található a Belépőkártya regisztráció is, ahol a látogatók belépőkártyát igényelhetnek a Kar épületeihez. Ennek adminisztrációjáért szintén a HÖK felel, melyet ezen menüpont alatt lehet elérni.

Hasonlóan a sorompó adminisztrációhoz, ez az oldal is két elemből tevődik össze – a kártyaigénylésekből és a már aktivált kártyák listájából. A belépőkártya kéréseknél jelennek meg a hallgatók által leadott igénylők, melyek nem rendelkeznek minden szükséges adattal a sikeres regisztrációhoz. Ezeket az adatokat a kártya kiállítása után a felelős tisztségviselő fogja intézni.

Belépőkártya kérések								
Név	Neptun-kód	E-mail cím	Diákigazolvány száma	Belépőkártya száma	Jogosultság	Regisztráció időpontja	Leadás időpontja	Műveletek
John Doe	NEPTUN	johndoe@gmail.com	111111111	-	-	2021-11-26 14:58	-	[M]  

Aktív belépőkártyák listája								
Név	Neptun-kód	E-mail cím	Diákigazolvány száma	Belépőkártya száma	Jogosultság	Regisztráció időpontja	Leadás időpontja	Műveletek
Jane Doe	NEPTUN	janedoe@gmail.com	22222222222	22222222222	IKHOKI	2021-11-26 14:58	-	[M]  

2.4.9. ábra – A Belépőkártya regisztráció tartalma

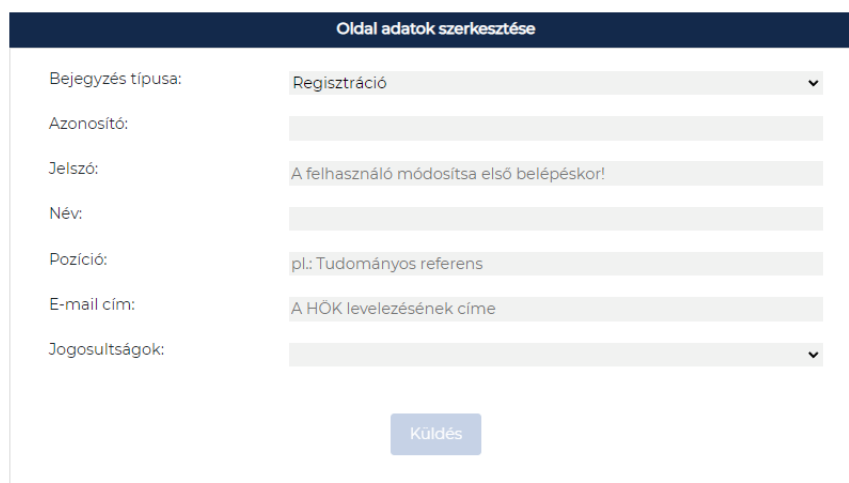
A regisztráló látogatónak szükséges megadnia a nevét, a Neptun-kódját, az e-mail címét, illetve a diákigazolványának számát a sikeres regisztráció érdekében. Mivel ezen adatmennyiséget egyetlen táblázatban megjeleníteni nem előnyös, ezért a hallgatói indoklásokat a kurzorral a megfelelő sorban található [M] betűre lebegtetve érhetjük el. Az itt található bejegyzéseket lehet szerkeszteni és törölni, emellett szerkesztés során az oldal átirányítja a felhasználót az Admin panel almenübe, ahol a meglévő adatokat fel is tünteti.

2.4.8 Az Admin panel oldal

A már többször emlegetett admin panel funkcionalitása meglehetősen hasonló az új bejegyzések létrehozásának almenüjével. Célja, hogy a honlapon található információkat a megfelelő jogosultságokkal rendelkező regisztrált felhasználók módosíthassák, illetve létre is hozhassák. Ezen adatok közé tartoznak a szervezet menüpont alatt található Elnökség, Kabinet és Küldöttgyűlés tagjai, a szolgáltatások alatt található étkezők és irodák, a belépőkártya és sorompó regisztrációk, illetve a honlapon lévő regisztrált fiókok.

Utóbbinak itt történő elhelyezése mellett azért döntöttem, mert tisztségviselői és képviselői fiókok regisztrálására évente egyszer van szükség, akkor is tömbösítve. Ha a honlapon lehetőséget biztosítanánk a regisztrációra, előfordulhat, hogy nem képviselő látogatók is megpróbálnának fiókot létrehozni a honlapon, melyet vissza kell utasítanunk.

Új regisztrált fiók létrehozásakor szükséges megadni egy egyedi azonosítót, egy kezdetleges jelszót (melyet a felhasználó azonnal módosítani is tud első belépéskor), egy nevet, egy tisztséget, e-mail címet, valamint jogosultsági kört. Bármelyik információ hiányában – mint ahogyan bármilyen más adat létrehozásakor vagy szerkesztésekor – a mentés nem lehetséges, az erre szolgáló gomb nem elérhető.



2.4.10. ábra – Az Admin panel kinézete új regisztráció létrehozása esetén

Új adat létrehozásakor a szükséges opciót egy legördülő menüből választhatjuk ki, melyet követően az adott adatstruktúrához tartozó bemeneti mezők, szöveges bemenetek és checkbox-ok fognak dinamikusan megjelenni. Ilyen adatok szerkesztésekor hasonlóan erre az oldalra kerül átirányításra a felhasználó, a bemeneti mezők pedig automatikusan kitöltésre kerülnek a szerkesztendő elem adataival.

2.4.9. Egyéb felhasználói funkcionalitások

Ezen alapvető funkciók implementálása mellett kiemelten fontos, hogy a felhasználói élményt is fejlesszük. Éppen ezért a honlap azon oldalain, ahol nagyobb mennyiségű adatra számíthatunk, ott az oldal alján elhelyezésre kerültek lapozók. Hozzávetőlegesen 24-25 elem fér el mindegyik oldalon, legyen az ülés, jegyzőkönyv, vagy akár hír.

Előfordulhat, hogy lassabb válaszidejű szerveret használunk, esetleg a felhasználók internet sebessége nem megfelelő a honlap hibátlan működéséhez. Ezen eshetőségekre a honlap oldalain loading spinner-ökkel találkozhatunk, melyek célja, hogy a már azon megjelent menüpontokban, melyeken az adatbázisból történő lekérdezés még nem ment véghez, vagy az adatok betöltése még nem történt meg, ott egy üres kijelző helyett egy animáció jelezze a betöltés folyamatát.

A biztonságosabb kezelhetőség érdekében a honlapon megerősítő ablakok ugranak fel, melyek megakadályozhatják a véletlen közzétételeket, módosításokat vagy törléseket. Ezen modal-ok a képernyő tetején jelennek meg bármilyen mentés vagy törlés esetén, ahol a felhasználóknak lehetőségük van a visszalépésre vagy a megerősítésre.

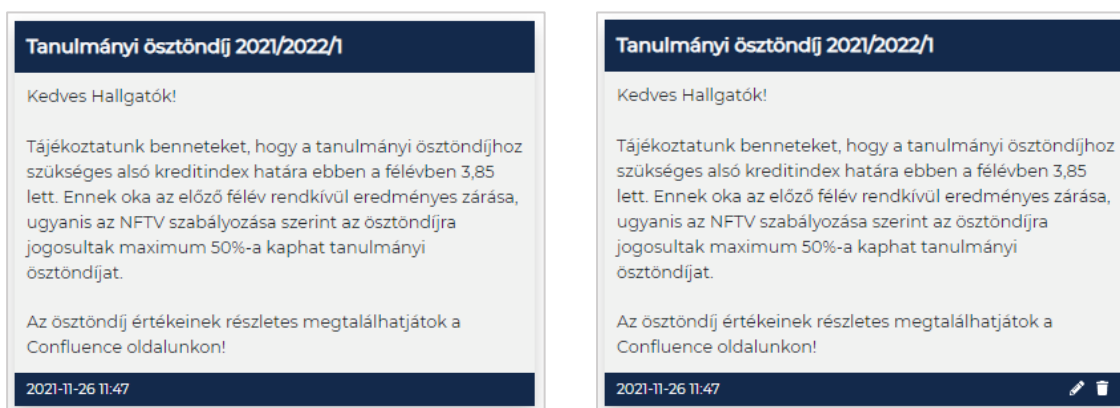
Mindezek mellett, amennyiben felhasználónk egy olyan oldalt szeretne elérni URL címen keresztül, mely nem létezik, akkor egy olyan komponenst fog maga előtt látni, mely a „Sajnos ez az oldal nem létezik” hibaüzenet tartalmazza, és egy figyelemfelkeltő képet.

2.5 Hozzáférés vezérlés, jogosultsági körök

A már sokszor emlegetett hozzáférés vezérlés egy kiemelten fontos aspektus egy ilyen volumenű applikáció megfelelő működésében. Mivel az oldalra való bejelentkezést követően elérhetővé válik egy Admin panel menüpont, így már a neve is azt implicálja, hogy ehhez az autorizáció nem mindenki számára javallott.

Éppen ezért 5 kategóriába osztottam szét a jogosultságokat, melyek függvényében változnak a hozzáférések szintjei, és a képernyőn megjelenő lehetőségek.

- Admin: Bejelentkezést követően minden funkció elérhető számára, admin közzétételeket és bejegyzéseket tud szerkeszteni
- Elnökség: Az admin felület kivételével minden funkció elérhető számára, bejegyzéseket tud szerkeszteni,
- Kabinet: Elérhető számára a profil, a saját beszámolók, a beszámolók összegzése és a határozatok tára. A honlapon bejegyzést nem tud szerkeszteni.
- Jegyzőkönyvvezető: Elérhető számára a profil, a beszámolók összegzése és a határozatok tára. A honlapon bejegyzést nem tud szerkeszteni.
- Küldött: Csak és kizárólag a profil és a beszámolók összegzése funkció érhető el számára. A honlapon bejegyzést nem tud szerkeszteni.



2.4.11. és 2.4.12. ábra – Egy ülés bejegyzés bejelentkezés után, nem megfelelő és megfelelő jogosultságokkal

Amennyiben a bejelentkezett felhasználó nem rendelkezik megfelelő jogkörökkel, úgy nem csak az oldalsó navigációs sávban nem jelennek meg a szükségtelen funkcionálisok, hanem a már létrehozott bejegyzéseknél sem lesz elérhető a szerkesztés, illetve a törlés gomb (2.4.11. ábra). Mindemellett, amennyiben egy felhasználó egy olyan URL címet szeretne elérni, melyhez nem megfelelő a hozzáférése, az oldal visszairányítja a látogatót a főoldalra.

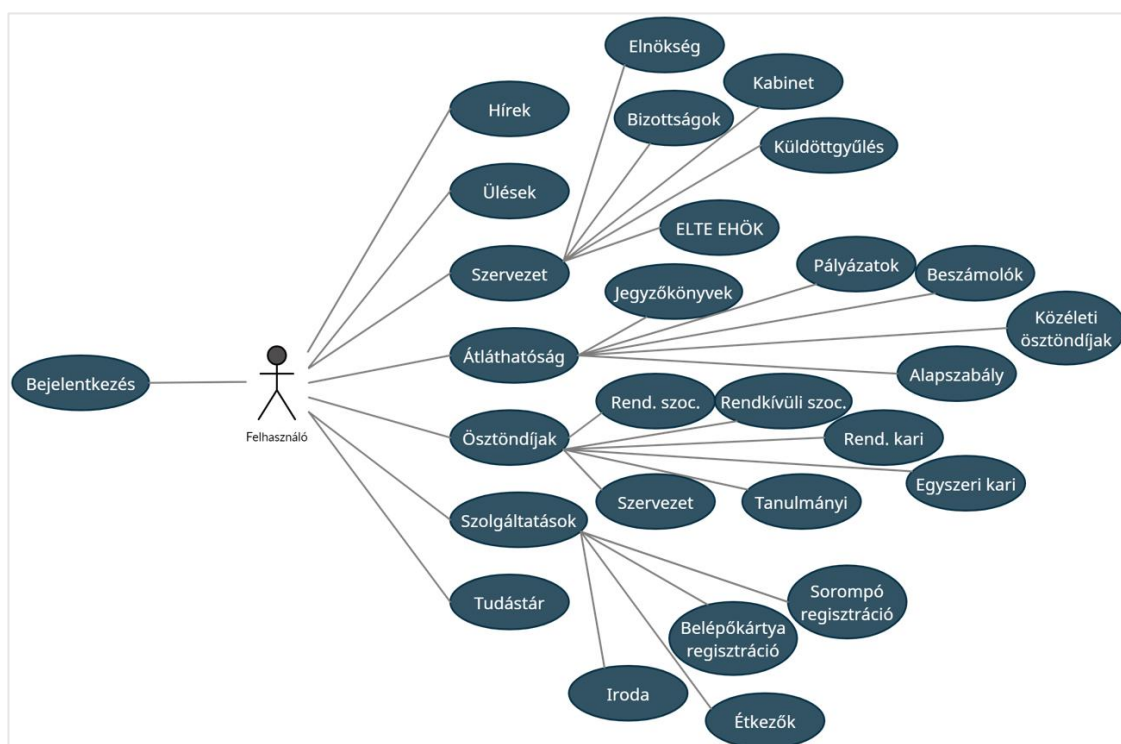
3. FEJLESZTŐI DOKUMENTÁCIÓ

3.1 A feladat leírása

Célunk, hogy egy olyan optimalizált webes applikációt készítsünk, mely az ELTE IK Hallgatói Önkormányzatának alapvető szükségletei mellett ellátja az új funkcionalitások igényeit is, legyen az hírek közzététele, vagy a belső működés megfelelő biztosítása. Elsősorban a hallgatóság felé történő kommunikációt kell megvalósítania, melyet hírek, ülések, átláthatóságot biztosító dokumentumok, ösztöndíj – és szolgáltatáslehetőségek, valamint a szervezeti felépítés segítségével tehet meg. Mindezek mellett el kell látnia a HÖK belső szervezeti szükségleteit is, melyek közé tartozik a beszámolók leadásának lehetősége és azok összesítése, a határozatok tárának vezetése és a hallgatói szolgáltatások adminisztrálása.

3.2 A felhasználói esetek

A be nem jelentkezett, illetve bejelentkezett felhasználók felhasználói eseteit ábrázolják a következő diagrammok.



3.1.1. ábra – Felhasználói esetdiagramm be nem jelentkezett felhasználó esetén



Az esetdiagrammon látható, hogy egyes jogosultsági köröket különböző színekkel jelöltem. A színek azt jelentik, hogy legalább olyan szintű jogkörrel kell rendelkeznie a bejelentkezett felhasználónak annak érdekében, hogy az adott funkcionalitást használni tudja.

Mivel a hozzáférések hierarchikusan lettek kialakítva, ezért fentről lefelé a legnagyobb hozzáféréstől a legkisebbig az alábbi autorizációk jelennek meg az applikációban:

- Admin – zöld
- Elnökségi tag – citromsárga
- Kabinet tag – narancssárga
- Jegyzőkönyvvezető – piros
- Küldött – rózsaszín
- Bármely felhasználó - sötétékék

A 3.1.2-es ábra értelmében azok az elemek, melyek zöld színűek, csak és kizárólag az admin jogosultsággal rendelkező bejelentkezett felhasználók számára érhetőek el. A citromsárgák csak az adminoknak és elnökségi tagoknak, a narancssárgák az adminoknak, elnökségi tagoknak és kabinet tagoknak, és így tovább.

Az ábrán láthatóak „Sz” és „T” betűvel jelölt esetek, melyek az adott bejegyzés szerkesztését és törlését jelölik. Több esetben előfordul, hogy a bejegyzéstípus megtekintésének jogköre alacsonyabb szintű, mint annak szerkesztése vagy törlése.

3.3 Alkalmazott technológiák, használt programcsomagok

A szoftver nem pusztán az 1.2. bekezdésben megnevezett technológiákat használja. Az applikációban olyan kiegészítő lehetőségeket biztosító csomagokat és lehetőségeket használok, melyek a későbbiek során a program fejlesztésekor nagymértékben megkönnyíti az új fejlesztők munkáját, és egyszerűsítik számukra a kódok értelmezését.

3.3.1 Optimalizált applikáció

A már emlegetett optimalizált rendszer kialakítása egy tág fogalomnak bizonyosul. Nem csak felhasználói oldalon érhetjük el a kívánt teljesítményt, de kifejezetten fontos, hogy a szoftver működése is ennek megfelelően legyen kialakítva. Ezen program optimalizált működésének bizonyításához példának fogom venni a HÖK jelenlegi honlapján.

Először is, egy webes applikációnak elengedhetetlen, hogy a megszokott 1920*1080-as képernyőméret mellett más eszközökre is ki legyen szabva. Ez bár a jelenlegi oldalon nem

teljesült, ebben az applikációban törekedtem mobil- és tabletnézet hiánytalan kialakítására, de ezen méretek között is bármely kijelző méreten funkcionál a honlap.

A felhasználói élmény is fontos aspektus egy optimalizált applikáció implementálásakor. A Search Engine Journal egy cikke¹³ tökéletesen hivatkozik a UX fejlesztésének elemeiről, és 7 támponton közelíti meg a cél elérését:

- **Hasznos:** A honlap tartalmának mindenképpen egyedinek kell lennie, ami kielégíti a felhasználói igényeket.
- **Használható:** A honlap használata legyen egyszerű, és adjon lehetőséget a könnyű navigációra.
- **Vonzó:** A honlap arculata és megjelenése idézze elő a felhasználókban a nagyrabecsülést, biztosítson pozitív értékelést,
- **Megtalálható:** A honlapon a navigációs és dizájn elemek integrálódása tegye egyszerűvé az oldalon történő keresést.
- **Elérhető:** A honlapon lévő tartalmak legyenek elérhetőek minden látogató számára.
- **Megbízható:** Az honlap legyen megbízható, és érje el, hogy felhasználói nyugodtan, bizakodva használják az oldalt.
- **Értékes:** A honlap biztosítson megfelelő élményeket a felhasználók számára, ezzel elősegítve a tulajdonos szervezet működését.

Ezen célkitűzéseknek megfelelően alakult meg a honlap látványterve. A navigáció, az aloldalak elérése és a könnyen kezelhetőség biztosított, a tartalmat pedig a Hallgatói Önkormányzat fogja kezelni sablontartalmaknak megfelelően.

3.3.2 Typescript

A Typescript egy open-source programozási nyelv, mely egy erősen típusos kibővítés a Javascript nyelvhez, és a fordításkor nemcsak típusellenőrzéseket hajt végre, hanem a fájlokon szemantikai és szintaktikai elemzéseket vezet végig. Ezen nyelvet nem csak a frontenden alkalmazom, hanem a backend fájljain is egyaránt.

3.3.3. Single Page Application

Egyoldalas alkalmazásnak (SPA) nevezünk olyan webalkalmazásokat, amelyek dinamikusan módosítják a jelenlegi oldalt ahelyett, hogy teljesen új oldalakat (HTML fájlokat) töltené be a

¹³ <https://www.searchenginejournal.com/website-optimization-essentials/280641/#close>

szerverről. Ez nagymértékben optimalizálja az oldalak közötti navigáció felhasználói élményét, egy reszponzív élményt nyújt a látogatóknak, emellett csökkenti a szerver irányába nyújtott felesleges lekérdezéseket.

3.3.4 RESTful API

A REST API (REpresentational State Transfer) egy állapotmentes szoftverarchitektúra típus, mely a backend-en http protokoll alapú kommunikációt ír le JSON alapú adatok segítségével. Használatának célja, hogy bármilyen kliensről, legyen az web, tablet vagy telefon, ugyanolyan módon érhesük el a szükséges információkat és a biznisz logikát. Ezen interfész állapotmentességének köszönhetően csak a szerverkérésben lévő adatokkal kell dolgoznia, ezáltal nem függ semmilyen kontextustól.

3.3.5 Angular Forms

Az Angular Forms segítségével a sablon alapú űrlapok (Template-driven Forms) készítésekor volt segítségemre. Ilyen űrlapokat használtam azon bejegyzések létrehozásának oldalán, melyekhez nem volt szükség fájlfeltöltésre.

3.3.6 Angular Animations

Ezen package használatára azért volt szükség a szoftver implementálásakor, mivel a betöltő képernyőkor a loading spinner-ek animációként funkcionálnak, és ezek egy natív Angular applikációban nem működnek segédcsomagok nélkül.

3.3.7 Angular Router

A honlapon történő átirányítások az Angular Router package segítségével lettek kialakítva. Ennek oka, hogy az applikáció Single Page Application-ként funkcionál, ezáltal az átirányítások nem új HTML fájlok betöltésével történik, hanem a komponensek közötti navigációval.

3.3.8 NgImageSlider

A híroldalon található képnézegető elem az NgImageSlider segítségével került létrehozásra.

3.3.9 NgxBootstrap

Bár az applikációhoz alpból hozzákötésre került a Bootstrap, az oldalon található megerősítő és figyelemfelkeltő modal-okat az NgxBootstrap csomag segítségével hoztam létre.

3.3.10 NgxPagination

Minden olyan oldalon, melynek alján lapozgató található, azok az NgxPagination package segítségével lettek kialakítva. A csomag segítségével megoldható volt a nyelvesítés és a színek módosítása is, a használata pedig egyszerű.

3.3.11. Http

Ezen Node.js modul segíti az adatmozgást adatbázisunkkal http protokollok segítségével. A modul létrehoz egy http szerveret, mely hallgatja a szerverünk portjait, és válaszokkal (response) tér vissza. Ezen csomag segítségével kerültek implementálásra a GET, POST, PUT, PATCH és DELETE metódusok.

3.3.12 JsonWebToken

Ez a Node.js csomag a honlapon történő bejelentkezés adminisztrálásában segít. Ennek segítségével felek között tudunk biztonságosan adatot átadni JSON formátumban.

3.3.13. Bcrypt

A Bcrypt a felhasználói jelszavak titkosításáért felel mind bejelentkezéskor, mind jelszómódosításakor, mind pedig regisztrációs fiók létrehozásakor.

3.3.14. Multer

A Multer egy Node.js middleware, melynek használatával egyszerűen kivitelezhetőek a kép- és fájlfeltöltések, a feltöltött fájlok kiterjesztésének ellenőrzése, valamint a mime type validálás.

3.4 Az adatbázis felépítése

Mielőtt részletesebben belemerülnék a szoftver működésének menetébe és metodikájába, érdemes megnéznünk az adatbázisban tárolt adatok felépítését. Bár a MongoDB, ahogyan azt korábban említettem, nem erőlteti az adatsémákat, kivétel nélkül minden ugyanolyan

dokumentumba tartozó elem felépítése megegyezik, ezáltal a hozzájuk tartozó objektumokban megtalálható az összes kulcs.

Dokumentum / Séma neve	Hozzá tartozó kulcsok	Megjegyzés
Auth	postType (string)	Az autentikációért felel, a felhasználó sémát írja le
	identifier (string) – szükséges, egyedi	
	fullName (string) - szükséges	
	position (string) - szükséges	
	email (string) - szükséges	
	permissions (string) - szükséges	
	password (string) - szükséges	
Belépőkártya	postType (string)	A belépőkártya regisztrációk adatainak tárolásáért felel
	fullName (string) - szükséges	
	neptun (string) - szükséges	
	email (string) - szükséges	
	studentId (string) - szükséges	
	card (string)	
	permissions (string)	
	date (string)	
	returnDate (string)	
	reason (string) - szükséges	
	isApproved (boolean)	
Beszámolók	postType (string)	Az átláthatóság menüpontban található beszámolók sémáját írja le
	title (string) - szükséges	
	date (string) - szükséges	
	file (string) - szükséges	
Büfék	postType (string)	A büfék és étkezők bejegyzési típus sémája
	name (string) - szükséges	
	brief (string) - szükséges	
	openHours (string) - szükséges	
Elnökség	postType (string)	Az elnökség menüpontban lévő adatok sémája
	name (string) - szükséges	
	position (string) - szükséges	
	email (string) - szükséges	
	file (string) - szükséges	
Határozatok	postType (string)	

	committee (string) - szükséges	Az oldalmenüben található határozatok tára elemeinek sémája
	title (string) - szükséges	
	number (string) - szükséges	
	decisionDate (string) - szükséges	
	content (string) - szükséges	
	mandate (number) - szükséges	
	vote (string) - szükséges	
	date (string) - szükséges	
	file (string)	
Havi beszámolók	postType (string)	Az oldalmenüben lévő havi beszámolók sémája, mely a saját beszámolók és a beszámolók összegzésében van használatban
	author (Auth) - szükséges	
	year (string) - szükséges	
	month (string) - szükséges	
	content (string) - szükséges	
	date (string) - szükséges	
Hírek	postType (string)	A hírek elemek sémája a főoldalon
	title (string) - szükséges	
	content (string) - szükséges	
	date (string) - szükséges	
Iroda	postType (string)	Az iroda bejegyzési típus sémája
	name (string) - szükséges	
	brief (string) - szükséges	
	openHours (string) - szükséges	
Jegyzőkönyvek	postType (string)	Az átláthatóság oldalon lévő jegyzőkönyvek sémája
	committee (string) - szükséges	
	title (string) - szükséges	
	decisionDate (string) - szükséges	
	date (string) - szükséges	
	file (string) - szükséges	
Kabinet	postType (string)	A kabinet menüpontban lévő adatok sémája
	name (string) - szükséges	
	position (string) - szükséges	
	email (string) - szükséges	
	file (string) - szükséges	
Közéletik	postType (string)	Az átláthatóság oldalon lévő közéleti ösztöndíjak sémája
	name (string) - szükséges	
	amount (string) - szükséges	
Küldöttgyűlés	postType (string)	

	fullName (string) - szükséges	A küldöttgyűlés bejegyzési típus sémája
	firstPosition (string) - szükséges	
	firstCommittee (string) - szükséges	
	secondPosition (string)	
	secondCommittee (string)	
	email (string)	
Pályázatok	postType (string)	Az átláthatóság menüpontban található pályázatok sémáját írja le
	title (string) - szükséges	
	date (string) - szükséges	
	file (string) - szükséges	
Sorompó	postType (string)	A sorompó regisztrációk adatainak tárolásáért felelő séma
	fullName (string) - szükséges	
	neptun (string) - szükséges	
	plate (string) - szükséges	
	type (string) - szükséges	
	card (string)	
	phone (string) - szükséges	
	email (string) - szükséges	
	date (string)	
	semester (string)	
	reason (string)	
	isApproved (string)	
Ülések	postType (string)	Az ülés elemek sémájának leírója
	author (Auth) - szükséges	
	committee (string) - szükséges	
	type (string) - szükséges	
	title (string) - szükséges	
	decisionDate (string) - szükséges	
	content (string) - szükséges	
	date (string) - szükséges	

Láthatjuk, hogy ezen típusok között gyakran előfordulnak string-ek. Ennek oka, hogy az adatbázisunk kötött mennyiségű típust ismer fel, ezáltal az olyan magasrendűeket, mint a File vagy Date nem adhatunk meg. Ezen típusok átalakítását a REST API hívásokkor, vagy a frontenden fogjuk elvégezni.

A postType minden sémában megjelenik, melynek célja, hogy ezen kulcs alapján megvalósíthassuk a bejegyzések szerkesztését. Ez az adat egy poszt szerkesztésekor

megjelenik az URL-ben is, mely alapján az Új bejegyzés vagy Admin panel oldalon kikeresésre kerül a szükséges form a modifikációk végrehajtásához.

3.5 A backend felépítése

A backend a már korábban említett sémák és modulok mellett tartalmazza az adott adattípusokhoz tartozó, Express.js segítségével megalkotott route-okat és middleware-eket, a files és images mappát, a server.js fájlt, valamint az app.js fájlt is.

3.5.1 Routes

A routes mappában található fájlokban a http metódusokat és REST API hívásokat gyűjtöttem össze, melyeket a hozzájuk tartozó modulok szerint szortíroztam. Ez elősegíti a könnyebb kiolvasást, és csak a megfelelő sémához tartozó metódusokat tároljuk egy helyen.

A használt http request metódusok közé tartozik a GET, a POST, a PUT, a PATCH és a DELETE. Mindegyik request metódushoz kapcsolódik legalább egy REST API hívás.

```
router.get('', (req, res, next) => {
  Post.find()
    .then(documents => {
      res.status(200).json({
        message: 'Posts fetched successfully',
        posts: documents
      });
    });
});
```

Az alábbi függvény felel a GET metódus funkcionalitásának biztosításáért, mellyel az adatbázisban tárolt Post (hírek) típusú elemeket adja meg visszatérési értéként. A router.get() függvény az Express router egyik callback metódusa, mely akkor kerül meghívásra, amikor ezen a végponton kérés érkezik. A többi modellben hasonlóan került kialakításra a GET metódus, egyes esetekben az első paraméterben megadásra került egy string, mely segítségével nem az összes elemet kapjuk vissza a documents argumentummal, hanem csak a paraméterben megadott feltételnek megfelelőeket – ezzel elérhetjük, hogy egy adott hírt kikeressünk az adatbázisból ID alapján, vagy akár többet is a címük szerint.

```

router.post('', checkAuth, (req, res, next) => {
  const post = new Post({
    postType: req.body.postType,
    title: req.body.title,
    content: req.body.content,
    date: req.body.date
  });

  post.save().then( result => {
    res.status(201).json({
      message: 'Post added successfully',
      postId: result._id
    });
  });
});

```

Ebben a kódrészletben a POST metódus került implementálásra a Hírek route-ban. Ennek segítségével menthetünk el adatbázisunkba új hírt, melyet az oldalmenüben tehetünk meg az Új bejegyzés aloldalon. A beérkező request tartalma alapján létrehozunk egy új Post típusú objektumot, melyet a save() függvénnyel az adatbázisban eltárolunk. Az egyedi postId ekkor kerül hozzáadásra az adatstruktúrában.

```

router.put('/:id', checkAuth, (req, res, next) => {
  const post = new Post({
    _id: req.body._id,
    postType: req.body.postType,
    title: req.body.title,
    content: req.body.content,
    date: req.body.date
  })
  Post.updateOne({_id: req.params.id}, post).then(result => {
    res.status(200).json({
      message: 'Post updated successfully'
    });
  });
});

```

A Hírek modell egyik elemének szerkesztésekor nem a POST http metódust használjuk az elmentése, hiszen nem új elemként szeretnénk adatbázisunkhoz csatolni a szerkesztett bejegyzésünket. Ekkor használhatjuk a PUT vagy a PATCH metódust.

A PUT segítségével módosíthatunk egy adatbázisban lévő adatot úgy, hogy a kliens oldal megadása szerint minden kulcs-érték pár frissítésre kerüljön. Ez azt jelenti, hogy amennyiben a felhasználó az alap elemeken kívül újakat is hozzátesz, vagy nem töltene fel minden alap

elemét a modellnek, úgy az adatbázisban lévő dokumentumot teljesen kicseréli az újonnan elküldött adatstruktúrával.

A PATCH segítségével hasonlóan adatokat módosíthatunk, azonban ezzel a metódussal csak azon kulcs-érték párokat fogjuk frissíteni, melyek a kliens oldaláról érkeznek. Ha valaki egy bejegyzés címét szeretné csak frissíteni, de megtartaná mindegyik másik tulajdonságát, akkor ezt ezzel a metódussal teheti meg.

Mindkettő működéséhez szükséges megadnunk a függvény paraméterében egy postId-t, mely alapján a metódus kikeresi adatbázisunkból azt az elemet, melynek ID-ja ezzel megfelel. Amennyiben ez nem lenne meg, nem történik semmi.

```
router.delete('/:id',checkAuth, (req, res, next) => {  
  Post.deleteOne({_id: req.params.id}).then(result => {  
    res.status(201).json({  
      message: 'Post deleted successfully'  
    });  
  });  
});
```

A router.delete() metódus pedig az adatok adatbázisból való törléséért felel. A frontend oldalról kapott hír elem ID-ja alapján kikeresi az adatbázisból az ennek megfelelő adatot, és ezt a Post.deleteOne() függvény segítségével ki is törli.

Egyes metódusoknál második argumentumként megadtam egy checkAuth paramétert. Ennek célja, hogy a már korábban említett autorizációt nem csak frontend oldalon ellenőrizzük, hanem a backenden is, annak érdekében, hogy a rosszindulatú felhasználók számára semmilyen körülmények között se engedélyezzük a módosításokat.

```
module.exports = (req,res,next) => {  
  try {  
    const token = req.headers.authorization.split(' ')[1];  
    const decoded = jwt.verify(token,  
"ikhhokSecretPass_forTokenIdentification");  
    req.authData = {  
      identifier: decoded.identifier,  
      fullName: decoded.fullName,  
      email: decoded.email,  
      position: decoded.position,  
      permissions: decoded.permissions,  
      userId: decoded.userId  
    }  
  }  
  next();  
} catch(error) {  
  res.status(401).json( { message: 'Authentication failed!' } )}}
```

A `checkAuth` middleware a backend-en került implementálásra, és ennek segítségével a `JsonWebToken` csomaggal ellenőrizzük, hogy a `request`-ből kinyert token létezik-e a felhasználó számára (tehát be van jelentkezve), emellett ez a `jsonwebtoken.verify()` függvénnyel validálható-e. A verifikációhoz szükségünk van arra a token jelszóra is, melyet a bejelentkezéskor adunk hozzá a bejelentkező felhasználók újonnan létrehozott token-jeihez.

Azon route-okban, ahol a megszokott `input` és `textarea` bemeneti mezőkön kívül fájlfeltöltés mezők is megjelennek az űrlapjaikon, szükséges a kép- és fájlfeltöltés validálása és kivitelezése is. Ehhez a `multer` Node.js middleware-t használom.

A jegyzőkönyvek, beszámolók, pályázatok, határozatok, elnökség és kabinet típusú adatstruktúrák esetén a hozzájuk tartozó route fájlban került implementálása a mime type és a fájlkiterjesztés verifikációja.

```
const MIME_TYPE_MAP = {
  'image/png': 'png',
  'image/jpeg': 'jpg',
  'image/jpg': 'jpg',
}

const storage = multer.diskStorage({
  destination: (req, file, cb) => {
    const isValid = MIME_TYPE_MAP[file.mimetype];
    let error = new Error('Invalid type');
    if (isValid){
      error = null
    }
    cb(error, "images");
  },
  filename: (req, file, cb) => {
    const name = file.originalname.toLowerCase().split(' ').join('-');
    const extention = MIME_TYPE_MAP[file.mimetype];
    cb(null, name + '-' + Date.now() + '.' + extention);
  }
});
```

Ez a függvény ellenőrzi a feltöltött fájl kiterjesztését, mely Kabinet esetén csak `jpg`, `jpeg` vagy `png` lehet. Amennyiben a feltöltendő fájl ezen kritériumoknak nem felel meg, úgy a `storage` argumentum hozzáadásával az adott http metódusok is hibaüzenetet fognak visszaadni.

```
router.post('/', checkAuth, multer({storage: storage}).single('file'),
(req, res, next) => {
  /* ... */
})
```

3.5.2 Az app.js

Az app.js modul felel az applikációnk backend-jének működtetéséért. Alapesetben itt kerültek volna eltárolásra a route fájlokban lévő REST API hívások, azonban az áttekinthetőség érdekében ezek ki lettek szervezve külön fájlcsoportokba. Ahhoz, hogy ezen route-ok használhatóak legyenek a backend által, importálni kell őket az app.js modulba.

```
const hirekRoutes = require('./routes/hirek')
const ulesekRoutes = require('./routes/ulesek')
const palyazatokRoutes = require('./routes/palyazatok')
/* ... */
```

Az importálásokat követően ahhoz, hogy használhatóvá váljanak a router modulok, az app.js use() funkcionalitását használtam, melynek segítségével elérjük a külső middleware-eket.

```
/* HÍREK */
app.use('/api/hirek', hirekRoutes);

/* BIZOTTSÁGI ÜLÉSEK */
app.use('/api/ulesek', ulesekRoutes);

/* PÁLYÁZATOK */
app.use('/api/palyazatok', palyazatokRoutes);

/* ... */
```

Az app.js modulban került implementálásra az adatbázishoz való csatlakozás is. Ennek kialakításához a mongoose package-et használtam, melynek célja, hogy leegyszerűsítse a MongoDB használatát backend oldalon applikációnkban.

```
mongoose.connect("mongodb+srv://elnok:xtiV4hKL050qaLbM@cluster0.pz2bf.mongodb.net/ikhokDatabase?retryWrites=true&w=majority")
  .then(() => {
    console.log('Sikeres csatlakozás az adatbázishoz!')
  })
  .catch(() => {
    console.log('Sikertelen csatlakozás.');
```

A mongoose connect() metódusával kísérjük meg a csatlakozást az adatbázisunkhoz. Sikeres csatlakozás esetén a backend konzolján láthatjuk a „Sikeres csatlakozás az adatbázishoz” üzenetet, ellenkező esetben pedig a „Sikertelen csatlakozás”-t.

Sikertelen csatlakozásnak több oka is lehet. A MondoDB a cluster létrehozásakor kéri, hogy adjuk meg azon felhasználói IP címeket, akik jogosultak a csatlakozáshoz. Amennyiben IP címünk nem szerepel a listán, úgy azonnal elutasításra kerül csatlakozási kérelmünk. Másik

eshetőség, hogy a cluster-hez tartozó jelszavunk nem megfelelő (mely jelen esetben az „elnok” fiókhoz a „xtiV4hKL05OqaLbM” automatikusan generált jelszó).

A 2.2. fejezetben részletesebben elmagyaráztam a szoftver kitelepítésének menetét. A kitelepítés során nem szükséges az applikációt egy felhőszolgáltatásra deploy-olni, kezelhetjük külön a frontend és backend oldalt. Ugyanígy, amennyiben a programot lokálból indítjuk el fejlesztői módban, láthatjuk, hogy a frontend a `http://localhost:4200` – as portján, míg a backend a `http://localhost:3000` – es portján fut. Ilyen esetekben, amikor két különböző port között szeretnénk a kommunikációt és request hívásokat megvalósítani, a megfelelő Header-ek beállításának hiányában Cross-Origin Resource Sharing¹⁴ (CORS) hibát kapunk.

Ennek kiküszöbölése érdekében az `app.js` fájlban beállíthatjuk a Header-eket, mellyel engedélyezzük a request metódusokat.

```
app.use((req, res, next) => {
  res.setHeader('Access-Control-Allow-Origin', '*');
  res.setHeader('Access-Control-Allow-Headers', 'Origin, X-Requested-
  With, Content-Type, Accept, X-Auth-Token, Authorization');
  res.setHeader('Access-Control-Allow-Methods', 'GET, PATCH, POST, PUT,
  DELETE');
  next();
});
```

Fontos megjegyezni, hogy ezen funkcionalitásra nincs szükségünk, amennyiben a 2.2.1. bekezdés szerint AWS Elastic Beanstalk szolgáltatásra helyezzük ki az applikációnkat.

3.5.3 A `server.js`

A Node.js lehetőséget biztosít számunkra, hogy mindenféle külső szoftverek telepítése nélkül is létrehozassuk szerverünket. A szerver implementációját a `server.js` fájlban végezzük el, melyhez a Node.js-ben alapvetően megjelenő `http` csomagra is szükségünk lesz.

```
const server = http.createServer(app);
```

Célunk, hogy szerverünk egy megadott port-on figyelje a beérkező requesteket, és ennek megfelelően elvégezze a szükséges teendőket. Fejlesztői módban való indításkor az alapvető port a 3000.

¹⁴ <https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS/Errors>

Port létrehozásakor foglalkoznunk kell hibakezeléssel. A hibakezelés során figyelniünk kell a port felhasználtságára, mivel ha a portot már egy másik applikáció vagy szoftver foglalja, akkor nem használhatjuk ezt szerverünk gyanánt. Emellett, amennyiben olyan portot választunk magunknak, melyhez nem megfelelőek a jogosultságaink, úgy akkor is tájékoztatnunk kell a felhasználót a hibáról.

```
const onError = error => {
  if (error.syscall !== 'listen')
    throw error;

  const bind = typeof address === 'string' ? 'pipe ' + address : 'port ' +
+ port;
  switch (error.code) {
    case 'EACCES':
      console.error(bind + ' requires elevated privileges');
      process.exit(1);
      break;
    case 'EADDRINUSE':
      console.error(bind + ' is already in use');
      process.exit(1);
      break;
    default:
      throw error;
  }
};
```

3.5.4 Képek és fájlok tárolása

Egyes bejegyzési típusok létrehozása során fájlokat és képeket is feltölthetünk. Ezeket adatbázisunkban nem tudjuk fájlként tárolni, ezért egy egyszerű megoldás az adatokhoz tartozó fájlok hozzárendelése az elérési útvonalaiuk szerint.

Ez azt jelenti, hogy minden egyes feltöltés során a fájl vagy kép a backend mappánkban kerül eltárolásra, az adatbázisba való feltöltéskor pedig csak egy linket adunk meg a megfelelő kulcsnak értéként. A képek és fájlok az images és files mappában kerülnek elmentésre.

Ahogy az a korábbiakban kifejtettem, az elmentésük során multer package-et használok. Ennek következtében a POST, PUT és PATCH metódusok az alábbiak szerint kerültek implementálásra:

```
router.post('', checkAuth, multer({storage: storage}).single('file'),
(req, res, next) => {
  const url = req.protocol + '://' + req.get('host');
  const post = new Presidium({
```



```

    postType: req.body.postType,
    name: req.body.name,
    position: req.body.position,
    email: req.body.email,
    file: url + '/images/' + req.file.filename,
  });
  post.save().then( result => {
    res.status(201).json({
      message: 'Presidium added successfully',
      post: {
        _id: result._id,
        postType: result.postType,
        name: result.name,
        position: result.position,
        email: result.email,
        file: result.file
      }
    });
  });
});
});
});

```

Egy új Elnökség elem létrehozásakor láthatjuk, hogy a fájl elmentése egy URL, egy mappanév és egy fájlnev alapján történik.

Ahhoz, hogy a files és images mappába bármit is el tudjunk menteni, statikus hozzáférést kell biztosítanunk ezek szerkesztéséhez. Ehhez az app.js fájlban, csakúgy, mint ahogyan a routes fájlknál, az alábbi módon ezt meg is tehetjük:

```

app.use('/images', express.static(path.join('images')));
app.use('/files', express.static(path.join('files')));

```

A fájlok elnevezésének is kialakítottam egy konvenciót. A feltöltött fájloknak nem tartom meg az eredeti nevét, hanem a bejegyzés címéhez hozzáfűzöm a feltöltés pontos időpontját és az eredeti fájlkiterjesztést is.

```

filename: (req, file, cb) => {
  const name = file.originalname.toLowerCase().split(' ').join('-');
  const extention = MIME_TYPE_MAP[file.mimetype];
  cb(null, name + '-' + Date.now() + '.' + extention);
}

```

3.5.5 Az angular mappa és a package.json

Amennyiben a fejlesztői módban elindított alkalmazásunkkal úgy érezzük, hogy készen vagyunk, és szeretnénk kitelepíteni egy külső szerverre vagy felső szolgáltatásra, úgy az applikációt tudjuk deploy-olni.

Deployment során kétféle irányból közelíthetjük meg a kitelepítést. Eldönthetjük, hogy a front- és backendünk egy applikációként funkcionáljon, vagy külön-külön. Első esetében az `ng build` paranccsal hozzáadhatjuk a backend mappánkhoz az Angular app-unk optimalizált verzióját is, mely pillanattól fogva kitelepítéskor egy porton fognak futni – ez egyébként azt is jelenti, hogy az `app.js` fájlban található CORS error handling szekció nyugodtan ki is törölhető.

Ahhoz, hogy az `ng build` automatikusan elhelyezze az Angular mappát a backenden, az `app.js` fájlunkban ki kell kötnünk, mi lesz az alap html fájl, melyben az elemek (komponensek) dinamikusán lesznek majd elhelyezve.

```
app.use((req, res, next) => {  
  res.sendFile(path.join(__dirname, "angular", "index.html"));  
});
```

Kitelepítéskor nem elegendő csak a frontend hozzáadása a backendhez – a `package.json` segítségével megmondhatjuk, melyek a backend dependenciáink. Ezen funkcionalitás inkább abban az esetben szükséges, amikor a két applikációt külön szeretnénk feltölteni és deploy-olni, hiszen ekkor a backend mappánk teljesen leválasztódik a frontend `package.json` fájljáról.

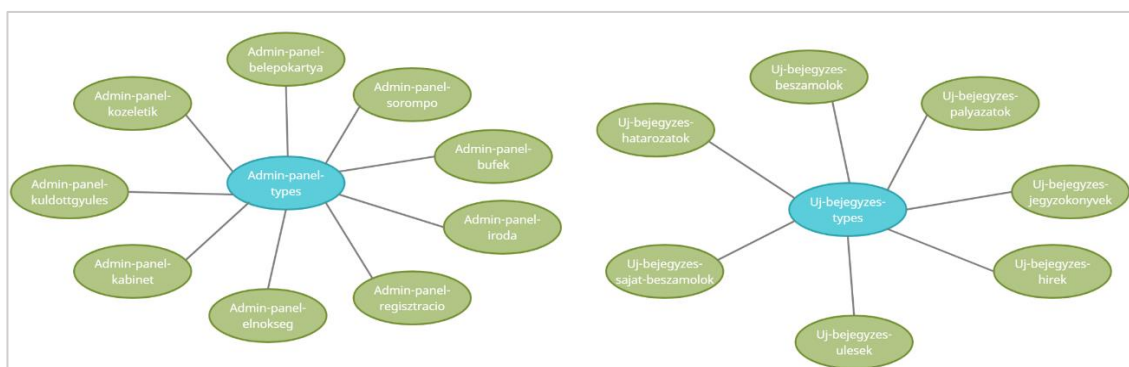
3.6 A frontend felépítése

Az Angular 12 alapú frontendünk komponensekből áll össze, a fő komponens pedig az `app`. Minden komponenshez tartozik egy Typescript, egy HTML és egy CSS fájl, melyek az adott komponens felépítéséért, funkcionalitásaiért és kinézetéért felelnek. A komponensekhez több esetben tartoznak modellek, melyek definiálják a használt adatstruktúra interfészét.

```
export interface Hirek {  
  _id: any;  
  postType: string;  
  title: string;  
  content: string;  
  date: string;  
}
```

Egy hír esetében a hozzátartozó interfészben láthatjuk, hogy nincs nagy különbség közte és az adatbázisban található sémája között – egyetlen szembetűnő eltérés az `_id`. Ennek oka, hogy az adatbázisba történő mentéskor a REST API végpontban elküldjük a backend számára az adatokat, azonban ott még nem határozzuk meg a bejegyzések ID-ját. Viszont abban az esetben, amikor lekérjük az összes hírt az adatbázisunkból, és ezeket szeretnénk megjeleníteni, szükséges elmentenünk a `postId`-kat is egyaránt, hiszen a későbbiekben ezek alapján tudunk majd szerkesztési requesteket küldeni a backend felé.

3.6.1 Komponens diagram



3.5.1. ábra – A komponens diagram első része

Az `uj-bejegyzes.component.html` és az `admin.component.html` tartalmazza az alapvető űrlapot, melyen kiválaszthatjuk a kívánt bejegyzés típusát. A kívánt elem kiválasztásának függvényében történik meg a form további elemeinek betöltése dinamikusan.

Abban az esetben, ha egy meglévő bejegyzést szeretnénk szerkeszteni, akkor a form automatikusan tudni fogja, mely bejegyzéstípus űrlapját kell használnia. Ez az `uj-bejegyzes.component.ts` és `admin.component.ts` fájlban került implementálásra az `OnInit` lifecycle hook-ban, itt nyeri ki a szükséges információkat az URL-ből.

```
ngOnInit(){
  if ((this.router.url).includes('szerkesztes')) {
    let arrayOfUrl = (this.router.url).split('/');
    let option = (arrayOfUrl[(arrayOfUrl.length - 2)]);
    this.selectedOption = option;
  }
}
```

Mivel új bejegyzés létrehozásakor nem minden felhasználó fér hozzá mindegyik bejegyzéstípushoz, ezért a bejelentkezéskor létrejövő token alapján ellenőrzésre kerül az autentikáció szintje. Az autentikáció szintje 1-től 5-ig terjedő skálán helyezkedik el, ahol az 1 az admin jogosultságot, az 5 pedig a küldötti jogosultságot jelenti.

```

<select
  [(ngModel)]="selectedOption"
  name="postType"
  required>
  <option value="hirek" *ngIf="getAuthLevel() <= 3">Hír</option>
  <option value="ulesek" *ngIf="getAuthLevel() <= 2">Bizottsági
  ülés</option>
  <option value="beszamolok" *ngIf="getAuthLevel() <= 2">Átláthatóság -
  Beszámoló</option>
  <option value="jegyzokonyvek" *ngIf="getAuthLevel() <= 2">Átláthatóság
  - Jegyzőkönyv</option>
  <option value="palyazatok" *ngIf="getAuthLevel() <= 2">Átláthatóság -
  Pályázat</option>
  <option value="hatarozatok" *ngIf="getAuthLevel() <= 2">Belső -
  Határozat</option>
  <option value="sajat" *ngIf="getAuthLevel() <= 3">Belső - Havi
  beszámoló</option>
</select>

```

Az Új bejegyzés komponensen és Admin komponensen belül találhatóak meg a bejegyzéstípusokhoz tartozó komponensek is. Mindegyik komponensnek a HTML fájljában a hozzá tartozó űrlap található meg.

Ezen komponensek Typescript fájljaiban kerültek implementálásra a bejegyzés létrehozási, illetve szerkesztési funkciók. Az űrlap mentésekor (onSubmit()) ellenőrizzük, hogy az adott bejegyzés új, vagy a felhasználó meglévőt szeretne szerkeszteni – ennek követéséért a mode publikus változó felel, melyet az NgOnInit lifecycle hook módosít minden alkalommal, amikor a komponens betöltésre kerül az oldalon.

```

ngOnInit() {
  this.route.paramMap.subscribe((paramMap: ParamMap) => {
    if (paramMap.has('id')) {
      this.mode = 'editPost';
      this.postId = paramMap.get('id');
      this.http.get<{message: string, post: any }>(BACKEND_URL + '/' +
this.postId)
        .subscribe((fetchData) => {
          this.editablePost = fetchData.post[0];
        });
    } else {
      this.mode = 'createNewPost';
      this.postId = '';
    }
  });
}

```

Minden új bejegyzést létrehozó komponens tartalmaz egy `addNewPost()` és egy `updatePost()` metódust. Első esetében létrehozunk egy új objektumot (hír létrehozása esetén egy `Hirek` típusút, ülések esetén egy `Ulesek` típusút és így tovább), melynek értékeiül az űrlapon megadott elemeket adjuk. Amint ez megtörtént, a `http POST` metódusának használatával ezt az új bejegyzést továbbítjuk a REST API végponton keresztül a backend felé. Sikeres feltöltés esetén az applikáció átirányítja a felhasználót az új bejegyzést tartalmazó oldalra.

Meglévő bejegyzés szerkesztésekor hasonlóan létrehozunk egy új objektumot, melyet a form-ban lévő adatokkal megtöltünk, azonban nem `POST` metódussal, hanem `PUT` vagy `PATCH` metódussal küldjük el az új bejegyzést a REST API endpoint-on keresztül.

```
addNewPost(form : NgForm) {
  const newPost : Iroda = {
    _id: null,
    postType: 'iroda',
    name: form.value.adminGroup.name,
    brief: form.value.adminGroup.brief,
    openHours: form.value.adminGroup.openHours,
  }

  return new Promise(resolve => {this.http.post<{ message: string,
postId: string }>(BACKEND_URL, newPost)
    .subscribe((data) => {
      const id = data.postId;
      newPost._id = id;
      resolve(data);
    })
  });
}
```

Az `addNewPost()` esetén láthatjuk, hogy a függvény egy `Promise`-sal tér vissza. Ennek oka, hogy amikor a felhasználó szeretné elmenteni bejegyzését, az `onSubmit()` aszinkron függvény a belső függvény befejeztét várja meg annak érdekében, hogy az oldal ne irányítsa át hamarabb a felhasználót az új bejegyzés létrehozásának oldaláról, minthogy a feltöltés sikeresen vagy sikertelenül végbemenjen az adatbázisunkba.

```
async onSubmit(form: NgForm) {
  if(this.mode === 'createNewPost') {
    await this.addNewPost(form);
  } else if (this.mode === 'editPost') {
    await this.updatePost(this.postId, form);
  }
  this.modalRef.hide();
  this.router.navigate(['/szolgaltatasok/iroda']);
}
```

Az Új bejegyzések komponenseiben, illetve az Admin panel komponenseiben ezen függvényeken kívül még megtalálható az `openModal()` és a `decline()` metódus is, melyek a modal figyelemfelhívó ablakok betöltéséért és működtetéséért felelnek.

Fontos megjegyezni, hogy nem mindegyik új bejegyzést sablon alapú űrlapokkal hoztam létre. Egyes esetekben, amikor képet vagy fájlt szeretnék hozzáfűzni a bejegyzéseinkhez, a reaktív megközelítést választottam. Ennek oka, hogy a `ReactiveFormsModule` segítségével egyszerűbben hozzacsatolhatóak fájlok az űrlaphoz, azok adatait egyszerűbben kinyerhetők, és szerkesztés esetén is dinamikusan betöltésre kerülnek a fájlok az űrlapba. Ezen komponensekben szükség volt a fájlfeltöltés függvényének implementálására is egyaránt, melyért az `onFilePicked()` metódus felelt.

```
onFilePicked(event: Event) {
  if(!(event.target as HTMLInputElement).files![0]) {
    return;
  }
  const file = (event.target as HTMLInputElement).files![0];
  this.form.patchValue({file: file});
  this.form.get('file').updateValueAndValidity();

  const reader = new FileReader();
  this.isFileUploaded = true;
  reader.readAsDataURL(file);
}
```

Azon bejegyzésekben, melyekhez hozzáfűzhető közzététel dátuma, automatikusan kitöltésre kerül a megfelelő input mező. Ezekért a `getDate()` metódusok felelnek.

3.6.3 Az bejegyzéseket tartalmazó komponensek

Ezen komponensek tartalmazzák a GET és DELETE http metódusok frontend megvalósítását. Előbbi célja, hogy az adatbázisban eltárolt elemeket a frontenden is elmentsük egy objektum tömbben, melynek segítségével ezt a HTML fájlban ki is rajzolhassuk az oldalra.

```
getPosts() {
  this.spinner.show();
  this.http.get<{message: string, posts: any }>(BACKEND_URL)
    .pipe(map(postData => {
      return postData.posts.map((post: any) => {
        return {
          _id: post._id,
          postType: post.postType,
```



```

        committee: post.committee,
        title: post.title,
        decisionDate: post.decisionDate,
        date: post.date,
        file: post.file
    }
});
}))
.subscribe((finalPosts) => {
    this.jegyzokonyvekObject = finalPosts;
    this.spinner.hide();
});
}

```

Látható, hogy a lekérdezett bejegyzések még nem megfelelő formátumban érkeznek a frontendre, ezért az RxJS¹⁵ map függvényének segítségével ezeket átalakíthatjuk. Amint ez megtörténik, egy privát objektum tömbben ezt elmentjük (mely a jegyzőkönyvek esetén a jegyzokonyvekObject), és a HTML fájlban az ngFor() segítségével kiírjuk.

```

deletePost(postId : string) {
    this.modalRef.hide();
    this.http.delete(BACKEND_URL + '/' + postId)
        .subscribe(() => {
            const updatedPost = this.jegyzokonyvekObject.filter(post =>
post._id !== postId);
            this.jegyzokonyvekObject = updatedPost;
        })
}

```

Bejegyzések törlése esetén a deletePost() metódust alkalmazzuk, melynek egyetlen paramétere a törlendő bejegyzés ID-ja. Ennek használatával a megadott bejegyzést a megfelelő REST API végponton az adatbázisból töröljük. Annak érdekében, hogy a frontend a törlést követően azonnal dinamikusan változzon, a privát jegyzokonyvekObject-ből a filter() függvénnyel kiszedjük a már nem létező elemeket.

Ezen működés az összes komponensben ugyanígy került implementálásra, a különbségek az objektumok kulcsai, és a privát objektum tömb elnevezése.

Azon oldalakon, ahol egy szűrő sáv is megtalálható, ott a filterObject() függvény segítségével modifikáljuk a frontenden kirajzolt adatokat. A függvény célja, hogy az adott típusú elemek közül csak azok jelenjenek meg, melyek bizottságai (committee) megegyezik a szűrő sávban kattintott gomb értékével.

¹⁵ <https://rxjs.dev/>

```
filterObject(data : Array<any>) {
    let copyOfObject = data;
    if (this.filterData !== null && this.filterData !== '') {
        copyOfObject = copyOfObject.filter(el => el.committee ==
this.filterData.id)
    }
    return copyOfObject;
}
```

A függvény működésén látható, hogy amennyiben a szűrő sávon még nem kattintottunk egyik elemre se, úgy nem történik változtatás az objektumtömbünkben.

Ahogy a 3.6.2. bekezdésben is említettem, itt is megjelennek a modal működtetéséhez szükséges `openModal()` és `decline()` függvények, melyek az ablakok betöltéséért és működtetéséért felelnek.

3.6.4 A bejelentkezés

A bejelentkezés funkcionalitásának biztosításához nem egy komponensben, hanem egy service-ben kerültek implementálásra a fontosabb metódusok. Ennek oka, hogy az itt lévő funkciókat más komponensekben is használunk, ezért cél, hogy ezek injektálhatóak lehessenek – az importálással egyszerűbben tudunk funkciókat megosztani komponensek között, mint a megszokott szülő-gyerek komponens kommunikációval, mint az `@Input` és az `@Output` dekorátorokkal.

Ebben a service-ben a `login()` és `logout()` függvényeket implementáltuk. A bejelentkezés során egy POST metódus segítségével elküldjük a backend részére a felhasználói token-t, a token lejáratí idejét, illetve a felhasználói adatokat a jelszó kivételével. A token a local storage-ben tároljuk, mely a bejelentkezést követően egy órán keresztül él. Ez azt jelenti, hogy amint letelik az egy óra, a bejelentkezés törlődik, ezáltal, ha valaki egyszer belépett az oldalra, nem marad az idők végeztéig bejelentkezve.

```
login(form: NgForm, template: TemplateRef<any>) {
    this.spinner.show();
    const userData: Felhasznalo = {
        _id: null,
        postType: 'auth',
        identifier: form.value.profileGroup.identifier,
        fullName: '',
        password: form.value.profileGroup.password,
        position: '',
        email: '',
        permissions: ''
    }
}
```

```

    this.http.post<{token: string, expiresIn: number, userId: string,
fullName: string, email: string, position: string, permissions:
string}>(BACKEND_URL, userData)
    .subscribe(response => {
        const token = response.token;
        this.token = token;
        if(token) {
            const expiresIn = response.expiresIn;
            this.setUserTimer(expiresIn);
            this.isAuthenticated = true;
            this.userInfo = {
                userId: response.userId,
                fullName: response.fullName,
                permissions: response.permissions,
                position: response.position,
                email: response.email
            }
            this.userAuthStatus.next(true);
            let expiration = this.getInTime(expiresIn);
            this.saveUserData(this.token, expiration, this.userInfo);
            this.router.navigate(['/']);
        }
    }, error => {
        this.userAuthStatus.next(false);
        this.openModal(template);
    })
    this.spinner.hide();
}

```

A bejelentkezés függvényen láthatjuk, hogy token-ünk elmentésre kerül egy privát változóban. Mivel ezt az információt nem feltétlenül érdemes a komponenseknek elküldeni, ezért helyette egy privát Subject-ben fogjuk tárolni az autentikáció jelenlegi állapotát. Minden bejelentkezés és kijelentkezés során ez a Subject egy true vagy false értéket fog kapni, amely módosulás azonnal jelzésre kerül a komponensek számára, és azonnal dinamikusan változik a UI és a hozzáférések.

A `saveUserData()` függvény célja, hogy a token-t, a token lejáratát és a felhasználói adatokat a jelszón kívül eltároljuk local storage-ben. Amikor kijelentkezik a felhasználó, ezt a tárolóegységet ki kell üríteni, melyet a `clearUserData()` metódus végez el. Ha a local storage-ben tárolt értékeket szeretnénk megkapni, azt a `getUserData()` végzi el számunkra.

Más komponensekben találhatóak olyan funkcionalitások, melyek nem elérhetőek nem bejelentkezett felhasználók számára. Ekkor ezen service injektálása szükséges az adott komponensbe, ahol ellenőrizhetjük az autentikáció állapotát és az autorizáció szintjét.

```

ngOnInit() {
  this.userData = this.userService.getUserInformation();
  this.authLevel =
this.userService.getUserAuthorizationLevel(this.userData);
  this.getPosts();
  this.isAuthenticated = this.userService.getIsAuthenticated();
  this.userAuthSubs =
this.userService.getUserStatusListener().subscribe(isAuthenticated => {
    this.isAuthenticated = isAuthenticated;
  });
}

```

Ennek megfelelően például így néz ki az Ulesek komponens `OnInit` lifecycle hook-jának tartalma. A `userService.getUserInformation()` függvény visszaadja a bejelentkezett felhasználó adatait a jelszaván kívül, a `userService.getUserAuthorizationLevel()` pedig az autorizációjának szintjét. A `userService.getIsAuthenticated()` függvénnyel megkapjuk, hogy valóban bejelentkezett-e felhasználónk, és a `userService.getUserStatusListener()` Subject-re feliratkozva pedig minden módosítás esetén azonnali frissítést biztosítunk.

Bár ezen funkciókkal leülthetünk egyes tevékenységeket, mint a bejegyzéstörölést és szerkesztést, azonban szükséges egyes route-ok védelme is nem megfelelő jogosultsággal rendelkező felhasználóktól. Az oldalak (komponensek) közötti navigációt az Angular Router segítségével oldottam meg, és egyes route-ok levédhetőek a `canActivate` kulccsal. A `canActivate` minden esetben egy guard-ot vár paraméterül, mely egy olyan injektálható osztály, ami a `CanActivate` interfészt implementálja.

```

@Injectable()
export class AdminGuard implements CanActivate{
  constructor(private userService: UserService, private router: Router){}

  canActivate(route: ActivatedRouteSnapshot, state: RouterStateSnapshot):
boolean | UrlTree | Observable<boolean | UrlTree> | Promise<boolean |
UrlTree> {
    const isAuthenticated = this.userService.getIsAuthenticated();
    const userData = this.userService.getUserInformation();
    if(!isAuthenticated || userData.permissions !== 'admin') {
      this.router.navigate(['/']);
    }
    return isAuthenticated && userData.permissions === 'admin';
  }
}

```

Jelen esetben az Admin guard kódrészlete látható. Azon route-ok, melyek csak és kizárólag az admin jogosultsággal rendelkező felhasználók számára lehetnek elérhetőek, azoknak a

canActivate kulcsához az AdminGuard osztályt kell megadni paraméterül a routing-ért felelős Typescript fájlban.

```
path: 'admin',  
component: AdminComponent,  
pathMatch: 'full',  
canActivate: [AdminGuard]
```

3.6.5 Egyedi függvények

Bár a komponensek tartalma sok esetben megegyezik, egyes esetekben olyan egyedi funkciókra volt szükség, melyek máshol nem jelennek meg.

Ülések létrehozásakor, amennyiben mindegyik input mező megfelelően lett kitöltve új bejegyzés létrehozásakor, első kattintásra automatikusan generálásra kerül az ülés tartalmának szövege és címe. Ezekért a generateMeetingTitle() és a generateMeetingContent() metódusok felelnek.

Azon oldalakon, ahol sorrendbe lehet állítani az adatokat, – mint például pályázatoknál, jegyzőkönyveknél és beszámolóknál – ott a név és dátum szerinti rendezésért az onSortByName() és az onSortByUploadDate() függvények felelnek.

Beszámolók összegzésekor figyelniük kell arra, hogy egy adott évhez és hónaphoz tartozó elem csak egyszer jelenleg meg a táblázatban. Ezért a filterReport() metódus felel, aminek célja, hogy egy új beszámolónak leadási idejét összeveti az eddigi leadási időkkal, és amennyiben nincs egyezés (tehát a felhasználó az első, aki az adott évben és hónapban először adott le beszámolót), akkor létrehoz egy új sort a táblában, ellenkező esetben nem.

4. TESZTELÉS

Annak érdekében, hogy meggyőződhesünk applikációnk megfelelő működéséről, mindenképpen érdemes a hibák ellenőrzése céljából teszteseteket létrehozunk. Ezen manuális tesztek egyésével ellenőriztem, és az alábbi tesztelési jegyzőkönyvben megjelenítettem az eredményeket.

A teszteléskor ellenőriztem a be nem jelentkezett felhasználók funkcióit, ugyanúgy, ahogy a bejelentkezettekét is mind az öt autorizációs szinten. Éppen ezért létrehoztam ideiglenes felhasználókat az alábbi adatokkal.

Autorizációs szint	Azonosító	Jelszó
1.	TestAdmin	testadmin
2.	TestElnokseg	testelnokseg
3.	TestKabinet	testkabinet
4.	TestJK	testjk
5.	TestKuldott	testkuldott

4.1 Tesztelési jegyzőkönyv

Bejelentkezés:

- ✓ Regisztrált azonosító és jelszó kombinációval a felhasználó be tud jelentkezni.
- ✓ Nem regisztrált azonosító és jelszó kombinációval a felhasználó nem tud bejelentkezni.
- ✓ Nem regisztrált azonosító és jelszó kombinációval egy hibaüzenet modal megjelenik a kijelzőn.

Navigációs sáv:

- ✓ Az alapvető oldalak megjelennek a sávon: Hírek, ülések, szervezet, átláthatóság, ösztöndíjak, szolgáltatások, tudástár.
- ✓ Be nem jelentkezett felhasználó esetén a sáv jobb oldalán egy bejelentkezés ikon jelenik meg.

- ✓ Bejelentkezett felhasználó esetén a sáv jobb oldalán egy kijelentkezés és egy hamburger ikon jelenik meg.
- ✓ A nyíllal rendelkező menüpontok a kurzorral való ráhelyezésre legördülnek.
- ✓ Minden menü- és almenü pont kattintható, átvezet a megfelelő oldalra.

Oldalmenü:

- ✓ Admin jogosultságú felhasználó esetén minden menüpont megjelenik.
- ✓ Elnökségi jogosultságú felhasználó esetén az admin panel kivételével minden menüpont megjelenik.
- ✓ Kabinet jogosultságú felhasználó esetén csak a profil, új bejegyzés, saját beszámolók, beszámolók összegzése és határozatok tára menüpont jelenik meg.
- ✓ Jegyzőkönyvvezető jogosultságú felhasználó esetén csak a profil, a beszámolók összegzése és a határozatok tára menüpont jelenik meg.
- ✓ Küldött jogosultságú felhasználó esetén csak a profil és a beszámolók összegzése menüpont jelenik meg.
- ✓ Bármely menüpont URL címből való elérésekor, amennyiben a felhasználó nem autentikált az oldal megtekintésére, visszairányítódik a főoldalra.

Hírek oldal:

- ✓ Az oldal megjelenik mind bejelentkezett, mind be nem jelentkezett felhasználók számára.
- ✓ Helyesen megjelennek a bejegyzések és a Hírek slider.
- ✓ Megfelelő jogosultsággal rendelkező felhasználóknak megjelenik a szerkesztés és törlés gomb.
- ✓ Törlés esetén egy megerősítő modal jelenik meg a kijelzőn, szerkesztés esetén pedig átirányításra kerül a felhasználó az új bejegyzés oldalra.

Ülések oldal:

- ✓ Az oldal megjelenik mind bejelentkezett, mind be nem jelentkezett felhasználók számára.
- ✓ Helyesen megjelennek a bejegyzések és a szűrő sáv.
- ✓ Megfelelő jogosultsággal rendelkező felhasználóknak megjelenik a szerkesztés és törlés gomb.
- ✓ A szűrő gombok megfelelően funkcionálnak, az ülések bejegyzései ennek megfelelően változnak.

- ✓ Törlés esetén egy megerősítő modal jelenik meg a kijelzőn, szerkesztés esetén pedig átirányításra kerül a felhasználó az új bejegyzés oldalra.

Szervezet oldalai:

- ✓ Az Elnökség, a Kabinet, a Küldöttgyűlés és a Bizottságok oldal megjelenik mind bejelentkezett, mind be nem jelentkezett felhasználók számára.
- ✓ Helyesen megjelennek a bejegyzések, a képek és a táblázatok.
- ✓ Megfelelő jogosultsággal rendelkező felhasználóknak megjelenik a szerkesztés és törlés gomb az Elnökség, a Kabinet és a Küldöttgyűlés oldalon.
- ✓ Törlés esetén egy megerősítő modal jelenik meg a kijelzőn, szerkesztés esetén pedig átirányításra kerül a felhasználó az admin panel oldalra.

Átláthatóság oldalai:

- ✓ A jegyzőkönyvek, a pályázatok, a beszámolók és a közéleti ösztöndíjak oldal megjelenik mind bejelentkezett, mind be nem jelentkezett felhasználók számára.
- ✓ Az alapszabály almenüpontra kattintva az oldal átirányít az ELTE HÖK oldalán található alapszabályra.
- ✓ Megfelelő jogosultsággal rendelkező felhasználóknak megjelenik a szerkesztés és törlés gomb.
- ✓ A szűrő és rendszerező gombok megfelelően funkcionálnak, a bejegyzések ennek megfelelően változnak.
- ✓ A Letöltés gombbal rendelkező elemek megfelelően funkcionálnak, a szükséges dokumentumok elérhetőek.
- ✓ Törlés esetén egy megerősítő modal jelenik meg a kijelzőn, szerkesztés esetén pedig átirányításra kerül a felhasználó az új bejegyzés oldalra, közéleti ösztöndíj esetén az admin panel oldalra.

Ösztöndíjak oldalai:

- ✓ Az alaptámogatás, a rendszeres és rendkívüli szociális támogatás, a rendszeres és kari ösztöndíjak és a tanulmányi ösztöndíj oldal megjelenik mind bejelentkezett, mind be nem jelentkezett felhasználók számára.
- ✓ Helyesen megjelennek az oldaltartalmak.
- ✓ Az oldalon található linkek elvezetnek az ELTE HÖK oldalára.

Szolgáltatások oldalai:

- ✓ A belépőkártya regisztráció, a sorompó regisztráció, az iroda nyitvatartások és az étkezők oldal megjelenik mind bejelentkezett, mind be nem jelentkezett felhasználók számára.
- ✓ Helyesen jelennek meg a bejegyzések és az oldaltartalmak.
- ✓ Megfelelő jogosultsággal rendelkező felhasználóknak megjelenik a szerkesztés és törlés gomb az iroda nyitvatartás és az étkezők oldalon.
- ✓ Törlés esetén egy megerősítő modal jelenik meg a kijelzőn, szerkesztés esetén pedig átirányításra kerül a felhasználó az admin panel oldalra.
- ✓ A belépőkártya adminisztráció és sorompó adminisztráció oldalon az űrlap leadható, leadást követően pedig megjelenik egy visszajelző modal.

Profil aloldal:

- ✓ Az oldal megjelenik minden bejelentkezett felhasználó számára.
- ✓ A felhasználói adatok form dinamikusan kitöltésre került a felhasználó nevével és e-mail címével.
- ✓ A felhasználói adatok mentése megtörténik a Mentés gombra kattintáskor.
- ✓ A jelszómódosítás megfelelő bemenetekkel megtörténik a Mentés gombra kattintáskor.
- ✓ A jelszómódosítás hibát dob, amennyiben a Jelenlegi jelszó mezőbe írt adat nem egyezik a felhasználó jelszavával.
- ✓ A jelszómódosítás hibát dob, amennyiben az Új jelszó mezőbe írt adat nem egyezik az Új jelszó ismét mezőbe írt adattal.
- ✓ Hiányos űrlap adatok esetén a mentés gomb nem elérhető.

Új bejegyzés aloldal:

- ✓ Az oldal megjelenik azon bejelentkezett felhasználók számára, akik legalább 3-as autorizációs szinttel rendelkeznek.
- ✓ A bejegyzés létrehozása form alapértéke az új hír.
- ✓ 1. és 2. autorizációs szinttel rendelkező felhasználóknak minden típus elérhető, 3. szintűeknek pedig csak az új hír és az új havi beszámoló opció.
- ✓ Bejegyzés típusának váltásakor az oldal bemeneti mezői dinamikusan változnak.
- ✓ A közzététel időpontjának alapértéke a jelenlegi serveridő.
- ✓ Bejegyzés közzétételekor megjelenik egy megerősítő modal, melynek elfogadásakor kerül csak feltöltésre a kívánt bejegyzés.

Saját beszámolók aloldal:

- ✓ Az oldal megjelenik azon bejelentkezett felhasználók számára, akik legalább 3-as autorizációs szinttel rendelkeznek.
- ✓ Helyesen jelennek meg a bejegyzések.
- ✓ A felhasználók számára megjelenik a szerkesztés és törlés gomb.
- ✓ Törlés esetén egy megerősítő modal jelenik meg a kijelzőn, szerkesztés esetén pedig átirányításra kerül a felhasználó az új bejegyzés oldalra.

Beszámolók összegzése aloldal:

- ✓ Az oldal megjelenik minden bejelentkezett felhasználó számára.
- ✓ Helyesen jelennek meg a bejegyzések.
- ✓ Minden év-hónap páros csak egyszer jelenik meg a beszámolók összegzése táblában.
- ✓ A megtekintés gombra kattintva az oldal alján megjelennek az arra az évre és hónapra leadott bejegyzések.
- ✓ A felhasználók számára megjelenik a szerkesztés és törlés gomb a beérkezett beszámolók táblában.
- ✓ Törlés esetén egy megerősítő modal jelenik meg a kijelzőn, szerkesztés esetén pedig átirányításra kerül a felhasználó az új bejegyzés oldalra.

Határozatok tára aloldal:

- ✓ Az oldal megjelenik azon bejelentkezett felhasználók számára, akik legalább 4-es autorizációs szinttel rendelkeznek.
- ✓ Helyesen jelennek meg a bejegyzések és a szűrőszám.
- ✓ Megfelelő jogosultsággal rendelkező felhasználóknak megjelenik a szerkesztés és törlés gomb.
- ✓ A szűrő gombok megfelelően funkcionálnak, a bejegyzések ennek megfelelően változnak.
- ✓ A Letöltés gombbal rendelkező elemek megfelelően funkcionálnak, a szükséges dokumentumok elérhetőek.
- ✓ Törlés esetén egy megerősítő modal jelenik meg a kijelzőn, szerkesztés esetén pedig átirányításra kerül a felhasználó az új bejegyzés oldalra.

Sorompó adminisztráció aloldal:

- ✓ Az oldal megjelenik azon bejelentkezett felhasználók számára, akik 1-es autorizációs szinttel rendelkeznek.

- ✓ Helyesen jelennek meg a sorompó jogosultság kérések és az aktív jogosultságok listája.
- ✓ Minden elemhez tartozik egy törlés és szerkesztés gomb, melyek megfelelően funkcionálnak.
- ✓ Törlés esetén egy megerősítő modal jelenik meg a kijelzőn, szerkesztés esetén pedig átirányításra kerül a felhasználó az admin panel oldalra.
- ✓ Az [M] betűre mozdítva egerünket megfelelően megjelenik a felhasználó indoklás.

Belépőkártya regisztráció:

- ✓ Az oldal megjelenik azon bejelentkezett felhasználók számára, akik 1-es autorizációs szinttel rendelkeznek.
- ✓ Helyesen jelennek meg a belépőkártya jogosultság kérések és az aktív jogosultságok listája.
- ✓ Az aktív jogosultságok listáján csak azon elemek jelennek meg, melyeknek az isApproved értéke true.
- ✓ Minden elemhez tartozik egy törlés és szerkesztés gomb, melyek megfelelően funkcionálnak.
- ✓ Törlés esetén egy megerősítő modal jelenik meg a kijelzőn, szerkesztés esetén pedig átirányításra kerül a felhasználó az admin panel oldalra.
- ✓ Az [M] betűre mozdítva egerünket megfelelően megjelenik a felhasználó indoklás.

Admin panel aloldal:

- ✓ Az oldal megjelenik azon bejelentkezett felhasználók számára, akik 1-es autorizációs szinttel rendelkeznek.
- ✓ Az oldal adatok szerkesztése form alapértéke az Elnökség.
- ✓ Bejegyzés típusának váltásakor az oldal bemeneti mezői dinamikusan változnak.
- ✓ Bejegyzés közzétételkor megjelenik egy megerősítő modal, melynek elfogadásakor kerül csak feltöltésre a kívánt bejegyzés.

5. FEJLESZTÉSI LEHETŐSÉGEK

Mint minden szoftvert, a HÖK új honlapját is tovább lehet fejleszteni új funkcionalitások implementálásával. Bár az oldal jelenleg is tökéletesen funkcionál, egyes megkönnyítő szolgáltatásokkal elérhetjük, hogy a használata még egyszerűbb lehessen, a tevékenységeket pedig tovább is automatizálhatjuk.

5.1 Új funkcionalitások

5.1.1 Levelezéssel való összekötés

Új bizottsági ülések létrehozásakor érdemes az érintett képviselők mihamarabbi tájékoztatása. Éppen ezért, amikor egy felhasználó létrehoz egy új ülés bejegyzést – azaz összehív egy bizottsági ülést az Önkormányzatban – automatikusan ki is küldheti az applikáció a bejegyzés szövegét a bizottság tagjai számára. Amennyiben a későbbiekben a HÖK úgy érzi, hogy mindenképpen szeretnének az ülésösszehívások mellé fájlokat is csatolni, ez esetben ezeket is kiküldhetik a honlap segítségével.

Hasonlóképpen működtethetünk levélküldési funkciókat a hallgatóság felé. Amennyiben regisztrálnak belépőkártyára vagy sorompó belépésre, automatikusan tájékoztathatjuk őket az igénylés menetéről és a folyamat jelenlegi állásáról.

5.1.2 Beszámolók összegzése, azok generálása dokumentumba

A beszámolók összegzésekor megtehetjük, hogy a feltöltött tartalmakat egy Word dokumentumba exportáljuk. Ennek célja, hogy a Hallgatói Önkormányzat által létrehozott közéleti ösztöndíj javaslatok alapesetben tartalmazzák a tisztségviselői beszámolókat is, melyet akár a honlap önmagának is generálhatna minden hónap végén.

5.2 Meglévő funkcionalitások fejlesztése

5.2.1 A Híroldal képnézegetője

Jelenleg a képnézegetőnk célja, hogy röviden elmagyarázzuk a hallgatóinkat a honlap működését, és bemutassuk az új funkcionalitásokat. Amennyiben erre az Önkormányzatnak a későbbiekben is szüksége lenne, megoldható, hogy tényleges képpel ellátott híreket is közzé tegyünk a képnézegetőben, és jobban felkeltsük a látogatók figyelmét a fontosabb eseményekre.

6. ÖSSZEFOGLALÁS

Összességében elmondható, hogy a Hallgatói Önkormányzat új oldala nem csak funkcionalitásaiban változik a jelenlegihez képest, hanem arculati és felépítésbeli javulásokat is mutathat. Ezeknek köszönhetően remélhetőleg egy nagyobb elérést fog biztosítani a HÖK számára, és több hallgató fogja gyakrabban látogatni az oldalt.

Véleményem szerint a felhasználói élmények sikeresen javítani tudtam a szakdolgozatom elkészítése során, és egy könnyebben átlátható, és megfelelően működő applikációt tehettem le az asztalra. Bár ez a hallgatóság számára nem egyértelmű, de a HÖK tisztségviselői rengeteg fontos feladatot látnak el a honlapon keresztül, melyeket sajnos megnehezítettek a jelenlegi honlap hiányosságai és nem működő funkciói. Mindezek mellett remélem, hogy az önkormányzat a későbbiekben egyszerűen és gyorsan tud majd az igényeinek megfelelően fejleszteni a honlapon, hiszen mint minden szoftvernek, ennek a fejlesztésének sincs még közel sem vége. A későbbi önkormányzatok mindig megpróbálják majd saját arcukra alakítani az oldalt, és ez a felépítésbeli változás remélem egyszerűen lehetővé is teszi számukra. Mindig lesznek újabb frontend technológiák, mindig lesznek gyorsabb és kezelhetőbb adatbázisok, de ezek módosítása innentől még egy másodéves számára is gyerekjáték lesz.

7. KÖSZÖNETNYILVÁNÍTÁS

Mindenki számára komoly megpróbáltatásnak bizonyosulnak az egyetemi évek, és én sem voltam különb. Nem csak a megszokott otthoni életből való kiszakadás, vagy a magasabb szintű tananyagok elsajátítása viseli meg a hallgatóságot, hanem a közeli barátok más irányokba való tovább állása is. Éppen ezért szeretném kifejezni hálámat mindazon személyeknek, akik tanulmányaim során végig támogattak, motiváltak, és a legnehezebb időszakokban is segítségüket nyújtották.

Szeretném megköszönni Ignéczi Orsolyának, hogy jókedvével és nagylelkűségével mindig mosolyt csalt az arcomra, és mindig tudtam, hogy van kire számítanom, ha bajban lennék. Köszönöm Gerely Viktor Andrásnak, hogy reális gondolkodásával és őszinte véleményével mindig a megfelelő út irányába terelt, és sosem hagyott egyedül a legnehezebb feladataim elvégzésekor sem. Köszönöm Fodor Zoltánnak, hogy minden nap ott volt mellettem, minden nap motivált az előrehaladásban, és mindig tudta, mikor van szükségem egy hosszú beszélgetésre, vagy friss levegőre.

Köszönöm Dr. Abonyi-Tóth Andornak, hogy szakmai segítségével és javaslataival elősegítette ennek a dolgozatnak az elkészülését, és bármikor fordulhattam hozzá a félév során.

Köszönöm a Hallgatói Önkormányzatnak, hogy éveken keresztül biztosította számomra második otthonomat, hogy mindig szeretettel fogadták minden elvetemült elképzelésemet, és mindig segítettek ezek megvalósításában is. Remélem ezzel a szakdolgozattal visszafizethetem minden boldog percet, minden őszinte mosolyt és minden fáradtságot, amit a HÖK adott nekem az elmúlt évek során.

Végezetül, köszönöm Feigl Eriknek, hogy mindig ösztönzött az élet minden aspektusában, végig támogatott mind egyetemi tanulmányaim, mind pedig szakmai fejlődésem során, és mindig arra ösztönzött, hogy a legjobbak közül is a legjobb akarjak lenni.

8. ÁBRAJEGYZÉK

1.3.1. ábra – Hírek főoldal látványterve.....	10
1.3.2. ábra – Ülések oldal látványterve.....	10
1.3.3. ábra – Szervezet oldal látványterve.....	11
1.3.4. ábra – Átláthatóság oldal látványterve	12
1.3.5. ábra – Szolgáltatások oldal látványterve.....	12
2.2.1. ábra – Új applikáció létrehozásának menete az AWS Elastic Beanstalk szolgáltatásban	14
2.2.2. ábra – AWS Elastic Beanstalk platform beállításai.....	15
2.2.3. ábra – AWS Elastic Beanstalk feltöltése beállítások.....	15
2.2.4. ábra – AWS Elastic Beanstalk elkészített környezet elérése, adatai	16
2.3.1. ábra – A tájékoztató csúszka a Hírek oldalon.....	18
2.3.2. ábra – Egy hír elem felépítése	19
2.3.3. ábra – Egy ülés elem felépítése	20
2.3.4. ábra – Az Elnökség almenü egyik eleme	21
2.3.5. ábra – A Bizottságok almenü egyik eleme.....	21
2.3.6. ábra – A Pályázatok almenü tartalma	21
2.3.7. ábra – A Közéleti ösztöndíjak almenü tartalma.....	22
2.3.8. ábra – Részlet az Egyszeri kari ösztöndíjak almenü tartalmából.....	22
2.3.9. ábra – A Sorompó regisztráció oldal tartalma	23
2.3.10. ábra – Az Iroda nyitvatartások almenü tartalmának részlete	24
2.4.1. ábra – A bejelentkezési felület tartalma	25
2.4.2. és 2.4.3. ábra – A felhasználói adatok módosítási panelja, illetve a jelszómódosítási panel.....	26
2.4.4. ábra – Az Új bejegyzés menüpont új beszámoló esetén	26
2.4.5. ábra – A Saját beszámolók oldal tartalmának részlete	27

2.4.6. ábra – A Beszámolók összegzése menüpont tartalma, a decemberi hónapra szűrve .	27
2.4.7. ábra – A Határozatok tára menüpont tartalmának egy részlete	28
2.4.8. ábra – A Sorompó regisztráció oldal tartalma	29
2.4.9. ábra – A Belépőkártya regisztráció tartalma.....	29
2.4.10. ábra – Az Admin panel kinézete új regisztráció létrehozása esetén	30
2.4.11. és 2.4.12. ábra – Egy ülés bejegyzés bejelentkezés után, nem megfelelő és megfelelő jogosultságokkal.....	32
3.1.1. ábra – Felhasználói esetdiagramm be nem jelentkezett felhasználó esetén	33
3.1.2. ábra – A felhasználói esetdiagram bejelentkezés esetén.....	34
3.5.1. ábra – A komponens diagram első része.....	51
3.5.2. ábra – A komponens diagram második része	52