**Eötvös Loránd Tudományegyetem**

Informatikai Kar

Média- és Oktatásinformatikai Tanszék

**Modern Internetes Képes Kommunikáció**

**Témavezető Szerző**

Dr. Horváth Győző Franta Áron

Egyetemi docens Programtervező informatikus

**Budapest, 2022.**

Tartalomjegyzék

[1. Bevezetés 5](#_Toc122910344)

[1.1 Az oldal leírása, látványterv 5](#_Toc122910345)

[1.2 A használt technológiák 8](#_Toc122910346)

[2. Felhasználói dokumentáció 9](#_Toc122910347)

[2.1 A program telepítése saját eszközön 9](#_Toc122910348)

[2.3 Az oldal telepítése külső hálózatra 10](#_Toc122910349)

[2.2. Az oldal használata 10](#_Toc122910350)

[3. Fejlesztői dokumentáció 13](#_Toc122910351)

[3.1 A feladat 13](#_Toc122910352)

[3.2 A program felépítése 13](#_Toc122910353)

[3.3 Felhasználói esetek 14](#_Toc122910354)

[3.4 A frontend felépítése 15](#_Toc122910355)

[3.5 A backend felépítése 17](#_Toc122910356)

[3.5.1 Middlewarek és a használt csomagok 17](#_Toc122910357)

[3.5.2 A végpontok és a szerver felépítése 18](#_Toc122910358)

[3.6 Az adatbázis 19](#_Toc122910359)

[4. Tesztelés 22](#_Toc122910360)

[4.1 Tesztelési terv 22](#_Toc122910361)

[5 Fejlesztési lehetőségek 25](#_Toc122910362)

[6 Köszönetnyilvánítás 27](#_Toc122910363)

„*Nem az a feladat, hogy minden hallgatónknak mást, hanem az, hogy mindegyiknek eleget adjunk.*”

- EötvösLoránd

Eredetiségi Nyilatkozat

Alulírott **Franta Áron** (Neptun-kód: ITE2DZ) ezennel kijelentem és aláírásommal 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, 2022. december 26.

Franta Áron

# 1. Bevezetés

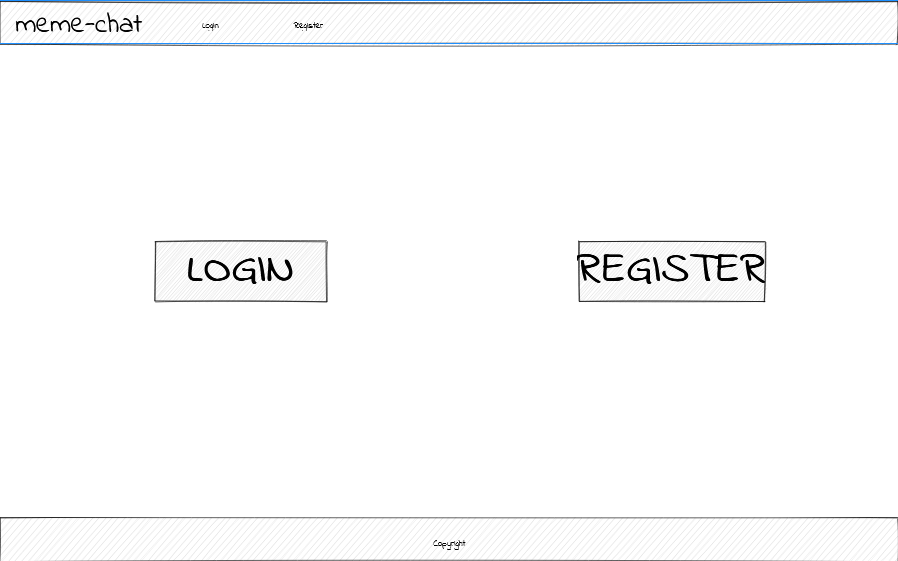
## 1.1 Az oldal leírása, látványterv

Szakdolgozatom egy modern eszközökkel készített internetes chat alkalmazást valósít meg, melyben az online kultúra aktív részévé vált mémekkel lehet kommunikálni. A mémek kis méretű képek melyekre szöveget helyeznek, ezek a szövegek általában egyszerű rövid mondatok, amely az adott szubkultúrában járatos embereknek tartalmat, hordoznak általában szórakoztató módon.

Mivel én is rendszeresen használom ezt a formátumot kommunikációra ezért érdekes kihívásnak tartottam egy olyan oldal megalkotását, ahol képeket manipulálva, különböző sablonokat kitöltve lehet kommunikálni.

Az oldalt egy egyoldalas (one-page) applikációként képzeltem el, amely egy főoldalból, egy bejelentkező és egy regisztrációs felületből áll, valamint egy sablonfeltöltő oldalból, egy chat listából és egy megjelenítő és küldőfelületből áll.

A főoldal nem bejelentkezett felhasználó esetén egyszerűen két gombot mutat, egy bejelentkezés és egy regisztráció.



1.1.1. ábra – A főoldal látványterve bejelentkezés nélkül

Bejelentkezett felhasználó esetén a főoldal a chateket listázó felületre mutat, amelyen megjelennek a chatek és van lehetőség új chatet létrehozni.

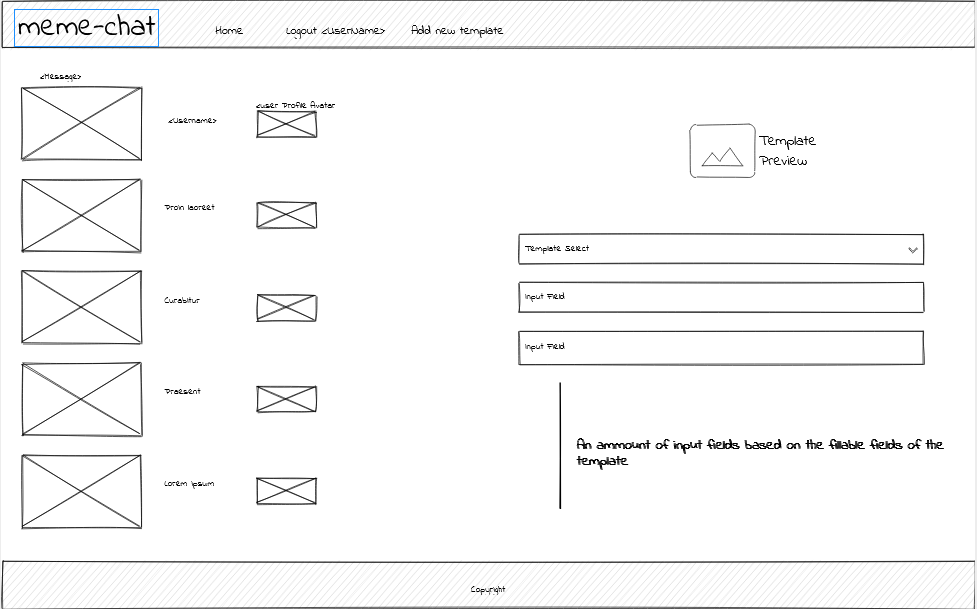
A képen szöveg látható

Automatikusan generált leírás

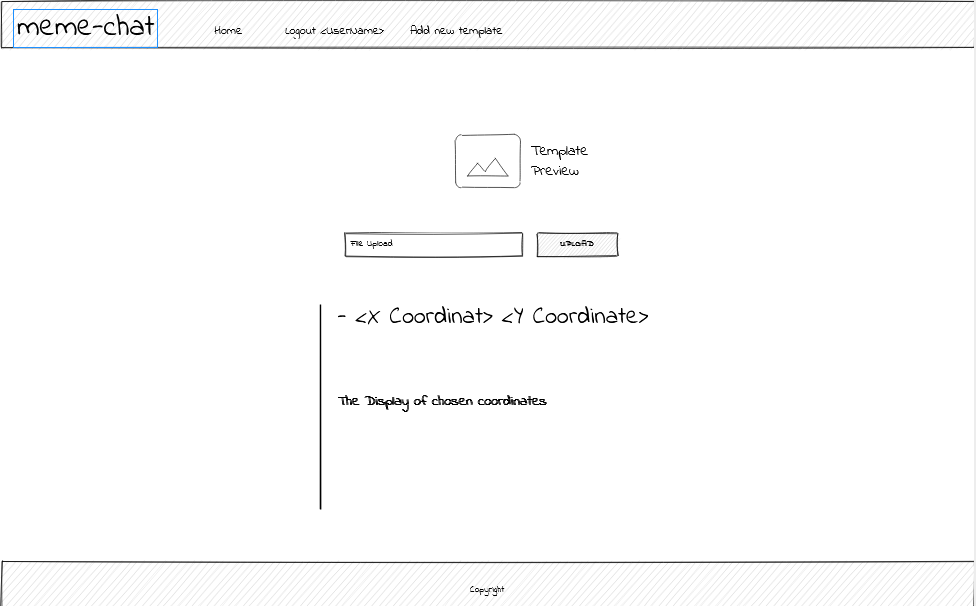
1.1.2. ábra – A chat lista látványterve

A chat oldalon láthatóak a chatbe küldött üzenetek, a küldő felhasználó és a felhasználó email címéhez tartozó profilkép. Mellette pedig látható a küldő felület. Ezen a felületen a felhasználó egy lenyíló menüből választhat az elérhető sablonok közül, a sablon kitölthető mennyiségű mezői szerint pedig megjelennek kitölthető szövegmezők. Ezek a 1.1.3. ábrán láthatók.

A sablon feltöltő felület (1.1.4. ábra) egy szövegmezőből áll, ahol elnevezhetjük a sablon, továbbá egy feltöltő mezőből. Ha feltöltünk egy képet a kép előnézete megjelenik. A képre kattintva a kép alatt megjelennek a kattintási koordináták. Ezek lesznek azok a mezők, amiket a küldő oldalon kitöltünk.



1.1.3. ábra – A chat megjelenítő oldal látványterve



1.1.3. ábra – A sablon feltöltő oldal látványterve

## 1.2 A használt technológiák

Az oldal megvalósításához teljes mértékben modern webes technológiákat használtam a moduláris működés és könnyű fejleszthetőség érdekében. Fontosnak tartom az ingyenes és open-source technológiák használatát ezért a fullstack alkalmazásomhoz tartozó keretrendszereket és az adatbázist is ez alapján választottam.

Adatbázisnak egy nem SQLt használó adatbázist választottam, a MongoDB-t, egyszerű használata és akár ingyenesen is használható online felhő alapú hosting rendszere miatt. A Mongoose keretrendszer használatával, a backenden könnyen lehet csatlakoztatni, továbbá nem igényli az összetett adatkapcsolatokat.

A backend server REST API-okon keresztül kommunikál a frontend-el, ehhez az egyetemi karrierem során megismert Fastify helyett az Express.js keretrendszert választottam, bár a Fastify gyorsabb és emelett szebb syntaxal rendelkezik, az Express.js ügyesebben kezeli az esetleges üres adatokat és mivel nagyobb a felhasználó bázisa az interneten ezért rengeteg dokumentáció és segédanyag elérhető, ami nagyban megkönnyítette, hogy az olyan funkciókat is probléma nélkül használjam amik ritkábbak és nem feltétlenül egyértelműek.

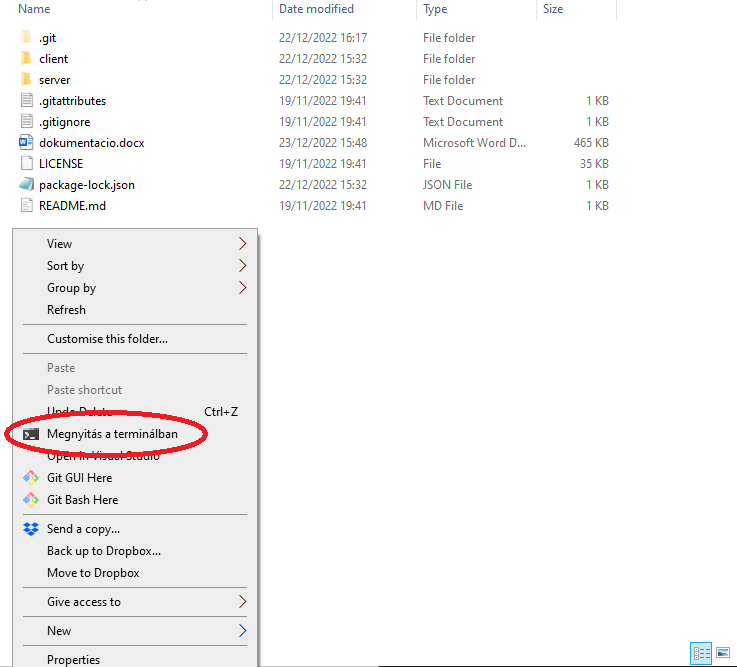
A frontend keretrendszernek a META által kiadott React keretrendszert választottam. A célom ezzel a választással, a könnyű komponensekből álló felépítés és a jól dokumentált működés volt, ezek mellet közre játszott továbbá az, hogy a React pár éve bevezette a funkcionális komponenseket melyek lecserélik az eddigi osztály alapú komponenseket. Ezek könnyebben tesztelhetőek mivel nincs saját állapotuk és teljes létezés alatt létező hook. Ehelyett ezek a JSX-et (JavaScript XML) visszatérítő java script függvények egyszeri propertyket használnak, emiatt használatba vettem a Reduxot hogy központosítsam az adattárolást egyetlen egy központi applikáció állapotba. Az oldal stílusosztályai Bootstrap template alapján működnek.

# 2. Felhasználói dokumentáció

## 2.1 A program telepítése saját eszközön

Első sorban a `node --version` segítségével ellenőrizzük, hogy elérhető-e a legfrissebb Node verzió eszközünkön, hiszen a Node Package Managert fogjuk használni (továbbiakban: npm). Ha ezzel megvagyunk akkor letölthetjük a szakdolgozat mappáját, ez tartalmaz egy server és egy client mappát, ezeknek mindegyike tartalmaz egy szükséges package-lock.json fájlt ami alapján az npm tudni fogja mely csomagokat kell telepíteni.

Nyissunk egy terminált a gyökér mappában:



2.1.1. ábra – Terminál nyitása Windows operációs rendszerben

Ezután a `cd server` paranccsal lépjünk be a szerver mappájába, majd az `npm i` utasítás kiadásával telepítsük a szükséges csomagokat. Ha ez sikeresen lefutott az `npm start` elindítja a szervert.

Ha ez megvan hasonlóan lépjünk be és indítsuk el a klienst `cd client` -> `npm i` ezek után pedig `npm start.`.

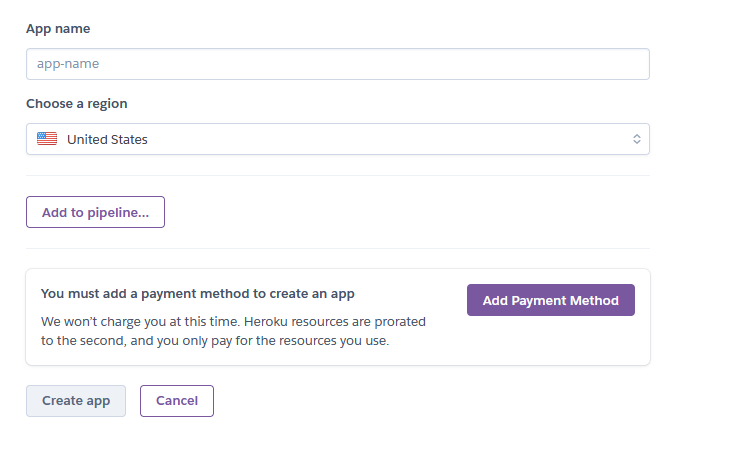
A szerver alapértelmezetten csatlakozik egy általam létrehozott felhőben futó Mongo adatbázishoz, ezt módosíthatjuk a `. /config` mappában a keys.js fájlban.

Ha minden lépés sikeresen teljesült a <http://localhost:3000/> címen elérhetjük az alkalmazást.

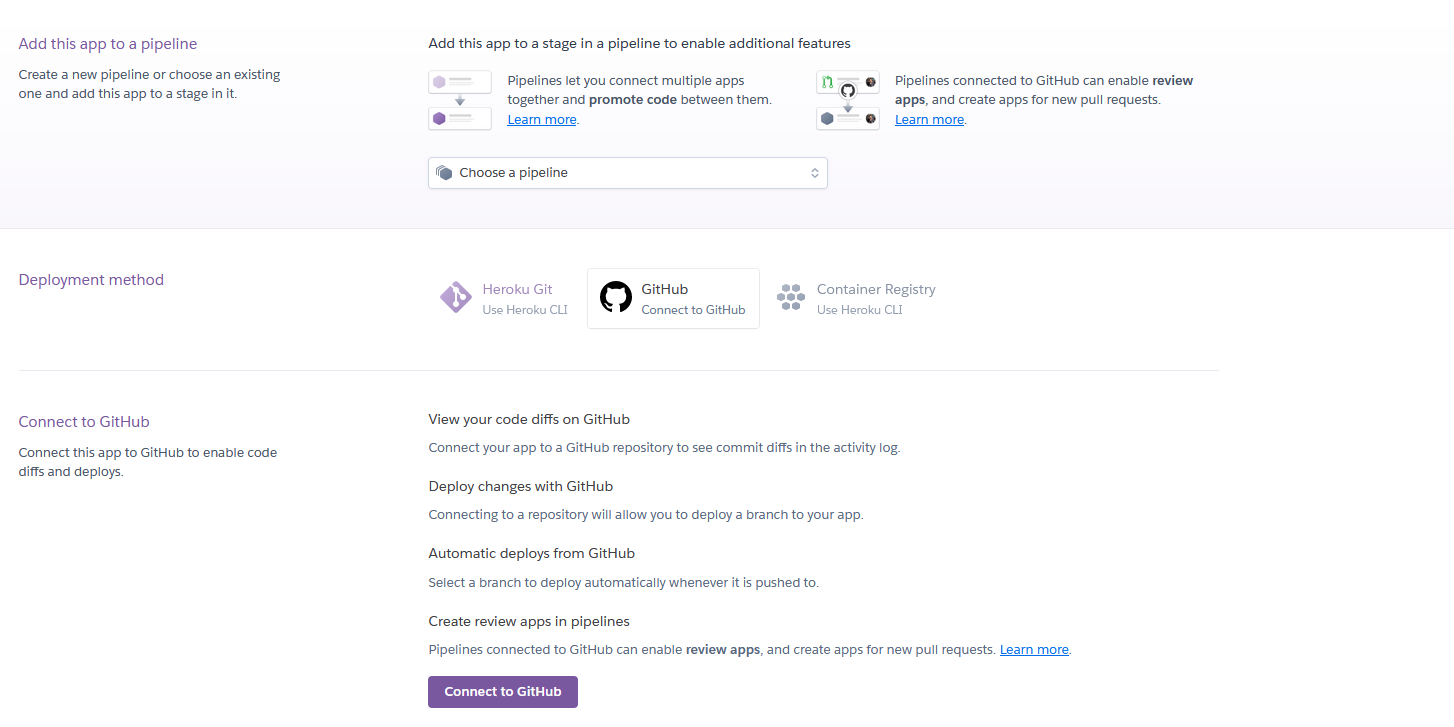
## 2.3 Az oldal telepítése a Heroku szolgáltatásra

Egy külső hálózatra való kitelepítésre való példának én a Heroku-t fogom használni, de megfelelhet nekünk egy Azure Web Application elkészítése is amennyiben szeretnénk CI/CD módszerekkel futtatni alkalmazásunkat.

Belépés után készítsünk egy új applikációt:



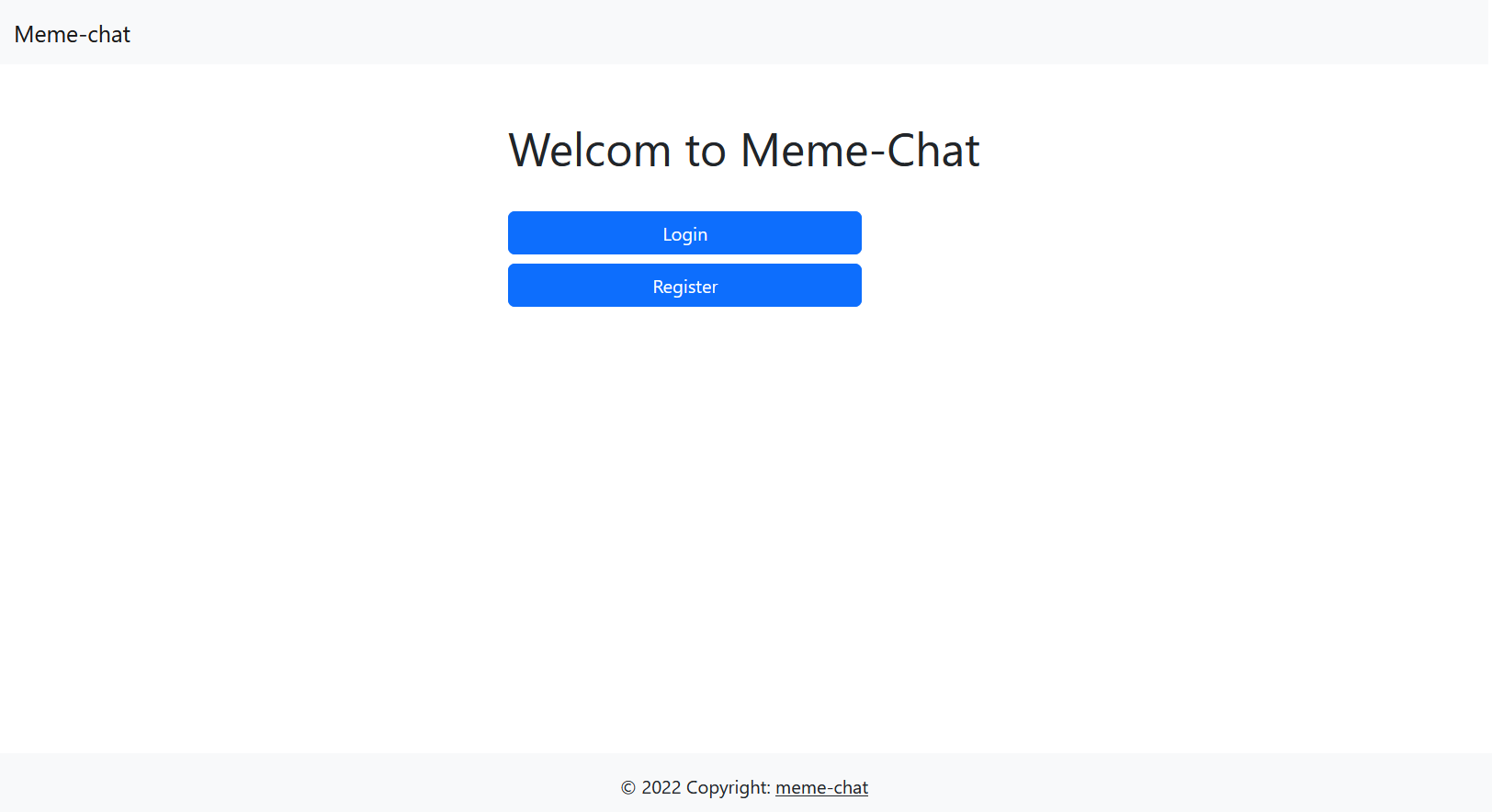
Ezután adjuk hozzá alkalmazásunkat egy pipelinehoz, amennyibben nincs még rendelkezésre álló, keráljunk egy újjat:



Amint hozzákötöttük alkalmazásunkat a GitHub vagy HerokuGit segítségével el is tudjuk indítani a deployolást.

## 2.2. Az oldal használata

Ha megnyitjuk az oldalt van opciónk belépni vagy regisztrálni. Nem belépett felhasználó nem tudja használni az oldal további funkcióit.



2.2.1. ábra – A bejelentkező felület

Amint a felhasználó bejelentkezett hozzáférést kap az oldal minden funkciójához. Az oldal fő részből áll, egy listából, ahol a bejelentkezett személy láthatja a chatek listáját, amelyekhez hozzá van adva. Egy sablon hozzáadó felületből, amelyen a felhasználónak lehetősége van képet feltölteni, egy nevet adni a sablonnak és meghatározni a pontokat, ahova szövegmezőt szeretne tenni. A szövegmezők a koordinátáktól jobbra lefele fognak elhelyezkedni, a kép bal felső sarka tartalmazza a 0, 0 koordinátát. Amennyiben mindent kitöltött el tudja küldeni a képet a szerver felé.

Az utolsó oldal pedig a chat oldal, ha a listában rákattintunk egy chatre akkor megtekinthetjük azt. Ezen az oldalon láthatjuk a chat nevét, és a korábbi üzeneteket a képernyő bal oldalán. Ezek kis kártyákban jelennek meg ahol megtalálható a küldő felhasználó neve, avatárja és a kép maga. A jobb oldalon a küldőfelület található, ahol egy lenyíló listából ki tudjuk választani a használni kívánt sablont. Ekkor a sablon megjelenik, és megjelenik megfelelő számú szövegmező, amit ki tudunk tölteni és elküldeni az üzenetet.

A képen szöveg látható

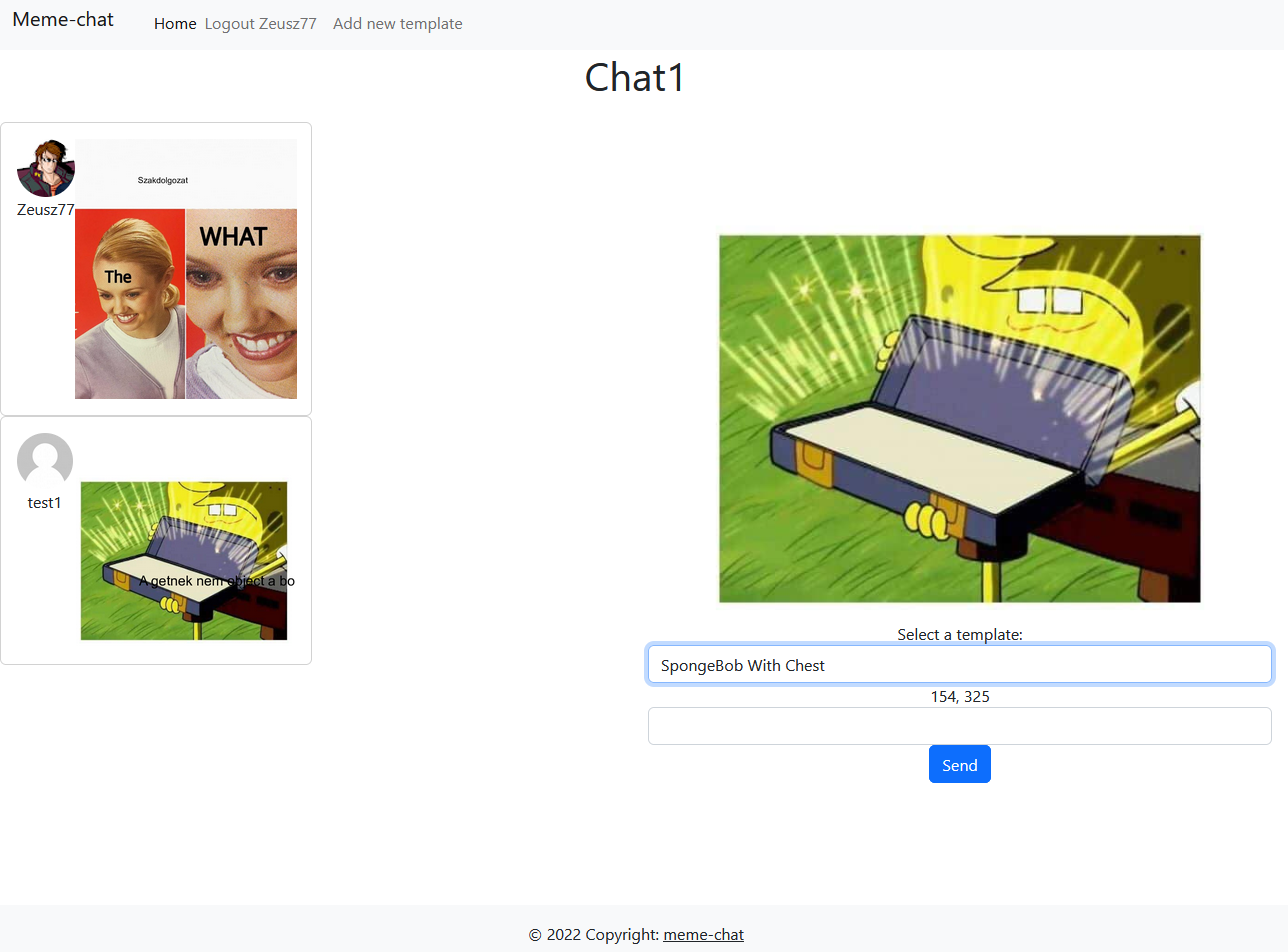
Automatikusan generált leírás

2.2.2. ábra – A chatek listája

A képen szöveg látható

Automatikusan generált leírás

2.2.3. ábra – Egy sablon feltöltése



2.2.4. ábra – Egy chat megjelenítése

# 3. Fejlesztői dokumentáció

## 3.1 A feladat

A szakdolgozatom témája egy modern technológiák segítségével megvalósított fullstack (frontend – backend – data layer) chat alkalmazás, melyen keresztül a felhasználók egyedi módon kommunikálhatnak egymással privát- és csoportos beszélgetésekben. Az alkalmazás lényege, hogy konvencionális szöveges üzenetek küldésére nincs lehetőség, a kommunikáció az internetes kultúrában gyökerező képekkel történik, melyeket ki lehet egészíteni egy-két sor felső- és alsószöveggel. A felhasználóknak lehetőségük van bejelentkezni és regisztrálni, beszélgetést kezdeményezni, valamint egyedi sablonokat feltölteni.

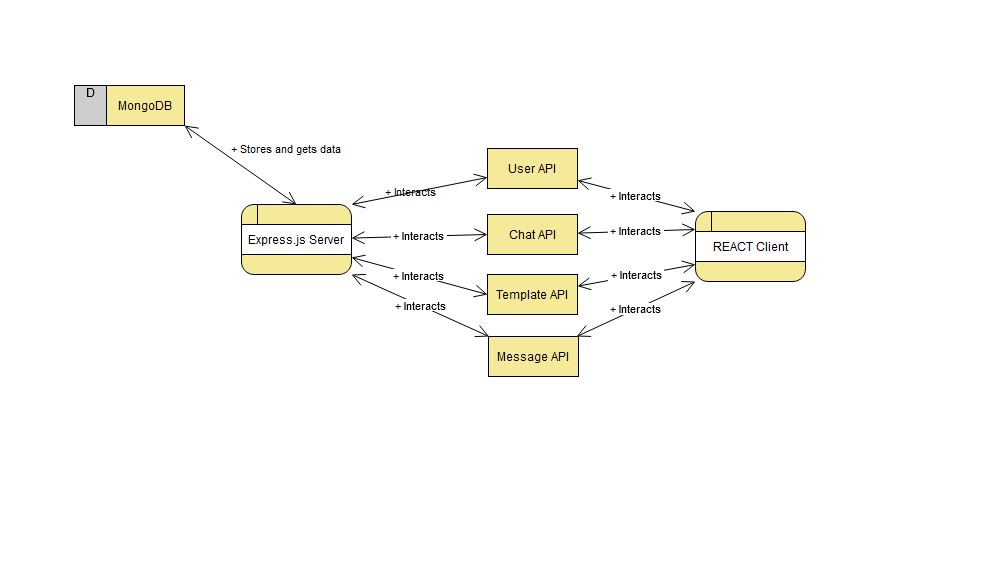
## 3.2 A program felépítése

A megvalósításhoz MERN (Mongo – Express – React - Node) stacket választottam.

Az általam készített webalkalmazás tervezése során két megközelítés merült fel, ez a két lehetőség egy ponton tért el egymástól, az üzenet által használt sablon képet módosíthatom vagy a frontenden, vagy a backenden. A frontenden való módosítás kisebb sújt helyezett volna a szerverre, és kicsit többet a felhasználó saját eszközére, mivel az internet sebességével napjainkban már nincs probléma és a cél az volt, hogy minél többen különböző erősségű eszközön is hozzáférhessen az alkalmazáshoz ezért én inkább a szerveren való képmanipulálás mellett döntöttem.

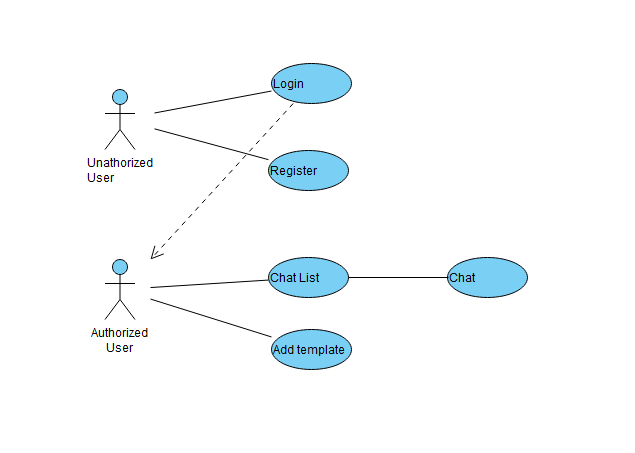
Mivel egy Mongo adatbázist használok, bár technikailag lett volna lehetőségem binárisként képeket tárolni, inkább a szerveren való tárolás mellett döntöttem, mivel a saját szerverem tárhelyét könnyebben bővítem, mint a felhő alapon hostolt adatbázisszerveremt.

Az adatfolyam végül a következőképp épül fel:



3.2.1 ábra – A Dataflow Diagramm

## 3.3 Felhasználói esetek

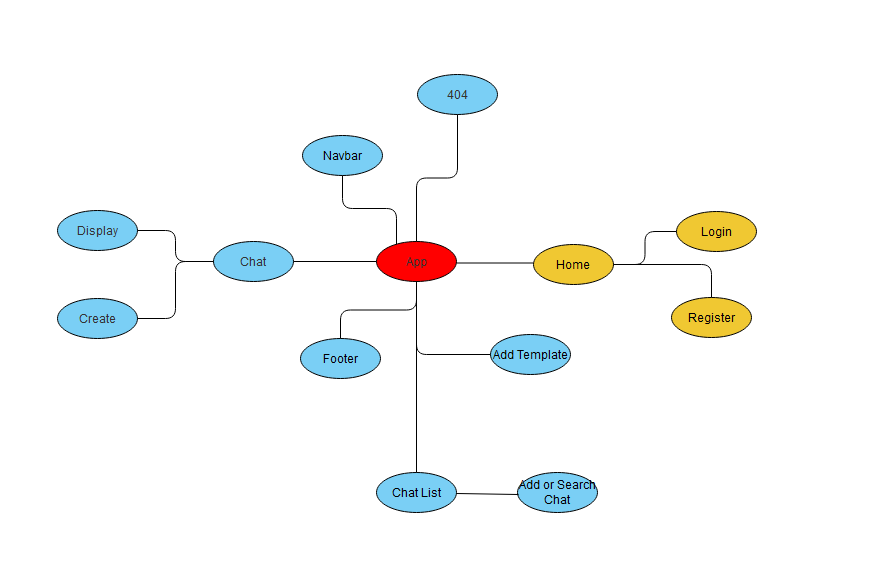


3.3.1 ábra – A felhasználói esetdiagramm

A felhasználói esetek a 3.3.1. ábrán láthatóak szerint kétfelé bomlanak, a nem bejelentkezett felhasználóknak két lehetőségük van az oldalon, regisztrálni vagy bejelentkezni. Miután egy felhasználó bejelentkezett elérést kap az oldal teljes funkcionalitásához. A chatek listáját látja először, amikor bejelentkezik, innen hozzáfér a saját chatjeinek listájához vagy kreálhat új chatet. Ha a felhasználó belép egy chatbe látja a chatbe küldött korábbi üzeneteket, a küldő avatárjával és felhasználó nevével, továbbá lehetősége van új üzenetet kreálni. Egy teljesen független felületen pedig elérhető sablonok feltöltéséhez.

## 3.4 A frontend felépítése

A React frontend komponensekből épül fel, ezeket a komponenseket egy központi router köti össze, ez bár egy külön komponens a felhasználó számára soha nem jelenik meg ezért nem reprezentálta a komponensdiagramban.



3.4.1. ábra – A komponens diagram

A kék komponensek csak bejelentkezés után érhetőek el, minden más esetben a router visszairányítja a felhasználót a Home komponensre. Ezt a működést a `react-redux` könyvtárban található szelektorok segítségével és a `react-router-dom` könyvtárban található navigátorokkal (`Link` – dom elem, `useNavigate()` - függvény) valósítom meg.

A szelektorok használatára a Redux használata miatt van szükség. Mivel önmagukban stateless funkcionális életciklus komponenseket használok, ezért bindolt propok helyett helyben létrehozott propokat és hookokat használok, és minden adatot a központi redux storeban tárolok. A reduxban az adatok tarolását és visszaadását Reducerek valósítják meg, a 4 féle adattípusnak van 1-1 saját reducere amik a hozzájuk tartozó adatok elkapásáért és dispatcheléséért felelnek. Amikor a Reducerek felé dispatchelünk egy adatot fontos, hogy megadjunk egy típust, ami alapján ezek a feldolgozó függvények esetekre vannak bontva. Ezek a köztes függvények egy middleware-é lesznek kombinálva és App szinten bekötve a store-ba és az alkalmazásba.

Emellett a frontend használja a local storaget a JWT token eltárolására, erről a tokenről bővebben a 3.5. fejezetben.

A backend és a frontend közötti kommunikáció HTTP protokollon keresztül történik, a backenden megvalósított APIkat a frontenden az Axios könyvtár segítségével fogjuk elérni. Az axios egy promise alapú http könyvtár, ami könnyűvé teszi az adatok fetchelését a szerverről. Implementálja az összes számunkra fontos request típust, és automatikusan json formátumba hozza a visszaérkezett adatokat.

A segéd függvények között definiáljuk a szelektorokat, például a getIsLoggedIn szelektort a következőképp:



3.4.2. ábra – Egy példa a szelektorokra

Ezeket a szelektorokat használjuk arra, hogy a Redux storeban eltárolt adatainkat visszanyerjük amikor a különböző komponensekben ezekkel az adatokkal szeretnénk dolgozni.

Egy másik segédfüggvény, ami a frontendhez készült a setAuthToken:

A képen szöveg látható

Automatikusan generált leírás

3.4.3. ábra – A setAuthToken segédfüggvény

Amikor az Axios segítségével kéréseket küldünk a szerver felé, fontos, hogy a kérésben bent legyen az előbb említett JWT token, ezért a beléptetéskor ez a függvény állítja be, majd kiléptetéskor ez a függvény kapcsolja ki az axiosból az alap értelmezett headert.

Az oldal komponensei között való közlekedéshez a a `react-router-dom` könyvtárból a BrowserRoutert használjuk, ez az oldal elérési útján keresztül dönti el, hogy melyik oldalra irányítson és kihasználja a modern böngészőkbe épített history stacket.

Az alábbi függvényeket használjuk:



3.4.4. ábra – A Router importálása

## 3.5 A backend felépítése

### 3.5.1 Middlewarek és a használt csomagok

#### CORS

Ez a csomag teszi lehetővé a Cross-Origin Resource Sharinget, ezzel tudjuk engedélyezni, hogy ugyan azon a hoston futtassuk a backend és frontend szolgáltatásunkat

#### bodyParser

A bodyParsert használjuk a beérkező kérés által hozott adatok feldarabolására. Legtöbbször az Expressel együtt használják, ahogy én is tettem a szakdolgozatomban.

A csomag lehetővé teszi, hogy a http formában beérkező kérésből többféle formájú adatot is kinyerjünk, például JSON, vagy nyers szöveg.

#### Mongoose

A Mongoose az adatbázis elérését szolgálja, egyszerű séma alapú modellekkel operál. A sémák specifikálják az adatok típusát, amik az adatbázisban lesznek tárolva, ezekben a sémákban adjuk meg a különböző validációka, megkötéseket is. Bár a Mongonak nincs szüksége rá, hogy minden adat konzisztens legyen, ezek a sémák biztosítják az egy kollekcióba tartozó adatok egyformaságát.

Az adatbázishoz az alábbi módon tudunk csatlakozni:

A képen szöveg látható

Automatikusan generált leírás

3.5.1.1. ábra - Az adatbázis bekötése

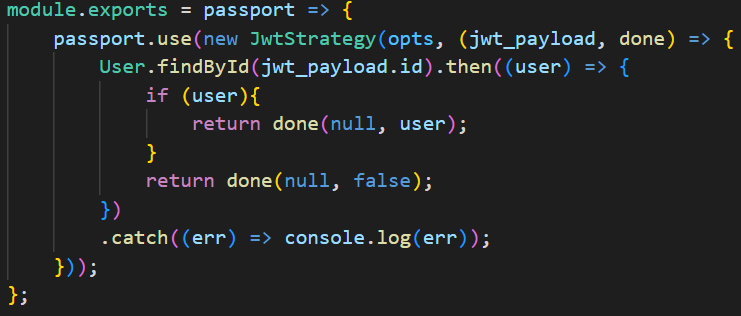
A konfigurációs mappában az a kulcsok közt található mongoURI adattagot módosítva más adatbázisokra is ráköthetjük az alkalmazást.

#### Passport és a JSON Web Token

A Passport egy gyakran használt authentikációs eszköz Node.js applikációkban, különböző felismerő stratégiákat lehet definiálni, amikkel különböző módokon lehet beazonosítani a felhasználót, például jelszó és felhasználónév alapján, vagy akár Facebook vagy Google felhasználó alapján is.

Ezt össze tudjuk kötni a JWT tokennel ami egy standard információ közlő eszköz. A passport le tudja ellenőrizni a generált JWT token érvényességét. Innen ki tudjuk nyerni a felhasználó azon adatait, amiket a tokenben eltároltunk. ezeket az authentikált végponton a `req.user` teszi számunkra elérhetővé.

A konfigurációs fileok között definiáljuk a működést:



3.5.1.2. ábra – A passport konfigurálása JWT tokenekhez

#### JIMP

A JIMP a program fő működéséért felel, ez egy JavaScript könyvtár felel a képmanipulációért. Sokféle különböző műveletet tudunk elvégezni, JPEG, PNG és GIF formátumot is tudunk vele manipulálni.

A csomag képes újra méretezni, körbe vágni, forgatni, más képeket, vízjegyeket és szövegeket rányomni a képre. Ezek mellett használhatjuk másféle manipulálásra, például fényerő változtatására, kontraszt állításra, színséma igazításra és filterek és effektek használatára.

Ezek mind a memóriába történnek, és nem szükséges hozzájuk semmi más külső könyvtárra nincs szükség, ezért könnyen használható és deployolható szerveroldali alkalmazásokban és microservice-inkben.

#### Validator

Ezt a külső könyvtárat a felhasználó által megadott bemeneti értékek érvényesítésére. A leggyakrabban alkalmazott használati módszer a form inputot ellenőrizni vele, de emellett lehet használni API válaszok tesztelésére és adatbázis lekérések ellenőrzésére is.

#### Bcryptjs

A bcryptet használjuk arra, hogy a felhasználó jelszavát kódoljuk, ehhez a Hash algoritmust használunk.

Ezek az algoritmusok egyirányú átalakítást végeznek, elfogadnak egy só értéket és egy folytonos hosszúságú kódolt szöveget térítenek vissza. Minél erősebb plusz szöveget és algoritmust használunk, annál nehezebb lesz nyers erővel feltörni vagy visszafejteni a tárolt jelszóinkat esetleges betörés estén.

### 3.5.2 A végpontok és a szerver felépítése

Az szerver gerince egy Express.js szerver, ez a keretrendszer a http modulra van ráépítve és egy könnyen használható interfacet szolgáltat http szerverek és kliensek készítésére. Ezek mellett különböző opciókat is szolgáltat többek közt válasz generálásra.

Egy API végpont egy http metódus és egy URL elérési pont kombinációja.

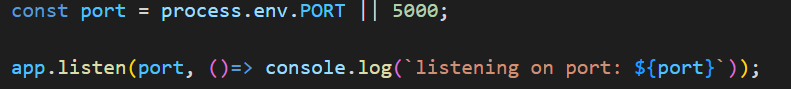
Egy teszt metódus a következőképp van bekötve az alkalmazásba:

A képen szöveg látható

Automatikusan generált leírás

3.5.1.3. ábra – Az Express definiálása

Miután az összes végpontunk be van kötve és definiálva van az alkalmazást a következőképp tudjuk elindítani:



3.5.1.4. ábra – Az alkalmazást elindító függvény.

Itt definiálunk egy port-ot, amin az alkalmazás kommunikálni fog. Ezt a port-ot lokális rendszeren alapértelmezetten 5000re állítottam, de egy külsőn rendszerre való kitelepítéskor automatikusan beállításra kerül.

## 3.6 Az adatbázis

Adatbázisnak az előbb is említett MongoDB-t használtam, ez egy doumentum orientált adatbázis, flexibilis, scalelhető és könnyen használható, az adatokat egy JSON szerű formátumban tárolja, amit BSON-nek hívnak (Binary JSON).

Az adatokat egy rugalmas, hierarchikus struktúrában tárolja, ezeket dokumentumoknak nevezi, ezek kulcs érték párok, és akármennyi beágyazott tömböt vagy dokumentumot tartalmazhat. A dokumentumok egy konvencionális adatbázis sorainak felelnek meg amiket kollekciókban tárolunk, ezek lennének a táblák.

Bár mint említettem az adatbázisunk nem erőlteti az adatok azonosságát kollekciókon belül a sémák meghatározásával biztosak lehetünk abban, hogy adataink megfelelő formátumúak feldolgozáskor.

Minden sémába tartozik egy az adatbázis által generált ID és egy dátum, ami a kreálás idejét jelzi.

A képeket az adatbázis helyett a szerveren tároljuk, mivel bináris adatok tárolása kényelmetlen, ezért a sablonoknál csak a sablonhoz tartozó kép fájlnevét tároljuk el.

|  |  |
| --- | --- |
| Séma | Adatkulcsok |
| User | handle (String) |
| email (String) |
| password (String) |
| avatar (String) |
| Message | template (String) |
| text (Array of Strings) |
| chat (String) |
| user (String) |
| userAvatar (String) |
| messageImage (String) |
| Chat | name (String) |
| participants (Array of Strings) |
| Template | name (String) |
| imageName (String) |
| numberOfFields (Int) |
| fileds (String) |

# 4. Tesztelés

Mielőtt beüzemelnénk az oldalt fontos, hogy meggyőződjünk az alkalmazás megfelelő működéséről. Kézzel tesztelés esetén az oldal minden funkcióját ki tudjuk próbálni az előre létrehozott teszt felhasználókkal

* test1 – [test1@gmail.com](mailto:test1@gmail.com)
* test2 – [test2@gmail.com](mailto:test2@gmail.com)

Ezeknek a felhasználóknak a jelszava „asd123”.

A felhasználókat szintén előre létrehozott csatornákban már elküldött üzenetekkel láthatjuk betölteni. Kreálhatunk új chateket vagy használhatjuk az előre kész adatokat.

Az API-k működéséről meggyőződhetünk Postman használatával, itt az elérési utat beírva el tudunk küldeni kéréseket a szerver felé, hibás kitöltés esetén hibaüzeneteket kapunk vissza megfelelő hibakóddal. Helyes kitöltés esetén le tudjuk ellenőrizni, hogy megfelelő adatokat kapunk e vissza, és a recordokat le tudjuk ellenőrizni az adatbázisunkban.

## 4.1 Tesztelési terv

Bejelentkezési oldal

* Az oldal nem regisztrált felhasználókat nem enged bejelentkezni
* A form figyelmeztet helytelen email cím formátum esetén és túl rövid jelszó esetén
* Hiba esetén a hiba szövege jól látható, elkülönített helyen piros színnel megjelenik

Regisztrációs oldal

* A form nem engedi elküldeni a kérést nem kitöltött mezők esetén
* A from nem enged túl rövid jelszóval regisztrálni.
* A from ellenőrzi, hogy a két jelszó sávba írt jelszó megegyezik-e egymással
* A form ellenőrzi, hogy az emailcím valid formátummal rendelkezik e.
* A szerver hibaüzenetet küld vissza abban az esetben, ha a felhasználó által megadott email cím már használatban van.
* Hiba esetén a hiba szövege jól látható, elkülönített helyen piros színnel megjelenik

Navigációs sáv

* A navigációs sáv nem bejelentkezett felhasználó esetén, egy a Login formra és egy a Regisztrációs fromra mutató linket jelenít meg
* Bejelentkezett felhasználó esetén a navigációs sáv 3 linket jelenít meg
  + Chat Lista, amelyik a chateket listázó oldalra mutat
  + Logout <Az aktuálisan bejelentkezett felhasználó neve>, amelyik kijelentkezteti a belépett felhasználót
  + Új Sablon, amelyik a sablon hozzáadó oldalra mutat

Chat lista oldal

* A chateket hozzáadó form megfelelően megjelenik
* Chat hozzáadásakor a chat nevét üresen hagyhatjuk, de a felhasználók listáját nem
* Chat hozzáadása estén az oldal azonnal az új chatbe irányít minket
* A chatlista csak olyan chateket jelenít meg amelyeknek a felhasználó nem tagja
* A chatlista, amikor új chatet hozunk létre majd visszalépünk az oldalra frissíti az újonnan létrehozott csatornával
* Egy chatre kattintás esetén a megfelelő chat oldalára irányulunk

Chat oldal

* A chat oldalra való belépéskor a chat neve megfelelően megjelenik
* A chat oldalra való belépéskor az üzenet küldő rész megjelenik
  + A Sablont kiválasztó menü tartalmazza a sablonokat
  + Sablon kiválasztása esetén az előnézet megjelenik
  + Sablon kiválasztása esetén az input mezők megfelelő számban megjelennek
  + Ha minden mezőt megfelelően kitöltöttünk el tudjuk küldeni az üzenetet
* A megjelenítő oldalon az üzenetek megfelelően megjelennek
  + Új üzenetkor az üzenet, amelyet küldtünk megfelelően megjelenik
  + Mástól érkező üzenet esetén az oldalt nem kell újra töltenünk
  + Az üzenet kártyájában a felhasználó neve és avatárja megfelelően megjelenik

Új sablon oldal

* Új sablon hozzáadása esetén visszakerülünk a chateket listázó oldalra
* A chatsablon nevét ki kell tölteni
* A kép feltöltése esetén a kép előnézete megfelelően megjelenik
* A képre való kattintáskor a kattintás relatív koordinátái megjelennek
  + A kattintások száma nem limitált
  + A kép kitölthető mezői megegyeznek a listában megjelenő koordinátákkal
  + Egy már megadott koordináta új kattintással eltüntethető

Kijelentkezés

* A felhasználó megfelelően kijelentkezik
* A lokális tárolóból és a Redux-ból az adatok kiürülnek

A manuális tesztelés helyett később készíthető automatikus tesztelő, amelybe implementáltuk a tesztelési terv lépésit, és tesztelhetjük a teljes működést, vagy külön a frontend megfelelő megjelenését, vagy a backend megfelelő működését.

Ezt az automatikus tesztelőt Pythonban írnám meg, a frontend teszteléséhez megfelelően használható a Selenium. A WebDriver segítségével szimulálni tudunk egy chromium alapú böngészőt, amelynek megjelenítése nem feltétlenül lényeges. Ezzel elemezni tudjuk a megjelenő HTML oldalak tartalmát, és a gombok megfelelő működését.

A Python requests könyvtárával tesztelni tudjuk az API végpontok önálló működését. Tudunk hozzájuk Unit teszteket is írni amennyiben felállítunk egy mock adatbázist és implementálunk egy teszt módot az Express szerverünkbe, amelyik az eredeti adatbázis helyett a teszt keretrendszert hívogatja.

A tesztelési lépéseket kézzel elvégeztem.

# 5 Fejlesztési lehetőségek

Az oldal fő funkciói ugyan megfelelően működnek, az oldalt több ponton lehetne fejleszteni a jobb felhasználói élmény érdekében.

Az oldal jelenlegi megjelenítése egy nagyon egyszerű Bootstrap template, egy saját jól kidolgozott CSS stílusoldal elkészítésével sokat lehetne javítani az oldal kinézetén. Implementálható lenne egy sötét mód és egy magams kontraszú mód a gyengén látók számára, ez nagyobb közönség felé is használhatóvá tenné az oldalt.

Használhatnánk a gombokhoz és az irányításhoz FontAwsome ikonokat a szöveges megjelenítés mellett, ezzel megtartanánk a szövegolvasók használhatóságát, viszont sokban javítana az oldalon való vizuális eligazodásban.

Emellett az oldalnak lehetne tervezni sajátos „brandet”, külön logóval, ami többféle méretben elérhető, lehetne hozzá színsémát készíteni, és készíteni a böngésző felső sávjában megjelenő favicont.

##### Felhasználói funkciók

Az oldal felhasználó funkciói jelen pillanatban nagyon kevés manapság már minden oldalon megtalálható funkciót implementál. A modularizált megvalósítás miatt az oldalon könnyen fejleszthető lenne az alábbi funkciókkal:

* A felhasználói profil módosítása: Implementálható lenne egy felület, ahol a felhasználó módosítani tudná az oldalon megjelenő nevét, jelszavát, emailcímét. Amennyiben a felhasználó profilképének az oldalon belüli módosítását is szeretnénk megvalósítani, a jelenlegi Gravatar linkek helyett tárolhatnánk a profilképet.
* Az üzenetek törölhetősége: Az üzenetek utólagos törlésére és módosítására jelen pillanatba nincs lehetőség, viszont két új API bevezetésével ezek könnyen megvalósíthatóak.
* 2FA: Egy TOTP (Time-based one-time) algoritmus segítségével tudnánk generálni egy első jelentkezési karaktersort, ehhez ugyan módosítani kéne az oldal jelenlegi felépítését, viszont ez nagyban javítaná az oldal biztonságosságát. Ezt a jelszót kiküldhetnénk a felhasználó email címére, hogy meg tudja adni első bejelentkezéskor. Egy gyakran használt könyvtár az OTP.js, ez együtt működhet akar a Google-Authentikátorral is.
* A megjelenített sablonok: A feltöltött sablonok jelen pillanatban minden felhasználó számára megjelennek, implementálni lehetne egy opciót, hogy publikus vagy privát sablont szeretnénk feltölteni, amennyiben publikus sablont töltünk fel a sablon mellett megjelenhetne a felhasználó, akinek a sablont köszönhetjük, privát sablon esetén pedig ennek a helyén megjelenhetne egy ezt jelző szöveg.
* Barát lista: A barát listához szükségünk lenne egy jobb chat létrehozó menüre, viszont ennek a két dolognak az implementálása nagyon sokat hozzátehetne az oldal felhasználói élményéhez.
* Kész mém küldése: A chatben lehetne lehetőség nem sablon alapján készült mémet küldeni egy egyszerű képfeltöltéssel.
* A képmanipulálás fejlesztése: a JIMP segítségével nagyon sokféleképp tudnánk manipulálni a képet, ezeket implementálhatnánk frontend oldalon több szabadságot adva a felhasználónak.

# 6 Köszönetnyilvánítás