**Eötvös Loránd Tudományegyetem**

**Informatikai Kar**

**Média-és Oktatásinformatika Tanszék**

**Lakóközösségek feladatait segítő webes alkalmazás**

**Témavezető:**

Tarcsi Ádám

egyetemi tanársegéd

**Szerző:**

Halda Tamás Tibor

Programtervező informatikus BSc.

Budapest, 2022

Tartalomjegyzék

[1. Bevezetés 4](#_Toc104755526)

[2. Felhasználói dokumentáció 5](#_Toc104755527)

[2.1 Az alkalmazás célközönsége 5](#_Toc104755528)

[2.2 Az alkalmazás rendszerkövetelményei 5](#_Toc104755529)

[2.3 Az alkalmazás használata 5](#_Toc104755530)

[2.3.1 Bejelentkezés 5](#_Toc104755531)

[2.3.2 Regisztráció 6](#_Toc104755532)

[2.3.3 Új hír felvitele 9](#_Toc104755533)

[2.3.4 Új szavazás felvitele 10](#_Toc104755534)

[2.3.5 Meglévő szavazáson új szavazat leadása 11](#_Toc104755535)

[2.3.6 Új lakógyűlés tábla felvitele (közösképviselő esetén) 12](#_Toc104755536)

[2.3.7 Gazdasági tábla felvitele (közösképviselő esetén) 12](#_Toc104755537)

[3. Fejlesztői dokumentáció 13](#_Toc104755538)

[3.1 Felhasznált technológiák az alkalmazás back-end oldalán 13](#_Toc104755539)

[3.1.1 MYSQL Server [1] 13](#_Toc104755540)

[3.1.2 NPM [2] 13](#_Toc104755541)

[3.1.3 Node Js [3] 13](#_Toc104755542)

[3.1.4 Express Js [4] 14](#_Toc104755543)

[3.2 Felhasznált technológiák az alkalmazás front-end oldalán 14](#_Toc104755544)

[3.2.1 Angular [5] 14](#_Toc104755545)

[3.2.2 JavaScript [7] 14](#_Toc104755546)

[3.2.3 TypeScript [8] 14](#_Toc104755547)

[3.2.4 HTML [9] 15](#_Toc104755548)

[3.2.5 CSS [10] 15](#_Toc104755549)

[3.3 Osztálydiagrammok, adatbázisszerkezet 15](#_Toc104755550)

[3.3.1 UML [11] 15](#_Toc104755551)

[3.3.1 Adatbázis seedelés [12] 16](#_Toc104755552)

[3.3.2 Adatbázisszerkezet, táblák 18](#_Toc104755553)

[3.3.2.1 „house” tábla: 18](#_Toc104755554)

[3.3.2.2 „news” tábla 19](#_Toc104755555)

[3.3.2.3 „poll” tábla 20](#_Toc104755556)

[3.3.2.4 „residental\_meeting” tábla 20](#_Toc104755557)

[3.3.2.5 „financial” tábla 21](#_Toc104755558)

[3.3.2.6 „users” tábla 22](#_Toc104755559)

[3.4 A fejlesztői környezet telepítése 23](#_Toc104755560)

[3.5 Fordítás és futtatás 25](#_Toc104755561)

[3.6 Az alkalmazás működése 27](#_Toc104755562)

[3.6.1 Backend 27](#_Toc104755563)

[3.6.2 Frontend 29](#_Toc104755564)

[3.6.2.1 Autentikáció (Auth) 30](#_Toc104755565)

[3.6.2.2 Komponensek (Components) 30](#_Toc104755566)

[3.6.2.3 Modellek (Models) 31](#_Toc104755567)

[3.6.2.4 Services (Szolgáltatások) 32](#_Toc104755568)

[3.6.2.5 Assets 33](#_Toc104755569)

[3.7 Tesztelés, tesztesetek 33](#_Toc104755570)

[4. Összefoglalás 37](#_Toc104755571)

[5. További fejlesztési lehetőségek 37](#_Toc104755572)

[6. Hivatkozások: 39](#_Toc104755573)

1. Bevezetés

Szakdolgozatom témájaként egy olyan alkalmazást választottam, mely a társasházakban élő lakóközösségek életét célzott megkönnyíteni. A társasházak a múlt század elejétől kezdve egyre nagyobb számban teszik elérhetővé a lakhatást Magyarországon. Megengedhetőbb alternatívát biztosítanak a lakosságnak azáltal, hogy több lakásra oszlik a ház, ezáltal annak terhei. Ezeket a lakókat egy un. közös képviselő köti össze, aki a felmerülő igényeket, problémákat kezeli. Ezeket rendszerint havi vagy kéthavi rendszerességgel megtartott lakógyűlések alkalmával intézik a lakóközösségek. Szakdolgozatom fő célja ezek a lakógyűlések valamilyen szintű helyettesítése azáltal, hogy térben és időben eltolva is lehetősége legyen mindkét félnek (lakók, illetve közös képviselő) kommunikálni a másikkal.

Az alkalmazás használatához a lakók, illetve a közös képviselő is regisztrálnak az kezdőképernyőből, majd ezután a két külön fél a maga lehetőségeit és információit láthatja. A lakók a saját lakcímükre vonatkozóan láthatják a különböző híreket, és szavazásokat melyekből ők maguk is hozhatnak létre a társasházukhoz. Emellett olvashatják azokat a korábbi lakógyűléseket, valamint ezek anyagi beszámolóit. A közös képviselők feladatköréből adódóan az adott házhoz, vagy házakhoz adhatnak hozzá a lakó számára olvasható lakógyűlési jegyzetet, illetve ehhez tartozó anyagi beszámolókat.

A szakdolgozatomhoz használt technológiákat igyekeztem az egyetemi tárgyakon megismertekhez leginkább igazítani, ezalapján nagyvonalakban az alkalmazás egy MS SQL szervert használ melyet egy Node Js backend illetve egy Angular frontend egészít ki. Ezeket részletesen a fejlesztői dokumentációban fejtem ki.

Ez a dokumentum tartalmaz egy felhasználói- illetve egy fejlesztői dokumentációt. Előbbiben az alkalmazás használatát, míg utóbbiban a felhasznált technológiákat, az alkalmazás felépítését, illetve a tesztelés eredményeit mutatom be.

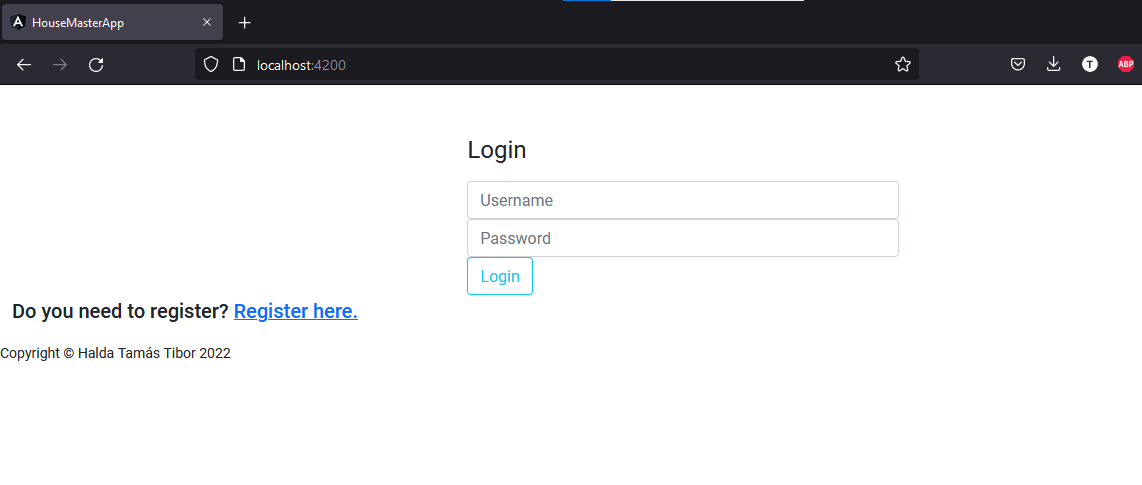
1. Felhasználói dokumentáció
   1. Az alkalmazás célközönsége

Az alkalmazás két felhasználói réteget céloz meg. Az egyikük azon lakók, akik a társasházban élnek, másik pedig a hozzájuk tartozó közös képviselők.

* 1. Az alkalmazás rendszerkövetelményei

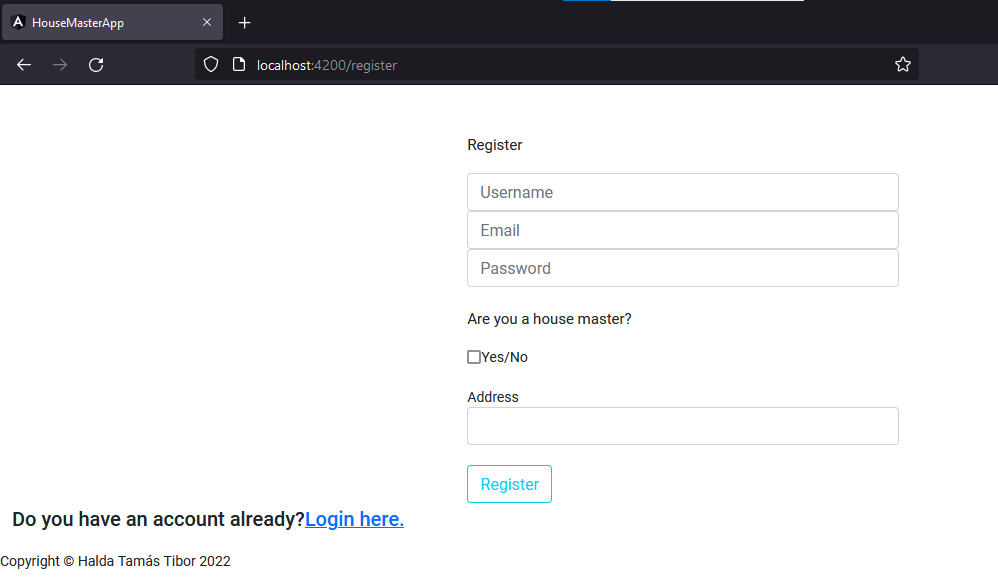
Az alkalmazás frontendje bármely böngészőből elérhető és használható, a backend pedig a legtöbb mai szervert tekintve rendkívül kevés erőforrással is elboldogul. Az alkalmazás tárhelyigénye is kevés ugyanis az adatbázisban csak szöveges bejegyzések találhatóak. Az alkalmazás gondmentes használatához valamilyen fajta CORS (Cross origin resource sharing) bővítmény használata ajánlott.

* 1. Az alkalmazás használata
     1. Bejelentkezés

Az alkalmazást elindítva a bejelentkezést kezelő felület jelenik meg. Ezen keresztül tudnak a felhasználók belépni a webes alkalmazásba. Illetve ha még nem rendelkeznek fiókkal, az itt feltűntetett „Register here” link átirányítja őket a megfelelő oldalra. Ha rendelkeznek fiókkal, az adataikat bírva a „Login” gombbal léphetnek be a dashboardra.

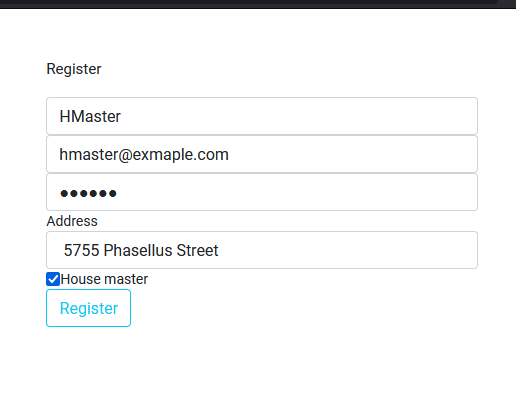
1. ábra bejelentkezés

* + 1. Regisztráció

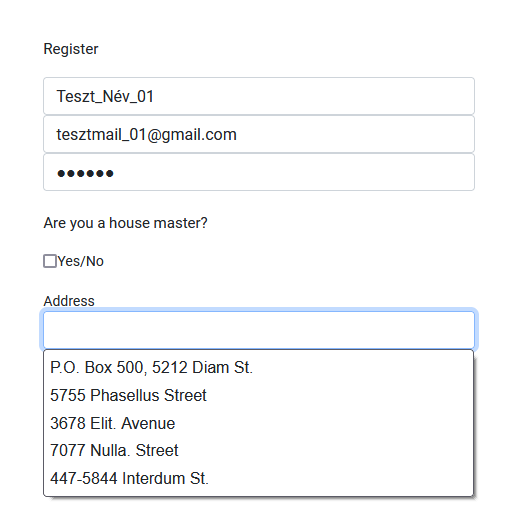
Ahogy az előző pontban már kitértem rá a felhasználók fiókkal kell, hogy rendelkezzenek az alkalmazás használatához. Ezt a fiókot ezen az oldalon hozhatják létre. Létrehozás után pedig a rendszer visszairányítja őket a bejelentkezést intéző oldalra, és onnan folytathatják a szoftver használatát a bejelentkezést követően. Emellett ha a felhasználó véletlen jutott volna el ide, van lehetősége vissza navigálni a Login oldalra a bal alul található „Login here” linken keresztül.

2. ábra Regisztrációs oldal

Az oldal lehetőséget biztosít a közösképviselők számára megjelölni a posztjukat, ezáltal azt is bevinni a rendszerbe.

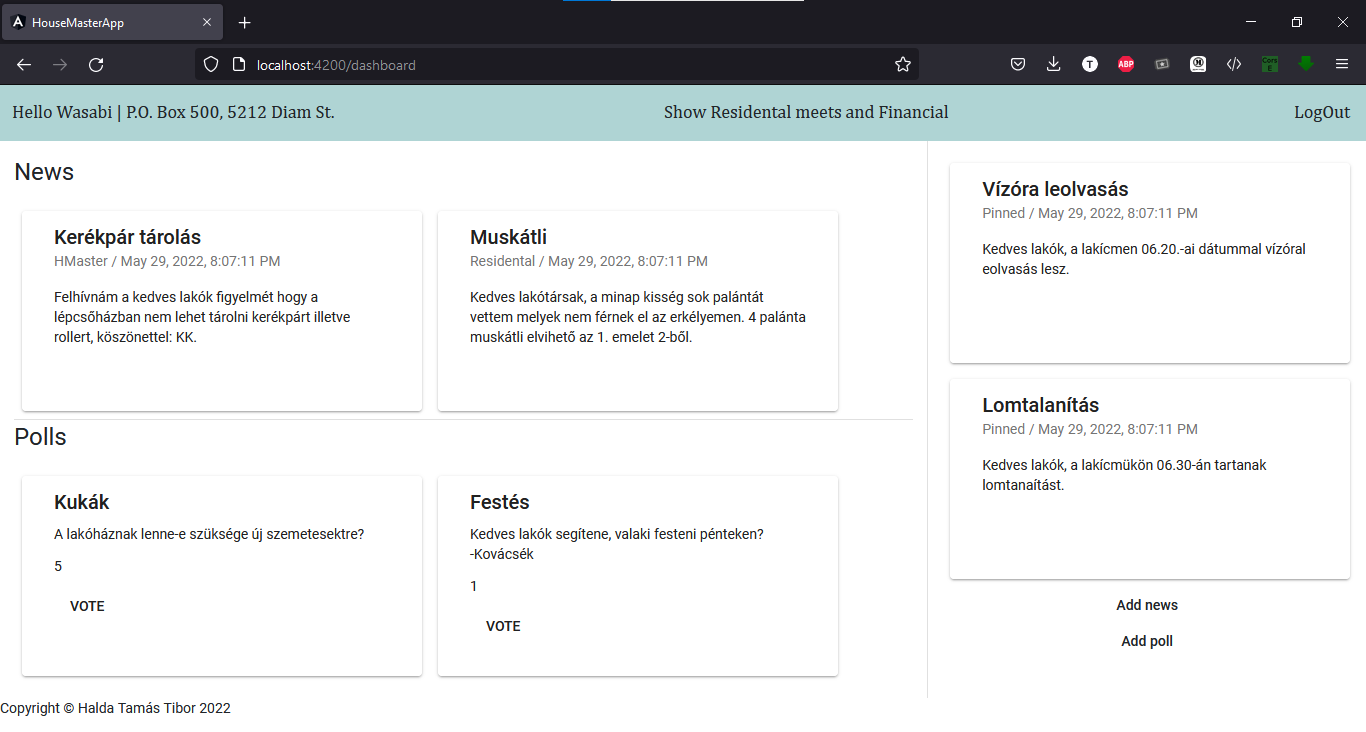


3. ábra Regisztráció közösképviselőként



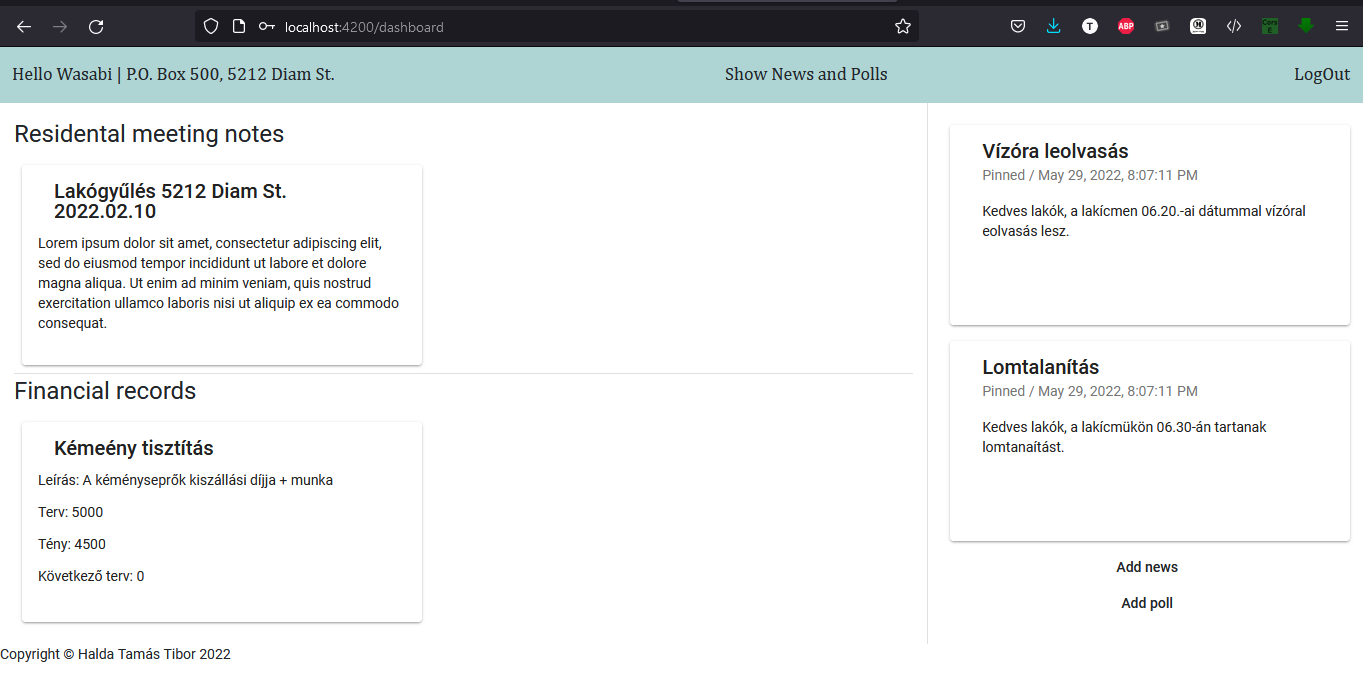
4. ábra Regisztráció lakosként

* + 1. Dashboard

A felhasználó legyen az lakó vagy közös képviselő, a bejelentkezés után a dashboardot fogja látni. Itt láthatja lakóként a saját lakcímének szóló híreket illetve szavazásokat. 

5. ábra A felhasználó keződképernyője a hírek illetve szavazások oldalon

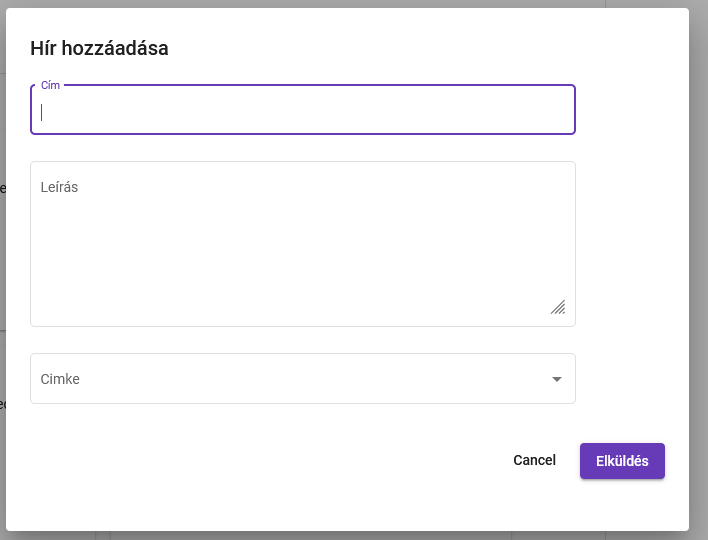
A kezdőképernyő több részből épül fel. A középső részen kettő szekciót találunk. Ezek között a fenti szekciót a hírek az alsót pedig a szavazások töltik ki. Ezeket minden lakó a saját lakcímére leszűrve kapja meg. Emellett található a jobboldalon az oldalsó sáv. Ezt a sávot szintén két elem alkotja. A felső azok az üzenetek, melyeket létrehozásukkor „Pinned” azaz kitűzötteknek készítettek. Ezek, mint ahogy a nevük is sugallja azért foglalnak el különleges helyet a kezőképernyőn, hogy ne tudjanak „elkallódni” a többi között. Ezek alatt foglalnak helyet a létrehozáshoz szükséges gombok, melyek az alábbi képen egy lakó jogkörét tekintve csak a hírek illetve a szavazatok. A közösképviselő esetében ez kiegészül négyre, a lakógyűlés tábla, illetve a gazdasági táblák felvitelének lehetőségével.



6. ábra A kezdőképernyő a lakógyűlés és gazdasági erkord oldalon

* + 1. Új hír felvitele

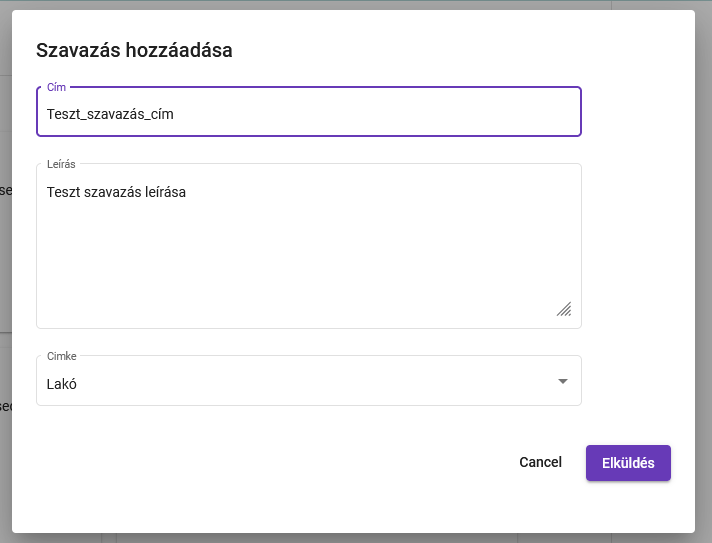
A lakóknak illetve a közösképviselőknek is van lehetőségük híreket felvinni a jobboldaon található sávból. Az „Add news’ gombra kattintva az alábbi pop-up ablak jelenik meg, melyben kitöltve az adatokat és a „Elküldés” gombra kattintva a rendszer azt továbbítja az adatbázis felé. A létrehozott hír az oldal frissítése után látható a Dashboardon.



7. ábra Hír hozzáadása tábla

* + 1. Új szavazás felvitele

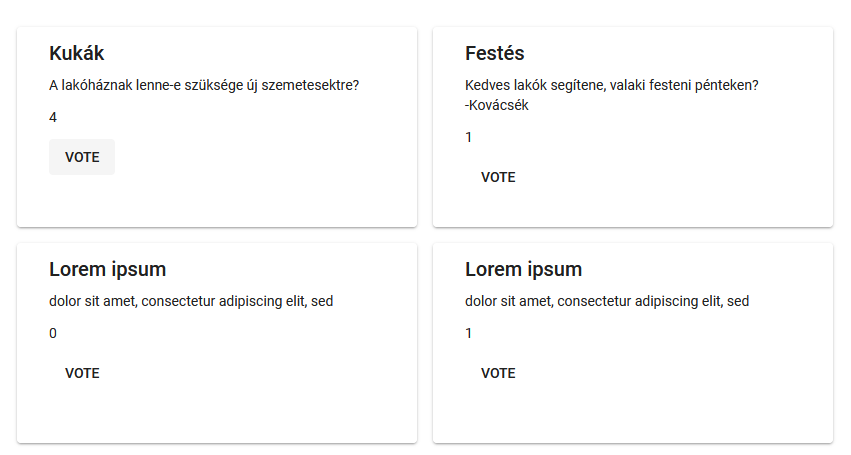
A hírekhez hasonló módon a szavazásokra is a jobb oldali sávból van lehetőség hozzáadni. Az „Add poll” gombra kattintva egy párbeszédablak jelenik meg, melyben a mezőket kitöltve és az „Elküldés” gombra kattintva a rendszerbe jut a szavazás. Persze ha a felhasználó meggondolná magát abban az esetben a „Cancel” gombra kattintva a párbeszédablak eltűnik és a folyamatot megszakíthatja.



8. ábra Szavazás hozzáadása

* + 1. Meglévő szavazáson új szavazat leadása

A dashboardon található kártyák közül a szavazatoké lehetőséget ad felhasználói interakcióra. Erre a kártyára való klikkeléssel a felhasználók tudják leadni a voksukat az adott kártyán látható kérdésre vagy szavazatra.

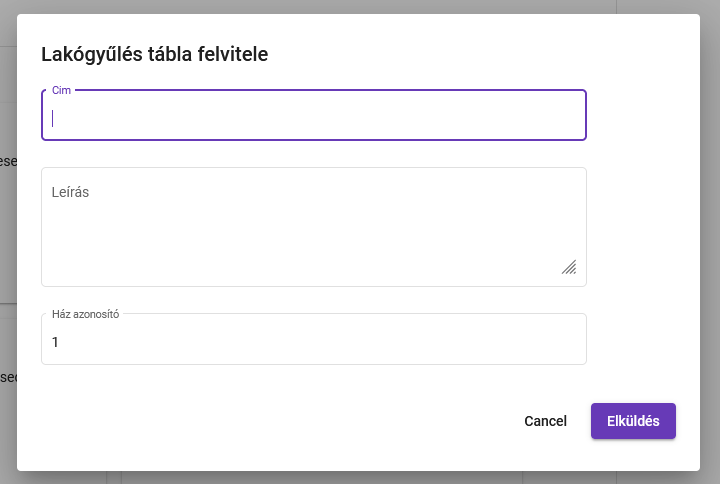


9. ábra A szavazat kértyák és szavazat-számlálóik

Ezen kártyákon a „Vote” gombbal tudják leadni a felhasználók a szavazatukat, mely ezzel egyidőben a szerveren is rögzül, tehát ha az oldalról azonnal el is navigálna, vagy frissítené sem történik semmi ami ne lett volna mentve.

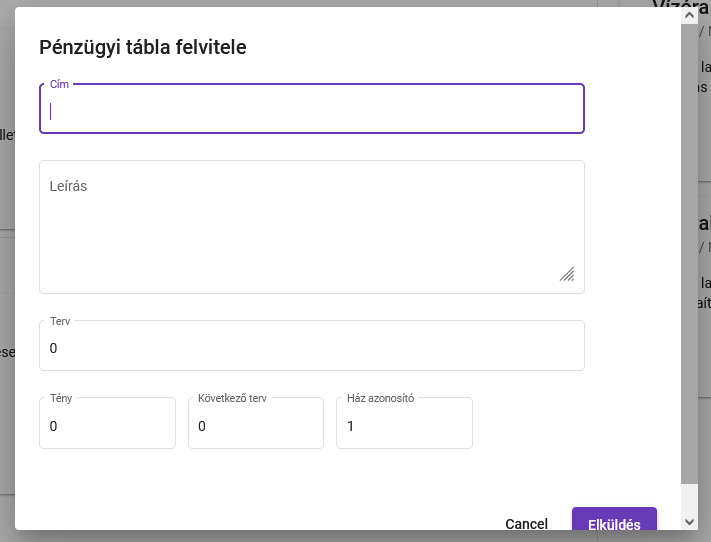
* + 1. Új lakógyűlés tábla felvitele (közösképviselő esetén)

A lakógyűlés tábla reprezentálja a lakóközösség egy lakógyűlés jegyzőkönyvét. Ide a közösképviselő feljegyezheti az ott elhangzott témákat, valamint hogy milyen konklúzióra jutottak a megbeszélés során. Ezek a táblák mindekét réteg számára olvashatóak felvitelüket viszont csak egy közösképviselő teheti meg. A tábla felvitele az előzőkhez hasonlóan a jobboldali sávból tehető meg. Az „Add meeting” gombra kattintva az alábbi párbeszédablak jelenik meg, melyet kitölthetünk és továbbíthatunk a szerverre az „Elküldés” gombbal. A többi adattípushoz hasonlóan, ha elvetnénk a bevitt értékeket abban az esetben a „Cancel” gomb bezárja a párbeszédablakot.



10. ábra Lakógyűlés tábla felvitele

* + 1. Gazdasági tábla felvitele (közösképviselő esetén)

A gazdasági táblák (vagy kártyák) mutatják meg egy lakógyűlés anyagi vonzatait. Gazdasági tábla felvitelét is csak a közösképviselő tudja megtenni, szintén a jobboldalon lévő oldalsávból, az „Add financial” gombbal. A folyamatot befejezni az „Elküldés”, megszakítani pedig a „Cancel” gombbal tudja. 

11. ábra Pénzügyi tábla felvitele

1. Fejlesztői dokumentáció
   1. Felhasznált technológiák az alkalmazás back-end oldalán
      1. MYSQL Server [1]

A MySQL egy többfelhasználós, többszálú, SQL-alapú relációs adatbázis-kezelő szerver. A szoftver eredeti fejlesztője a svéd MySQL AB cég, amely kettős licenceléssel tette elérhetővé a MySQL-t; választható módon vagy a GPL szabad szoftver licenc, vagy egy zárt (tulajdonosi) licenc érvényes a felhasználásra. 2008 januárjában a Sun felvásárolta 800 millió dollárért a céget. 2010. január 27-én a Sunt felvásárolta az Oracle Corporation, így a MySQL is az Oracle tulajdonába került

* + 1. NPM [2]

Az NPM (teljes nevén Node Package Manager) egy csomagkezelő a JavaScript programozási nyelvhez. A Node Js alapértelmezett csomagkezelő rendszere, mely egy parancssori felületből, egy online adatbázisból mely nyilvános illetve privát csomagokat is tartalmaz. Segítségével tudunk telepíteni, frissíteni, törölni vagy konfigurálni megfelelő csomagokat az operációs rendszerre, melyekkel kiegészíthetjük alkalmazásainkat.

* + 1. Node Js [3]

A Node Js egy JavaScript futtatási környezet, mely lehetővé teszi, hogy böngészőkön kívül is futtathatók legyenek a nyelven írt programok, jellemzően backenden. A környezet szerverek készítésére lett optimalizálva, melyeket JavaScriptben írható, eseményvezérlet aszinkron I/O-val skálázható terhelést szem előtt tartva.

* + 1. Express Js [4]

Vagy csak simán Express, egy nyílt forráskódu, back end webes alkalmazás keretrendszer Node Js-hez. Arra tervezték, hogy webes alkalmazások API-pontjait lehessen vele létrehozni, melyeken keresztül tud kommunikálni a szerver illetve a kliens.

* 1. Felhasznált technológiák az alkalmazás front-end oldalán
     1. Angular [5]

Az Angular (mely nem tévesztendő össze az Angular JS-el) egy ingyenes, nyílt forráskódu, TypeScript alapú web-alkalmazás keretrendszer melyet a Google fejlesztett ki. Egyik fő könyvtára az Angular Material, melyet felhasználtam szakdolgozatomban. Ez a könyvtár elsősorban felhasználói felületet szabja személyre a Google által 2014-ben megszabott „Material design”[6] minta szerint.

* + 1. JavaScript [7]

A JavaScript, vagy röviden Js, egy objektum orientált programozási nyelv, mely alapvető technológiája a Wolrd Wide Webnek a HTML illetve a CSS mellett. Elsősorban a kliens oldali webprogramozásra fejlesztették, a benne írt programkódot a böngészők dedikált JavaScript engine-je futtat a felhasználó készülékén. Ezek a JavaScript engine-ek eredetileg csak a böngészőkbe építve voltak elérhetőek, de mára több szerver oldali illetve külön futtatókörnyezet is rendelkezésre áll. A leggyakrabban használt ilyen futtatói környezet a nyelvhez a Node Js.

* + 1. TypeScript [8]

A TypeScript egy programozási nyelv, melyet a Microsoft fejlesztett ki illetve tart karban. A TypeScript fordítóprogramja a TSC, mely szintén TypeScriptben lett megírva. Egy szigorú szintaktikai szuperszet a JavaScript-en belül mely opcionális típusossággal bővíti a nyelvet. Nagy alkalmazások fejlesztéséhez hozták létre, és használható mind kliens, mind szerver oldali alkalmazások implementálásához.

* + 1. HTML [9]

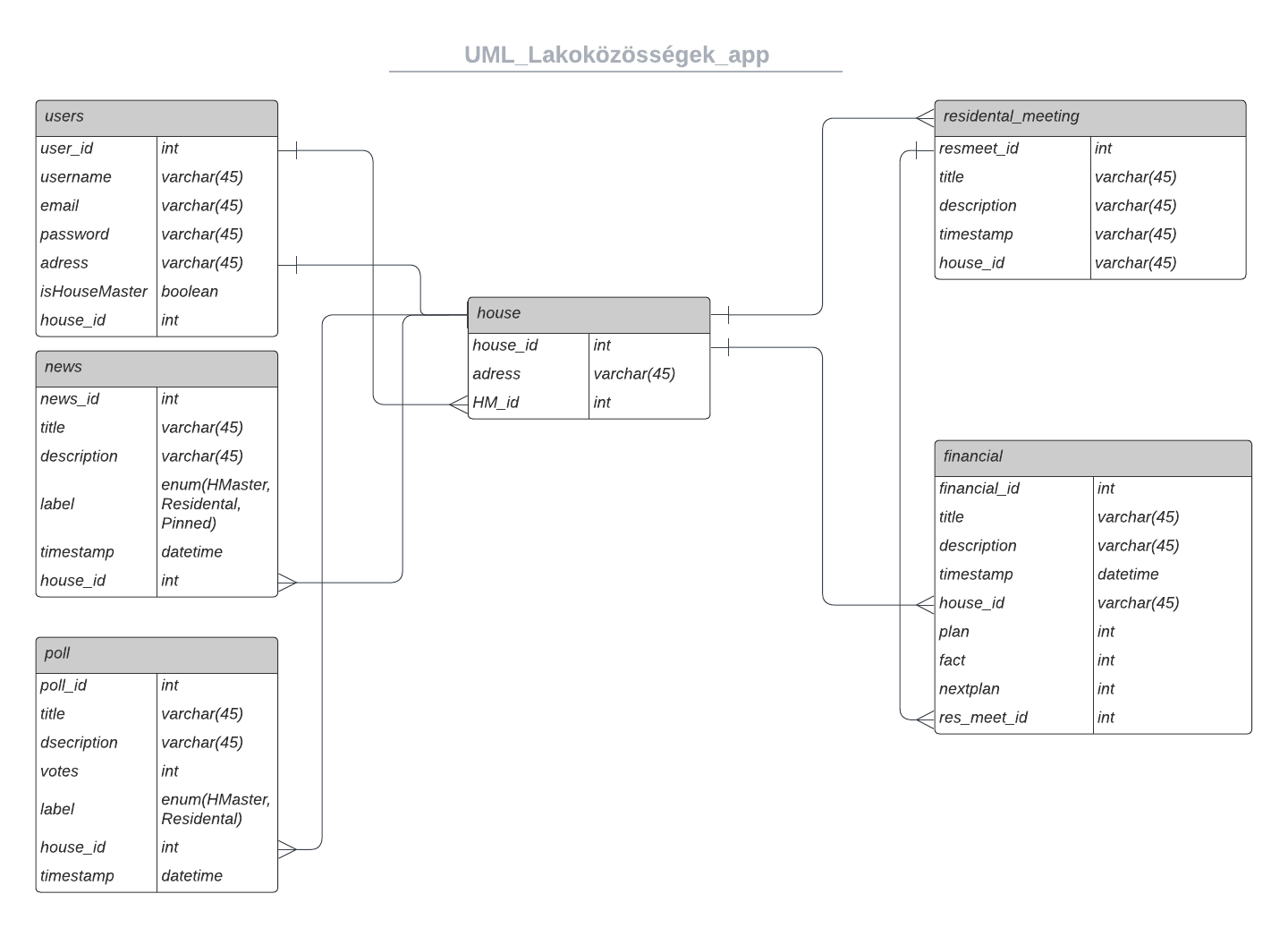
A HTML vagy teljes nevén HyperText Markup Language egy leíró nyelv melyet weboldalak készítéséhez fejlesztettek ki és mára már internetes szabvánnyá vált a W3C (World Wide Web Consortium) támogatásával.

* + 1. CSS [10]

Cascading Style Sheets vagy ismertebb nevén CSS, egy stílusleíró nyelv, mely HTML, vagy XHTML típusú, strukturált dokumentumok megjelenítését írja le. Ezenkívül használható bármilyen XML alapú dokumentum stílusának leírására is, mint például az SVG, XUL stb. A CSS specifikációját a World Wide Web Consortium felügyeli.

* 1. Osztálydiagrammok, adatbázisszerkezet
     1. UML [11]

Az UML (Unified Modeling Language) szabványos, általános célú modellező nyelv, üzleti elemzők, rendszertervezők, szoftvermérnökök számára. Az UML egy gyakorlati, objektum orientált modellező megoldás, nagyméretű programrendszerek modelljeinek vizuális megjelenítésére alkalmas eszköz. Az alábbi ábrát a Lucidchart webes alkalmazással készítettem el.

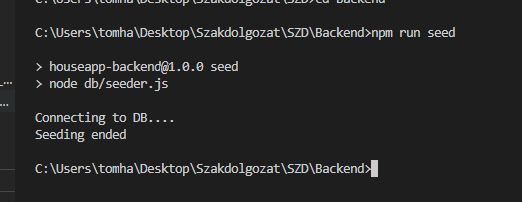


12. ábra Az alkalmazás UML diagramja

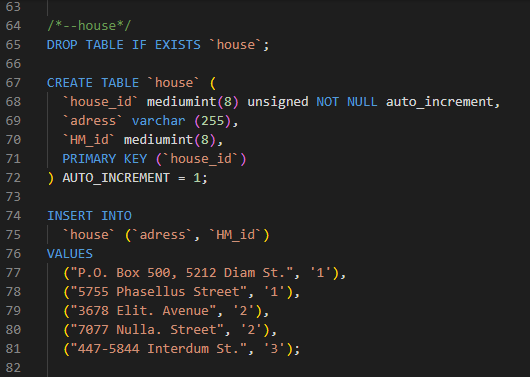
* + 1. Adatbázis seedelés [12]

Az adatbázis seedelése egy folyamat mely során egy scipttel feltöltjük az adatbázist teszt értékekkel. Ez a folyamat lehetőséget ad gyorsítani a tesztelési fázist, amelyben az adtabázisban lévő elemeket olvasnánk ki, akár a backend endpointot, akár a frontend megjelenítést vizsgáljuk. Szakdolgozatomnak egy olyan seedert írtam mely npm paranccsal futtatható, ezáltal nagyban megkönnyíti a használatát. A seeder működése a következőképp zajlik: először a szerver kapcsolatot létesít a MS SQL szerverrel, majd belépés után lefuttatja azt az SQL Query-t melyet a seeder.js átad neki. Ez a query először létrehozza a táblákat a megadott adattagokkal és elsődleges kulcsokkal, majd feltölti az adott táblát a kívánt teszt-adatokkal. Abban az esetben ha hibára futna az adatbáziskezelő ( pl nem megfelelő adattípust szúrna be a táblákba, esetleg nem megfelelő számú argumentum található a beszúrásban) a seeder jelzi és a hibával kilép, majd a hibát kiírja a konzolra és felhívja a figyelmet arra hogy a seeder melyik szakaszán, illetve milyen hibakóddal jelent meg a hiba.

Abban az esetben, ha hiba nem lép fel a táblák illetve az adatok létrehozásában, a folyamat végén tájékoztat róla hogy sikeres volt a seedelés, majd az adatbáziskapcsolatot lezárja.



13. ábra A seeder futtatása



14. ábra a "house" tábla létrehozása és seedelése

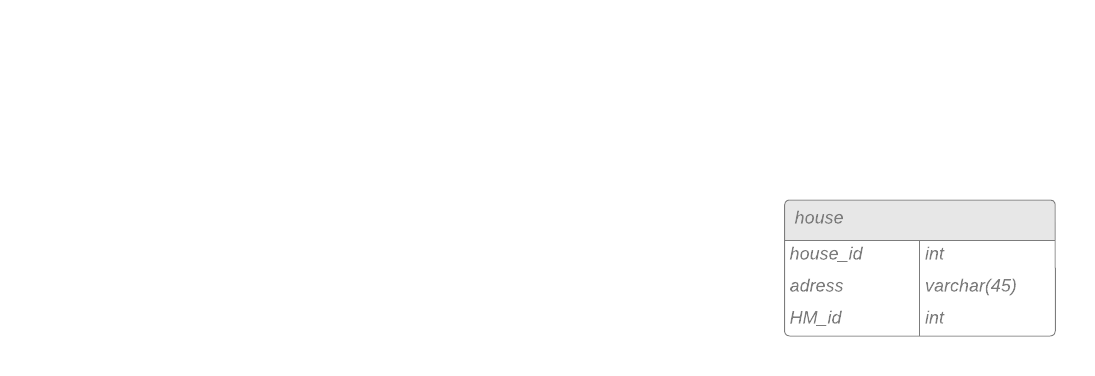
A seedeléshez használt adatokat nem teljes kezüen én írtam, nagyban segítségemre volt a generaldata.com- oldal [15] ,mely révén SQL insert függvényeket lehetett könnyebben generálni az általános adattípusok és táblanevek mellé.

* + 1. Adatbázisszerkezet, táblák

Szakdolgozatom tervezését és megvalósítását is az adatbázisszerkezet tervezésével kezdtem. Elsősorban azt tartottam fontosnak, hogy egy olyan adatbázis struktúrában tároljam az adatokat, melyek teret adnak a program későbbi módosításainak, annak függvényében, hogy milyen gyorsan haladok az implementálással. A következő nagy fejezetben részletesen kifejtem a szakdolgozat lehetséges további fejlesztésit, amik viszont megalapozzák ezeket azok leginkább az adattáblákra, illetve szerkezeteikre vezethetők vissza. A táblák szerkezete leginkább a házhoz, mint központi elemhez vezethető vissza. A házhoz tartozó egységek mind tartalmaznak „TIMESTAMP” típusú adattagot, melyeket az inicializálásukkor az adatbáziskezelő hoz létre, az azonosításra szoruló „\*táblanév\*\_id”-hoz hasonlóan. Ezek az elemek az egy házhoz tartozó „news”, „poll” , illetve az egymást kiegészítő „residental\_meeting” és „financial” táblák. A továbbiakban részletesen ismertetem az adatbázis ezen tábláit is, illetve azok tartalmát.

* + - 1. „house” tábla:

Magát a házat reprezentálja, illetve segít összekapcsolni minden táblát. Az alkalmazás több eltérő lakcímet képes kezelni, illetve akár egy közös képviselő számára támogat több lakcímet.



15. ábra a "house" tábla UML diagramja

|  |  |
| --- | --- |
| Adattag | Leírás |
| house\_id | egyedi azonosítója a ház példánynak |
| adress | a ház címe |
| HM\_id | a ház közösképviselőjének user\_id-je |

* + - 1. „news” tábla

Egy adott lakcímhez tartozó hírt vagy akár közleményt képvisel. A „label” enum típusú adattagja alapján elkülönül, hogy az alkalmazás melyik részén jelenik meg, emellett egy rövidebb címe illetve egy hosszabb leírással rendelkezik.

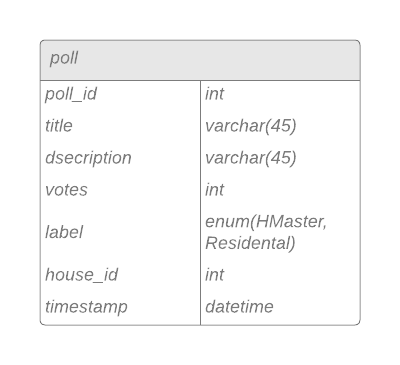


16. ábra a "news" tábla UML diagramja

|  |  |
| --- | --- |
| Adattag | Leírás |
| news\_id | egyed azonosítója a híreknek |
| title | a hír címe |
| description | a hír leírása / törzse |
| label | a három enum közül megadja hogy a hírt lakó, vagy közös képviselő tette-e ki esetleg kitűzött üzenet (mely így máshol jelenik meg) |
| timestamp | az adatbáziskezelő hozza létre a hír készülésekor |
| house\_id | megadja, hogy melyik házhoz tartozik a hír |

* + - 1. „poll” tábla

A címeken található szavazásokat tartalmazza egy adott kérdéssel kapcsolatban. Egy címe, rövid leírása illetve egy szavazat-számláló adattagja van, a létrehozásakor generálódó korábban említetteken kívül.

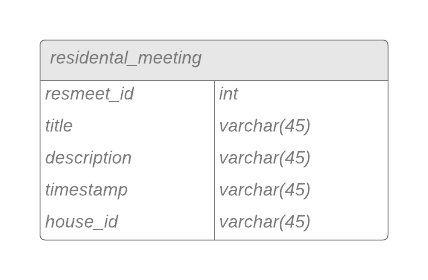


17. ábra a "poll" tábla UML diagramja

|  |  |
| --- | --- |
| Adattag | Leírás |
| poll\_id | a szavazás egyedi azonosítója |
| title | a szavazás címe |
| description | a szavazás leírása |
| votes | a érkezett szavazatok számlálója |
| label | a két enum típus adja meg hogy a szavazást lakó vagy a közösképviselő adta-e meg |
| hosue\_id | megadja, hogy melyik házhoz tartozik a szavazás |
| timestamp | az adatbáziskezelő hozza létre a készülésekor |

* + - 1. „residental\_meeting” tábla

A lakóközösség egy lakógyűlésének beszámolóját szemlélteti, az anyagi jelentést kihagyva. Tartalmazza a címét vagy témáját (ez a felvitelt végző közös képviselőtől függ) a gyűlésnek, valamint a leírását az itt elhangzottaknak (ami a való életben „jegyzőkönyv” néven lehet ismerős).

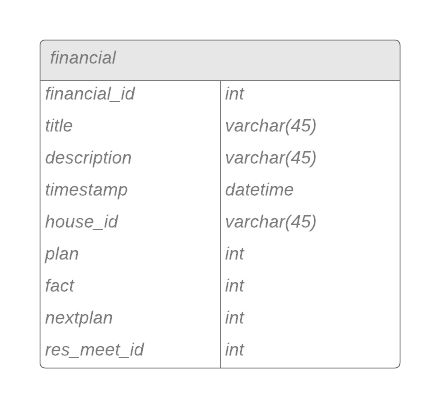


18. ábra a "residental\_meeting" tábla UML diagramja

|  |  |
| --- | --- |
| Adattag | Leírás |
| resmeet\_id | saját egyéni azonosítója a táblának |
| title | a lakógyűlés címe (vagy témái) |
| description | a jegyzőkönyv leírása |
| timestamp | az adatbáziskezelő hozza létre a tábla készülésekor |
| house\_id | azonosítja, hogy melyik házhoz tartozik a lakógyűlés jegyzőkönyve |

* + - 1. „financial” tábla

A lakógyűlések beszámolóinak anyagi vonzatait mutatja be. Egy ilyen tábla egy sornak tekinthető a lakóközösség kiadásait részletező táblázatokban. A kiadások rendelkeznek címmel, leírással, valamint a terv-tény „cellákkal” melyek a tervezett, illetve a valós kiadást mutatják. Emellett mivel ritka, hogy egy közösségi kiadást igénylő tevékenység (pl.: Lépcsőház kifestése, kéménytisztítás) egyszeri eset, az ismétlődése esetén a következő terv szolgál irányadásnak, de az opcionális.

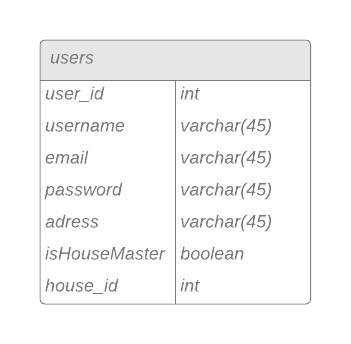


19. ábra A "financial" tábla UML diagramja

|  |  |
| --- | --- |
| Adattag | Leírás |
| title | a költségvetés rövid „címe” |
| description | a hosszabb leírása a kiadásnak |
| timestamp | az adatbáziskezelő hozza létre a rekord létrehozásakor |
| house\_id | azonosítja hogy melyik házhoz tartozik a költségvetés |
| plan | a kiadás tervezettértéke |
| fact | a kiadás valós értéke |
| nextplan | opcionális ha a következő „etapra” tervezen kiadást a ház |

* + - 1. „users” tábla

Egy felhasználót jelenít meg az adatbázisban, mely regisztráció során adja meg az adatait. A „isHouseMaster” logikai változóval különít el az adatbázis hátmester illetve lakos szintű felhasználókat.



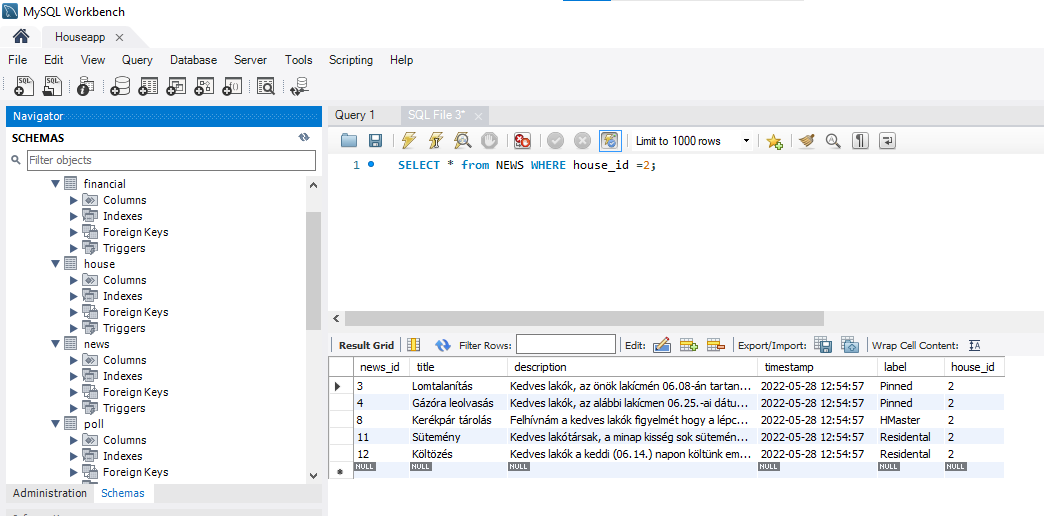
20. ábra a "users" tábla UML diagramja

|  |  |
| --- | --- |
| Adattag | Leírás |
| user\_id | egyéni azonosító a felhasználónak |
| username | falhasználónév |
| email | emailcím amivel regisztál |
| password | jelszó |
| adress | lakcím melyet lakó regisztrációkor kiválaszt a listából közösképviselőként meg is adhat újat |
| isHouseMaster | logikai változó, ha igaz akkor a felhasználó közösképviselőként lesz a rendszerben |
| house\_id | a saját lakcímének azonosítója Id-ként amit a backend generál |

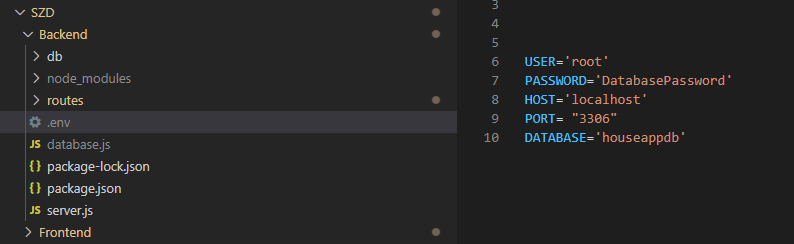
* 1. A fejlesztői környezet telepítése

Ahhoz hogy az alkalmazást használatba tudjuk venni, először szükséges az alábbi lépések elvégzése a számítógépen:

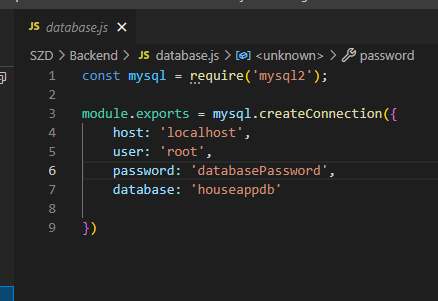
* Valamilyen egyszerűbb kódszerkesztő, én Visual Studio Code-ot használtam implementálás alatt.
* Egy MySQL Szerver telepítése valamint egy séma létrehozása, melyet az alkalmazás használhat illetve kezelni tudja az itt tárolt adatokat. Az SQL weboldaláról [13] letöltött .zip állomány kicsomagolása után tudunk telepíteni. Opcionális a hozzá tartozó My SQL Workbench alkalmazás, mely grafikus felületen teszi lehetővé (az egyetemi tanulmányok során használt SQL Developerhez hasonló működés, illetve kezelőfelület jellemzi) az adatbázistáblák kezelését, illetve az SQL lekérések eredményeit is láthatjuk. Mint mondtam ez az alkalmazás opcionális, a MySQL szerver konfigurálása illetve használata is megvalósítható parancssoros felületen is. Én magam a MySQL Workbench-el használtam a fejlesztés során, ezt is javasolom.



21. ábra A MySQL Workbench használat közben

* Egy .env illetve egy database.js file, melyben rögzítjük a MySQL szerverhez szükséges hozzáférési adatokat. Ezeket a file-okat a backend alkalmazás gyökér könyvtárában kell elhelyezni. Tartalmazniuk a kell az adatbázis eléréséhez a következőket: felhasználónév, jelszó, host elérése, port száma, illetve a létrehozott adatbázis séma neve. A .env file a seeder míg a database.js az endpointok adatbázis elérését kezeli.

22. ábra A .env file sematikája illetve helye



23. ábra a database.js file sematikája

* Node Js a [14] –es hivatkozásban megjelölt url-ről, a .msi kiterjesztésű telepítővel lehet telepíteni a számítógépre.
* Express Js, melynek telepítéséhez már szükséges a meglévő Node Js. Ezután a parancssorban ( vagy a kódszerkesztő termináljában) már csak ki kell adni az alábbi parancsokat:

$ npm init

$ npm install express

Az elsővel inicializáljuk az npm-et az adott mappába és egy package.json file-t generál, míg a másodikkal az Expresst telepítjük.

* Angular mint ahogy az Expresshez is, az Angular telepítéséhez is nélkülözhetetlen a Node js korábbi installálása. Ezt szinté a parancssorból vagy a kódszerkesztő termináljából tudjuk azután megtenni az alábbi paranccsal:

$ npm install -g @angular/cli

* 1. Fordítás és futtatás

A MySQL szerverrel a továbbiakban nem lesz teendőnk ugyanis a telepítéskor alapértelmezetten szolgáltatásként települ, tehát folyamatosan fut a háttérben (hacsak nem tiltjuk le, illetve nem telepítjük másképp). Emellett megjegyezném, hogy a MySQL Workbench sem szükséges hogy fusson az alkalmazás használata során, de az adatvizualizációs képességei miatt adatbázis-ellenőrzésre mindeneképp hasznos.

Az szakdolgozat két külön részét két külön parancsoros interfésszel érjük el. A „Backend” mappában a backend, míg a Frontend/SzDFrontend mappában a frontend alkalmazás található. A Bakend mappába lépve először is telepítsük az npm-el a megfelelő csomagokat az alábbi paranccsal:

npm update

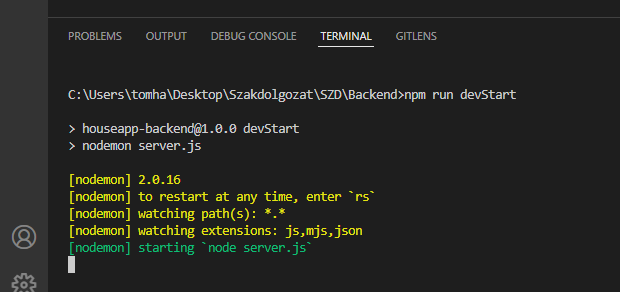
Ezután az adatbázistáblákat illetve alap adatokat hozzuk létre a seeder segítségével:

npm run seed

Majd végül indítsuk el az alkalmazást a nodemon csomag segítségével az alábbi paranccsal:

npm run devStart

Ha mindent jól csináltunk a konzolon az alábbi visszajelzés jelenik meg:



24. ábra Backend futtatása a nodemon csomaggal

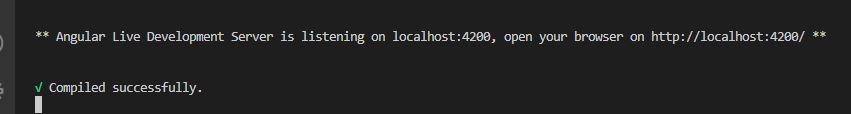
Ezután egy új parancssorban lépjünk be a Frontend/SzDFrontend mappába és itt is tepeítsük a megfelelő csomagokat a Node Js segítségével:

npm update

Majd futtassuk az Angular alkalmazást a következő paranccsal:

ng serve

Ez után az alábbi üzenetet kell, hogy lássuk:



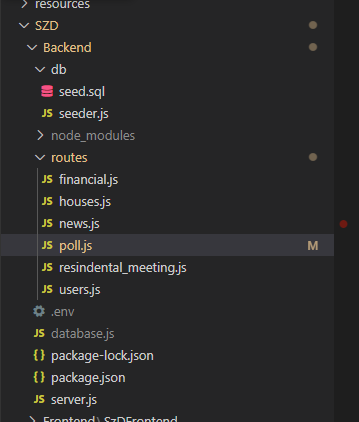
25. ábra Angular alkalmazás futtatása

Mind a front- mind a bakend részek úgy lettek megírva, hogy azok a fileokban történt változtatások mentése után egyből újra fordítanak illetve futtatják magukat. Ezt a folyamatot a konzolba írt Ctrl + C , majd a frontend esetében Y gombokkal szakíthatjuk meg.

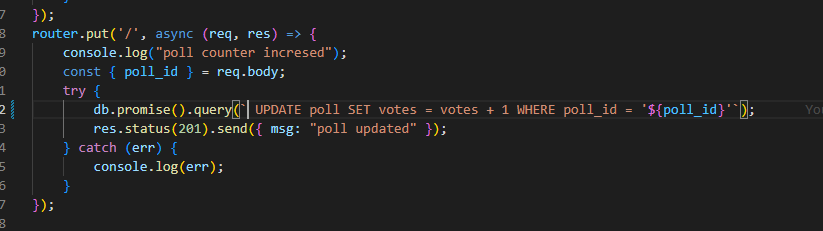
* 1. Az alkalmazás működése
     1. Backend

A backend alkalmazás központi része a server.js, mely definiálja a szerver által használt portot, a függőségek használatát illetve a szerver által használt route-okat.

Emellett két mappa található még a Backend mappában, az egyik a „db”, mely az adatbázis seedert illetve a feltöltés alapjául szolgáló MySQL utasításokat tartja magában. A második mappa pedig a „routes” mely magába foglalja a megnevezett url routokat, amiket a backend figyel, illetve API pontokat kezel. Ebben a mappában találhatóak a megadott elérési utak illetve ezek .js kiterjesztésű filejaik melyek egy-egy ilyen úton írják le az API pontokat.



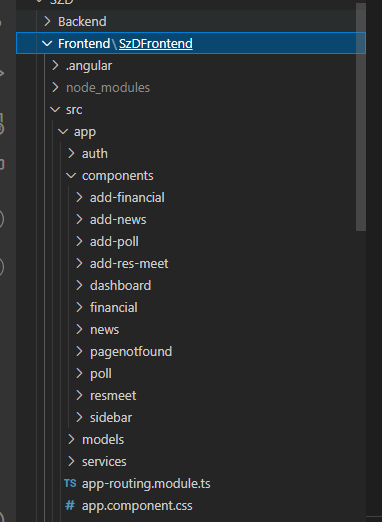
26. ábra a backend mappaszerkezete

Az API pontokba bekerülő utasításokat az adatbázis felé egy myslq2 nevű csomaggal kezeli a backend, ami ugyan nem valósít meg ezáltal komplexebb módszereket ( mint például a MVP) ám a használata egyszerűbb. Technikailag a Backend alkalmazás, amikor az adatbázis felé nyúl a szokásos CRUD API-okon keresztül ( GET, POST, PUT, DELETE), akkor az adatbáziskezelőbe mint a megadott (root) felhasználó belép és egy SQL query-t futtat le. Ezáltal az endpoinok tulajdonképpen a megfelelő SQL query-ből és az általuk használt adattagokból tevődnek össze.

27. ábra A poll.js put endpointja

Az endpointok felépítése pedig a következő képpen alakul. A legelején a router-nek megadjuk hogy az endpoint mivel hívható (PUT, GET, POST stb), majd a paraméterlistájában megadjuk hogy az URL változzon-e az alapvetőhez képest. A jelenlegi esetben a poll.js-ben az alap elérési út a „http://localhost:4000/poll” és ehhez tudunk hozzáfűzni további url tagokat ha indokolt. Ezután az alapvető „request” és „response” adattagokat (melyek rövidítve req és res ) felvéve egy anonim függvényhívással kifejtjük hogy az endpoint mit végezzen el az adatbázisban. Mivel az adatbázisművelet nem azonnali emiatt promise-el kell dolgozzunk és az egész függvény egy async, tehát aszinkron kell hogy legyen. Az adatok amik az endpointba érkeznek, azok jelen esetben is a kérés törzséből olvashatóak ki, illetve ezután adjuk értékül egy változónak. Ezután a változó értékét a megfelelő szintaktikával az adatbázisba tudjuk juttatni, majd ha onnan nem érkezik hibaüzenet, akkor válaszolunk a megfelelő státuszkóddal illetve opcionálisan tudunk hozzá csatolni üzenetet is.  
A fejlesztés és a tesztelés során az üzenet és státuszkód küldés is rendkívül sokat segített, mindenképpen ajánlom használatát másoknak is.

* + 1. Frontend



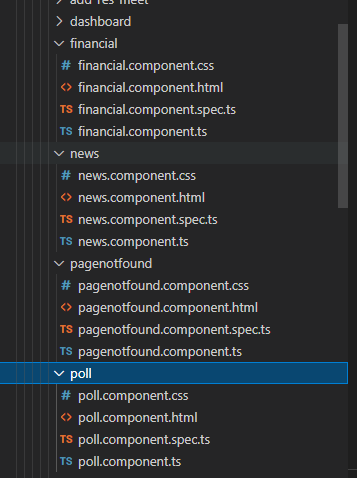
28. ábra A frontend mappaszerkezete

* + - 1. Autentikáció (Auth)

Az alkalmazásban található egy Auth mappa, melyben a hitelesítéshez, regisztrációhoz illetve a bejelentkezéshez szükséges komponensek találhatóak. Az alkalmazásban JWT (vagy tejles nevükön JSON Web Tokenek) tokenek segítségével van megoldva a hitelesítés.

* + - 1. Komponensek (Components)

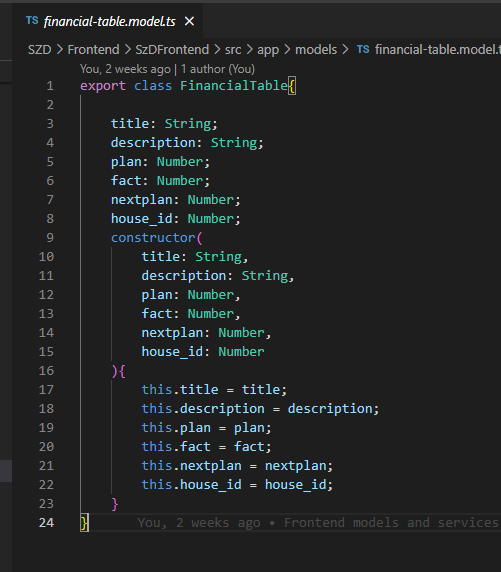
A komponensek az Angular keretrendszer alapjait képezik. Egy komponenst négy file egysége képez, ezek a következők: egy \*komponens-neve\*.component.css illetve .html file, melyek a megjelenítésért felelősek, egy \*komponens-neve\*.component.spec.ts mely a komponensen lefutó tesztekre készült, és végül a typescript file, melyben a komponensen belüli logikát írjuk le. Ezek a rendezett négyesek lényegében külön entitásként funkcionálnak, melyek közt lehet örököltetni is adattagokat, illetve funkciókat.



29. ábra komponensek

* + - 1. Modellek (Models)

A modellek mappában kapott helyet minden olyan entitás melyet példányosítani szeretnénk a frontenden valamilyen formában. Ezek leginkább olyan formában hasznosíthatóak, hogy az objektumokat a frontenden hozzuk létre és innen a backendre egy JSON string formájában küldjük át. Emiatt egy modell felépítésének (adattagok és típusok) meg kell egyezni a backenden tárolt képével.



30. ábra financial-table.model.ts

* + - 1. Services (Szolgáltatások)

A szolgáltatások segítenek összekötni a megjelenítést illetve az api-pontokon bejövő (vagy kimenő adatokat). Rendre minden modell rendelkezik egy saját service.ts file-al melyek a backenden található endpointokkal kapcsolatban írják le a kommunikációt. Azt hogy milyen típusú adatot illetve pontosan milyen URL-re küldjön, valamint hogy a formátuma illetve a változónevek a törzsben miként alakuljanak.



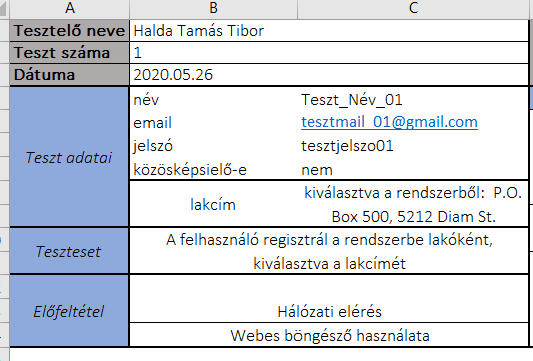
31. ábra részlet a news.service.ts -ből

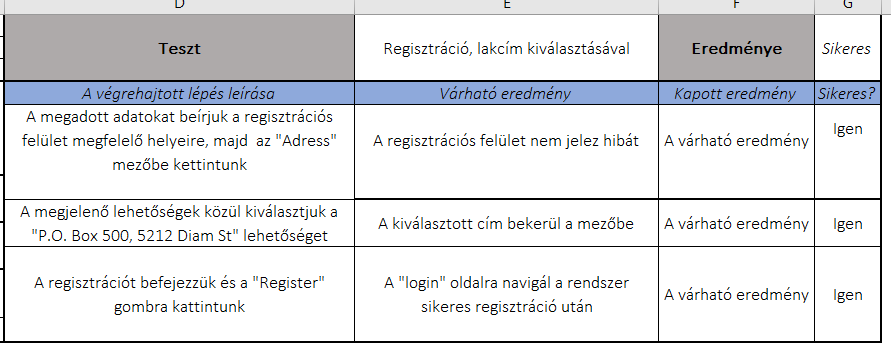
* + - 1. Assets

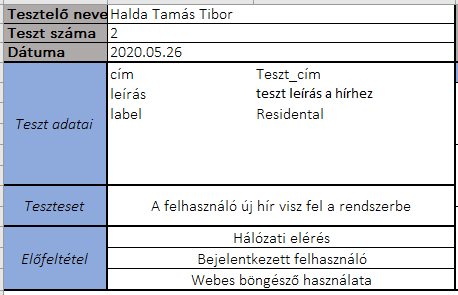
Az Assetts mappában található minden olyan média, melyre a frontendnek szüksége lehet. A jelenlegi esetben a különböző képeket helyezem el benne, melyek közül a jelenlegi formájában a pagenotfound modul használ egy képet, abban az esetben, ha olyan url címet adnánk meg, mely nem definiált előzőleg.

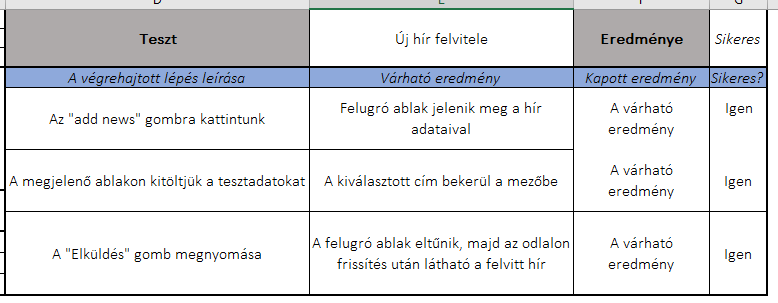
* 1. Tesztelés, tesztesetek

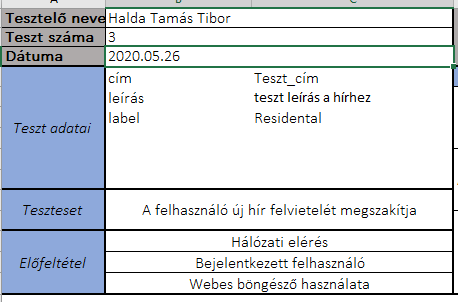
A szakdolgozatom tesztelését egy tömör, de kifejező un. Teszt jegyzőkönyvvel valósítottam meg illetve dokumentáltam a jegyzőkönyvek kivonatai pedig a következők:

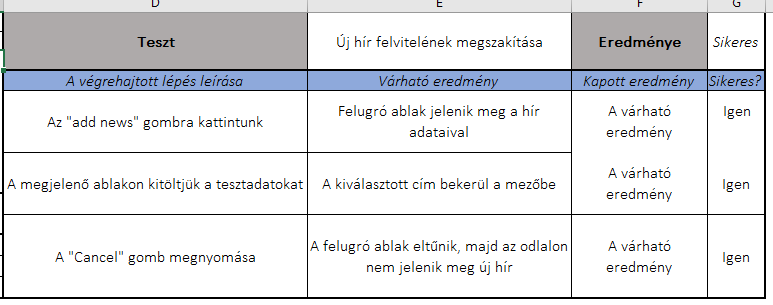


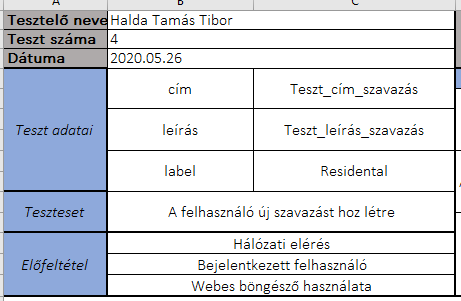


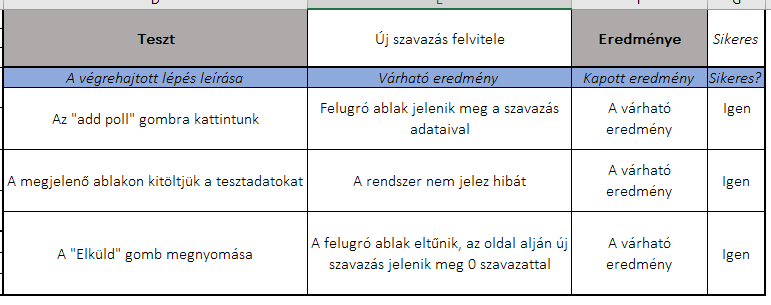


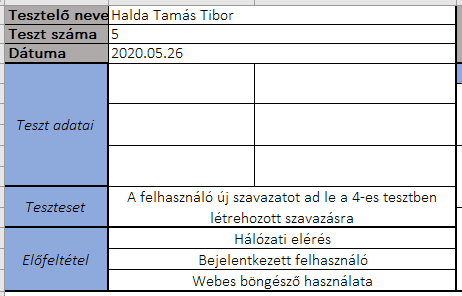


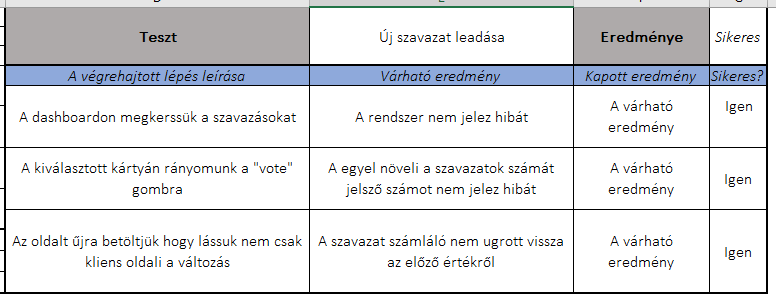












# Összefoglalás

A korábban vett felhasználói dokumentációban bemutattam a szoftver használatát, illetve kinézetét, valamint ki kitekintést tettem arra, hogy ki is a célközönsége a szakdolgozatom által megvalósított szoftvernek. Ezt követően a fejlesztői dokumentációban ismertettem a felhasznált technológiákat a szoftver backend illetve frontend oldalán. Ezt követően az adatbázis felépítését, tábláit, majd az egész környezet rekonstruálását írtam le. Végül kér fejezetben tárgyaltam a backend illetve a frontend felépítését illetve logikáját. A következő nagy fejezetben pedig ismertetem az alkalmazás lehetőségeit a további fejlesztés szempontjából.

# További fejlesztési lehetőségek

A szakdolgozatomban megvalósított alkalmazás (mint ahogy a legtöbb) még tartogat magában potenciált megvalósítható funkciók számára. Ezeket a funkciókat most igyekszem olyan sorrendben felvázolni, hogy a sorrendjük a lehető leghamarabb, legkevesebb hozzáadott logikával implementálhatótól a legkésőbbiig terjedjen. Ezeknek a sorrendjét a korábban említett adatbázis jelentősen meghatározza, ugyanis a funkciók implementálásnak folyamata a következő sorrendet alkotta a szakdolgozatom készítése alatt:

* Első lépés: adatbázisterv átszerkesztése, ami jelenthette a táblák adattagjainak kiegészítését, de akár teljesen új táblák vagy kapcsolatok létrehozását is.
* Második lépésként a backenden létrehozni ezekhez az új adattagokhoz vagy táblákhoz tartozó endpointokat, illetve a routingot felépíteni.
* Harmadik lépés a frontenden a modelleket, serviceket illetve a komponenseket kiegészíteni vagy teljesen létrehozni

Ebből a sorrendből jól látszik, hogy minél feljebb jutunk a backend felől annál kevesebb funkciót kell erősen figyelni, illetve implementálni. Ezek alapján a legelső bővítési lehetősége a frontendnek lenne szám szerint több is.

* A legelső funkció egy olyan lenne melyet idő hinyában nem tudtam hibátlanul megvalósítani emiatt kivettem a végleges szoftverből. Ezzel a funkcióval a közösképviselő regisztrációkor megadhatott volna új lakcímeket is. Ezt viszont az Express js SQL csomagjának imitációi miatt nem tudtam maradéktalanul implementálni időben.
* A legelső megvalósítható funkciók között lenne a frontend olyan továbbfejlesztése mely során a dátumok alapján lehetne valamilyen szűrést végezni, esetleg az Angular material „paginator” api-val oldalakra bontva lapozni ezeket. Ezáltal a felhasználó a számára megjelenített hírek közül a legfrissebbeket látná először, valamint egyszerre csak annyit látna amennyi kifér egy oldalban, alul pedig tudna lapozni pl. a hírek között vagy akár vissza tudna léptetni egy pár hónappal korábbi lakógyűlés jegyzőkönyvéig.
* A következő inkább már a másik rétegnek lenne könnyebbség: A közös képviselők a lakógyűlésekhez kapcsolódó anyagi tábláknál ki kell töltség hogy melyik lakógyűléshez tartozik az adott költséget leíró rekord. Ez azonban jelentősen egyszerűbb lenne, ha ezt a kettőt egy közös felületen lehetne felvinni, melyben, amíg be nem zárjuk automatikusan kitölti a megfelelő „resmeet\_id”-val, ezzel összekötve a két táblát.
* Ehhez szorosan tartozik a két tábla megjelenítése is, melyek mivel logikailag egy egységet képeznek egyszerre is kéne megjeleníteni őket. Az elképzelésem szerint a következőképpen: Fent a lakógyűlés címe, leírása és dátuma, majd alatta a hozzá kapcsolódó anyagi pénzügyi bejegyzések egy táblázat formájában, majd a következő lakógyűlés-pénzügyi tábla „párok” mivel a kettő közt ugye egy-sok kapcsolat áll fent.
* Színkódolás az egész UI-on hogy optikailag jobban el lehessen különíteni a különböző fontosságú híreket vagy szavazásokat.
* Következő fejlesztési lehetőségek közé tartozik, hogy az anyagi táblák importálását automatikusan lehessen végezni. Ezt egy .csv, vagy hasonló táblázatot leíró formátumból a frontenden betallózva juttathatnánk az adatbázisba. A parser a táblázat minden egyes sorát egy új „financial-table” rekordként tudná az adatbázisba juttatni, ezáltal nem kéne bajlódnia a számok begépelésével a közösképviselőnek amennyiben rendelkezik a saját gépén valamiféle táblázatba foglalt kiadási kivonattal.
* A szavazások módosítása egy viszonylag fontosabb fejlesztés lenne. A lakóknak és a házmestereknek is a szoftver jelenlegi állapotában csak „voksolni” van lehetőségük. Azonban ezt a rendszert ki lehetne egészíteni kettő vagy három lehetőséges szavazásokkal is, melyeknek a lehetőségeit szintén a szavazást felvivő egyén adhatná meg.
* További lehetőség lenne a backend endpointoknál az adat-validáció. Ezzel elkerülhető lenne, hogy akár véletlenül akár szándékosan olyan adatbázis-műveletet indítson a MySQL szerver felé, melyben adatvesztés vagy SQL hiba lépne fel. A jelenlegi formájában a backend nem fogad más típust, mint amit a frontenden lévő adatbevitel megenged. Ez ebben a formájában már egy védelmi sáv , ugyanakkor a backend önmagában is képes kéne hogy legyen véleményem szerint a beérkező adat vizsgálatára.

1. Hivatkozások:

[1] MYSQL Server <https://hu.wikipedia.org/wiki/MySQL>

[2] NPM <https://en.wikipedia.org/wiki/Npm_(software)>

[3] Node Js <https://en.wikipedia.org/wiki/Node.js>

[4] Express Js <https://en.wikipedia.org/wiki/Express.js>

[5] Angular <https://en.wikipedia.org/wiki/Angular_(web_framework)>

[6] Material Design https://en.wikipedia.org/wiki/Material\_Design

[7] JavaScript https://en.wikipedia.org/wiki/JavaScript

[8] Typescript https://en.wikipedia.org/wiki/TypeScript

[9] HTML https://hu.wikipedia.org/wiki/HTML

[10] CSS https://en.wikipedia.org/wiki/CSS

[11] UMLhttps://en.wikipedia.org/wiki/Unified\_Modeling\_Language

[12] Seedeléshttps://en.wikipedia.org/wiki/Database\_seeding**/**

[13] SQL Letöltés<https://dev.mysql.com/downloads/mysql/>

[14] Node letöltés<https://nodejs.org/en/download/>

[15] Generaldata- <https://generatedata.com/>