Pannon Egyetem

Műszaki Informatikai Kar

[Tanszék neve]

[Szak neve]

**SZAK-/DIPLOMADOLGOZAT**

**[Dolgozat címe]**

**[Név]**

Témavezető: [Témavezető neve]

Külső/belső konzulens: [Konzulens neve]

[évszám]

Témakiírás

A szkennelt formában megkapott témakiírás beillesztése a dolgozatba.



Hallgatói nyilatkozat

Alulírott <<hallgató neve>> hallgató kijelentem, hogy a dolgozatot a Pannon Egyetem <<tanszék neve>>én készítettem a <<végzettség>> végzettség megszerzése érdekében.

Kijelentem, hogy a dolgozatban lévő érdemi rész saját munkám eredménye, az érdemi részen kívül csak a hivatkozott forrásokat (szakirodalom, eszközök stb.) használtam fel.

Tudomásul veszem, hogy a dolgozatban foglalt eredményeket a Pannon Egyetem, valamint a feladatot kiíró szervezeti egység saját céljaira szabadon felhasználhatja.

Dátum: Veszprém, [év hónap nap]

*<<hallgató neve>>*

Témavezetői nyilatkozat

Alulírott <<témavezető neve>> témavezető kijelentem, hogy a dolgozatot *<<*hallgató neve*>>* a Pannon Egyetem <<tanszék neve>>én készítette <<végzettség>> végzettség megszerzése érdekében.

Kijelentem, hogy a dolgozat védésre bocsátását engedélyezem.

Dátum: Veszprém, [év hónap nap]

*<<témavezető neve>>*

Köszönetnyilvánítás

A hallgató köszönetet nyilvánít mindazoknak, akiktől (elméleti, gyakorlati, erkölcsi stb.) segítséget kapott.

Tartalmi összefoglaló

Tartalmi összefoglaló magyarul. Az összefoglalónak tartalmaznia kell (rövid, velős és összefüggő megfogalmazásban) a következőket:

* téma megnevezése,
* megoldott feladat megfogalmazása,
* megoldási mód,
* elért eredmények,
* kulcsszavak (4-6 darab)
* terjedelme nem lehet több 1 A4-es oldalnál.

Az összefoglalót magyar és angol nyelven kell készíteni. Sorrendben a dolgozat nyelvével megegyező kerül előrébb. A cím Title stílusú, formázása: Times New Roman, nagybetű, 14 pt, félkövér, középre igazított; az összefoglaló Normál stílusú, formázása: Times New Roman, 12 pt, sorkizárt, 1.5-ös sortávolság.

**Kulcsszavak:** [4-6 kulcsszó felsorolása, vesszővel elválasztva]

Abstract

Abstract in English

**Keywords:** [list 4-6 keywords]

Tartalomjegyzék

[Jelölésjegyzék 9](#_Toc100675152)

[1. Bevezetés 10](#_Toc100675153)

[2. Versenytárs elemzés 11](#_Toc100675154)

[3. Saját megoldás ismertetése 12](#_Toc100675155)

[3.1. Backend 12](#_Toc100675156)

[3.2. Adatbázis 12](#_Toc100675157)

[3.3. Frontend 12](#_Toc100675158)

[3.4. Proxy 13](#_Toc100675159)

[4. Felhasznált technológiák 14](#_Toc100675160)

[4.1. Mangadex API(Külső API) 14](#_Toc100675161)

[4.2. Backend 14](#_Toc100675162)

[4.2.1. NodeJS 14](#_Toc100675163)

[4.2.2. Express 15](#_Toc100675164)

[4.2.3. MySQL(Node package) 15](#_Toc100675165)

[4.2.4. JSON Web Token (JWT) 16](#_Toc100675166)

[4.2.5. Socket Io 17](#_Toc100675167)

[4.2.6. Nodemailer 18](#_Toc100675168)

[4.2.7. Bcrypt(Node Package) 18](#_Toc100675169)

[4.2.8. Node Cron 19](#_Toc100675170)

[4.2.9. Uuid 19](#_Toc100675171)

[4.3. Adatbázis 20](#_Toc100675172)

[4.3.1. MariaDB 20](#_Toc100675173)

[4.4. Frontend 21](#_Toc100675174)

[4.4.1. React 21](#_Toc100675175)

[4.4.2. Redux 21](#_Toc100675176)

[4.4.3. Axios 21](#_Toc100675177)

[4.4.4. Socket Io Client 21](#_Toc100675178)

[4.4.5. Typescript 21](#_Toc100675179)

[4.4.6. Classnames 21](#_Toc100675180)

[4.5. Proxy 21](#_Toc100675181)

[4.5.1. http-proxy-middleware 21](#_Toc100675182)

[5. Specifikáció 21](#_Toc100675183)

[5.1. Funkcionális követelmények 21](#_Toc100675184)

[5.2. Nem funkcionális követelmények 21](#_Toc100675185)

[5.3. Rendszer architektúra 21](#_Toc100675186)

[5.3.1. Kommunikációk rétegek között 21](#_Toc100675187)

[5.3.2. MangaDex API 21](#_Toc100675188)

[5.3.3. Frontend 22](#_Toc100675189)

[5.3.4. Backend 22](#_Toc100675190)

[5.3.5. Adatbázis 22](#_Toc100675191)

[5.3.6. Proxy 22](#_Toc100675192)

[6. Tesztek 22](#_Toc100675193)

[6.1. Backend 22](#_Toc100675194)

[6.2. Frontend 22](#_Toc100675195)

[7. Továbbfejlesztési lehetőségek 22](#_Toc100675196)

[8. Telepítési útmutató 22](#_Toc100675197)

[9. Felhasználói dokumentáció 22](#_Toc100675198)

[Irodalomjegyzék 23](#_Toc100675199)

[Mellékletek 24](#_Toc100675200)

[Ábrajegyzék 25](#_Toc100675201)

[Táblázatjegyzék 26](#_Toc100675202)

Jelölésjegyzék

|  |  |
| --- | --- |
| AI: | Artificial Intelligence (Mesterséges Intelligencia) |
| GPU: | Graphical Processing Unit (Grafikus Processzor / Grafikus Feldolgozó Egység) |
| API: | Application Programming Interface (Alkalmazásprogramozási Felület) |
| CPU: | Central Processing Unit (Központi Feldolgozó Egység / Processzor) |
| GUI: | Graphical User Interface (Grafikus Felhasználói Felület) |
| HCI: | Human Computer Interaction (Ember-gép kapcsolat) |
| CIS: | Cognitive Information System (Kognitív információs rendszer) |

# Bevezetés

Azért választottam ezt a témát mert szakdolgozat téma választás idejében találtam rá a MangaDex API-ra ami lehetővé teszi hogy saját manga olvasó programot/weboldalt hozhassunk létre, és mivel szeretem a mangákat meg a webfejlesztés témaköre is érdekelt, úgy gondoltam hogy tökéletes szakdolgozat téma lenne egy saját közösségi weboldalt készíteni ami a mangák körül forog.

A weboldal főleg olyan embereknek készült akik nem csak mangákat szeretnének olvasni, hanem más, manga iránt érdeklődő emberekkel szeretnének találkozni, beszélgetni.

# Versenytárs elemzés

Nagyjából 3 fajta manga olvasó weboldal létezik az interneten

* Kiadó által üzemeltetett oldalak
* Fordító oldalak
* Gyűjtő oldalak

A kiadó által működtetett, és a fordító oldalakkal nem fogunk most foglalkozni mert bár lehet rajtuk mangákat olvasni, vannak hátrányaik mint például a hiányzó közösségi funkciók(Ez egy univerzális hiányosság a manga olvasó oldalak többségénél), a fordítói oldalaknál a kevés kínálat, a kiadói oldalaknál az előfizetés és néhány esetben a kevés kínálat.

A gyűjtő oldalak többsége nagy kínálatot biztosít és legtöbbször reklámokkal/adományokkal biztosítják fennmaradásukat. Ezek az oldalak az esetek túlnyomó részében web scraper megoldással különböző oldalakról(Fordító oldalakról, más manga kiszolgáló oldalakról) szedik össze a mangákat és adják át az olvasóknak.

A MangaDex(az az oldal ami a szakdolgozatban elkészített weboldalhoz biztosítja a manga api-t) is gyűjtő oldalnak mondható de ők a manga szolgáltatáshoz nem web scraper megoldással szerzik a mangákat, hanem saját szerverükre töltik fel a mangákat amiket a fordítók nyújtanak így is kimélik a fordító oldalakat a fölösleges hotlinkingtől.

A manga olvasó weboldalak többsége a fejezetek alá való kommentelésen(gyakoribb), fórumokon(ritkább) kívül nem rendelkezik közösségi funkciókkal és inkább más platformokon(Discord vagy IRC-ken) folytatnak aktív beszélgetéseket. De a közösségi funkciókon kívül szinte az összes manga olvasó weboldalon lehet ugye mangákat olvasni, saját listákat létrehozni ahová gyűjthetünk például olyan mangákat amik érdekelnek minket, így könnyebben lehet emlékezni hogy miket olvasunk.

# Saját megoldás ismertetése

## Backend

Backend szerverhez Node.js-t használom ami egy javascript futási környezetet biztosít, szóval lehet javascript kódokat lefuttatni a böngészőn kívül is.

A projetkhez használt backendnek fő feladata a közösségi funkciók szolgáltatása a frontendnek, de 2 fajta módon szolgálja ki a klienst:

* REST API
* Valós idejű szerver-kliens kommunikációval

A valós idejű szerver-kliens kommunikáció biztosítja a barátok közötti beszélgetés lehetőségét, míg a REST API http végpontokkal biztosítja az adatok szolgáltatását és változtatásának a lehetőségét.

Az api végpontjainak kezelésére Express.js-t használok, a szerver-kliens valós kommunikációhoz pedig socket.io-t.

## Adatbázis

Az adatok tárolására MariaDB-t használok ami egy relációs adatbázis. A backend és az adatbázis egy Node.js package(mysql) segítségével kommunikál.

## Frontend

A weboldalhoz React-ot használok mert az a fajta modularitás amit a React nyújt, átlátható és könnyen kezelhető kódot biztosít komponensek formájában.

A frontenden adatok tárolására React Redux ot használok, ami egy globális állapotban(Store) tud tárolni adatokat, amiket a program bármelyik komponenséből elérhetünk, és könnyen lehet követni a Store-ban történő változásokat, ami elősegíti az debugolást.

## Proxy

Magához a weboldalhoz még használok egy proxy szervert is ami ugyan úgy Node.js-ben fut Expressjs és egy Node.js package(http-proxy-middleware) segítségével. Ez a proxy a frontendről érkező azon kéréseket kapja, amik a külső api-al akarnak interaktálni.

Hogy miért használok proxyt? Elsősorban azért hogy a MangaDex-en lévő képeket legyen az Manga borítókép vagy a manga fejezetek paneljeit, letöltse saját tárhelyre hogy elkerüljük a hotlinkelést, szóval ha valaki mangát olvas a mi oldalunkon, akkor a képeket nem a MangaDex szerveréről fogja megkapni, hanem saját szerverről így csökkentjük a MangaDex szerverén a terhelést. A másik indok amiért proxyt használok, hogy elrejtsem az api útvonalát. Míg én most csak útvonal elrejtésre használom, a proxy használata tökéletes arra hogy biztonságos helyen(saját szerveren) tároljuk külső api-khoz szükséges hitelesítő kulcsokat, így nem áll fennt annak a veszélye hogy kikerül a nagyvilágba.

# Felhasznált technológiák

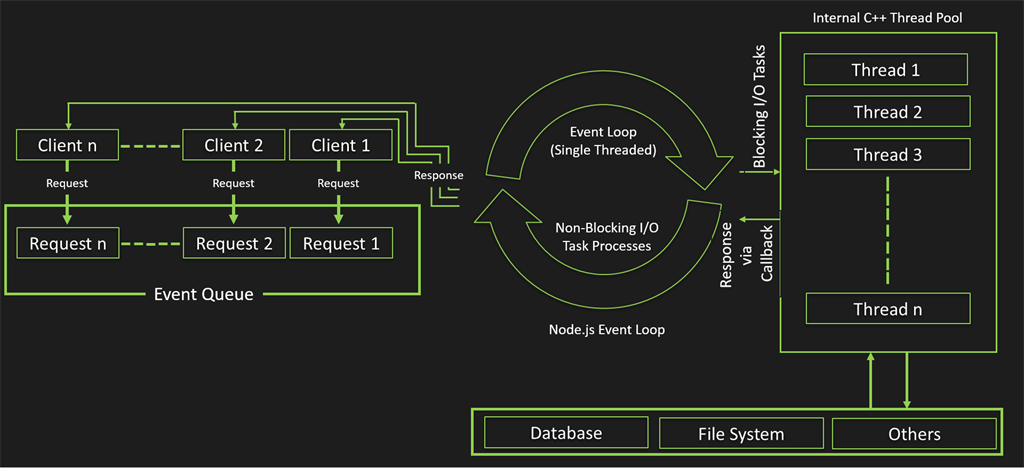
## Mangadex API(Külső API)

A MangaDex API lehetőséget biztosít arra, hogy a MangaDex hatalmas manga tárházával interaktálni tudjunk, legyen az manga információk lekérdezése, új manga feltöltése, fordító csapatok létrehozása és még sok más. Az adatokat HTTP API módon szolgáltatják, tehát URL címekre történő http requestekkel lehet lekérni, módosítani, feltölteni és törölni a kívánt adatokat.

## Backend

### NodeJS

A Node.js egy cross-platform futási környezetet biztosít ami lehetővé teszi a jól skálázható szerverek építését Javascript nyelvben. Node a kapcsolatokat „elméletben” 1 szálon dolgozza fel aszinkron módon, gyakorlatban a fő szál(Event Loop) más szálaknak(Workers) osztja ki a feladatokat egy event sorból amiket ők elvégeznek a megfelelő C++ megvalósítással párhuzamosan vagy aszinkron módon és vissza küldik az eredményt a fő szálnak, amit vissza adhat a kérést kezdeményezett kapcsolatnak



<https://www.c-sharpcorner.com/article/node-js-event-loop/>

A Node.js legnagyobb előnye a hatékony Input/Output kezelés mint például a fájlrendszerre való írás/olvasás vagy a hálozatról érkező adatok olvasása, a nehézkes szálak kezelésével. De mivel már elég népszerű lett ez a futási környezet ezért az interneten megtalálható nagyon sok segítség, és mivel javascriptet használ, ezért a frontend fejlesztők is könnyen beléphetnek a backend világába. Előnyhőz lehetne sorolni a Node.js csomagkezelőjét(Package manager) a Node Package Managert(npm), mert ennek segítségével nagyon hamar adhatunk hozzá a programunkhoz olyan kódokat amik megkönnyítik munkákat mert nem kell saját kódot írnunk hanem más, az esetek 99%-ban jobban megírt megvalósítását használhatjuk.

Viszont egyik hátránya a Node.js nek hogy a CPU intenzív feladatokkal nem tud olyan jól megbírkózni, mint a fennt említett I/O feladatokkal, mert a azokat a műveleteket amik a CPU-t veszik igénybe szinkron módon futnak, míg az I/O műveletek aszinkron módon.

### Express

Express.js egy framework(Nodejs package) amivel könnyen és gyorsan lehet NodeJS web applikációt, API-t létrehozni. A NodeJS alap http/https moduljait használja fel. Maga az Express sok „opcionális” middlewarel rendelkezik amiket külön lehet telepíteni, vagy saját middlewaret is írhatunk amik kibővítik az API vagy web applikációnak a funkcióit, és mivel ezeket a middlewareket külön lehet megszerezni és nem az Express alap csomagjában jönnek elég lightweight és könnyen testreszabható a web applikációnk vagy API-unk.

### MySQL(Node package)

Ez a Node Package egy driver-t biztosít amivel lehet csatlakozni egy MySQL adatbázishoz(Mi esetünkben egy MariaDB-hez de a MariaDB és a MySQL szinte azonosak), és egy olyan interfacet biztosít amivel könnyen lehet az adatbázis felé lekérdezéseket indítani. Ez nem egy ORM, vagyis az SQL utasításokat nekünk kell teljes egészében megírni és lefuttatni.

Ez a Driver lehetőséget ad több egy idejű csatlakozás biztosítására, hiba kezelésre, adat escapelésre hogy megvédjük szerverünket SQL injectionöktől, és az adatbázisban felmerülő típusokat SELECT query esetén Javascript típusokba kasztolja és egyéni kasztolásra is van lehetőség

### JSON Web Token (JWT)

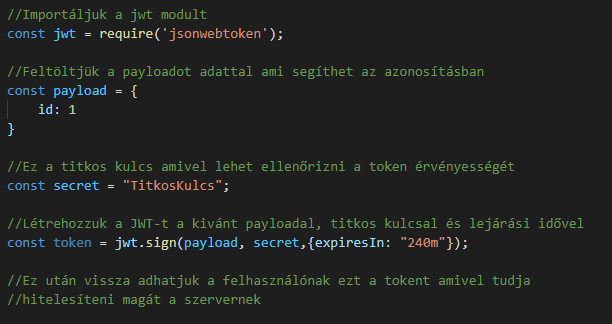
#### JSON Web Token

A JWT módot biztosít arra hogy kettő vagy több fél között biztonságosan tudjon adatokat küldeni. Ez a JWT egy hosszú string ami egy JSON-t tartalmaz amibe bármi féle adat lehet, és lelehet ellenőrizni ennek a tokennek a sértetlenségét, valóságát. Ezt a tokent tovább lehet titkosítani hogy a tartalmát is teljesen biztonságba tudjuk szállítani, mert alapból a JWT csak azt tudja biztosítani hogy ha a tokent megváltoztatták, akkor érvénytelen legyen, de attól még ha valaki megszerzi a tokent, dekódolhatja és a benne lévő adatokat megszerezheti, ezért ha nem használunk további titkosítást csak olyan adatokat küldjünk amikkel a támadó nem tud mit kezdeni.

Nagyon jól lehet használni hitelesítéshez a JWT-t, például ha a felhasználó bejelentkezik, átadunk neki egy ilyen tokent amit a kliens eltárol, és az elkövetkezendő requestekhez hozzá teszi ezt a tokent és a szerveren megvizsgáljuk a tokent hogy érvényes-e, és ha az, akkor a szerver végrehajtja a requestet.

#### Jsonwebtoken(Node Package)

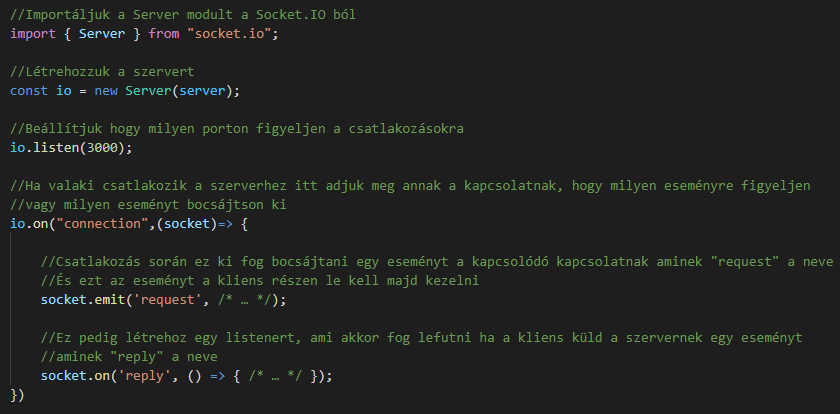
Ez a node package implementálja ezt a JWT technológiát, szóval lehet Javascriptben JWT-ket létrehozni, ellenőrizni, dekódolni. Könnyen belehet állítani a létrehozásnál hogy milyen algoritmust használjon a titkosításra, mikor járjon le az érvényessége, ki állította ki és még mást is.



### Socket Io

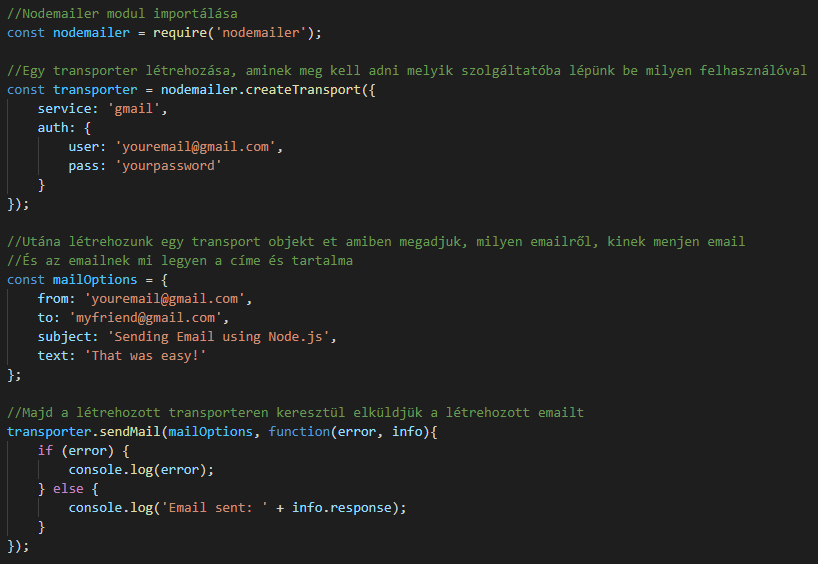
Socket.IO egy alacsony késleltetésű, kétirányú esemény alapú kommunikációs könyvtár ami lehetővé teszi a kliens és szerver közötti valós idejű kommunikációt.

A Socket.IO esemény alapú amit annyit jelent, hogy a szerver és a kliens létrehozott eseményekre figyel, vagy eseményeket bocsájt ki



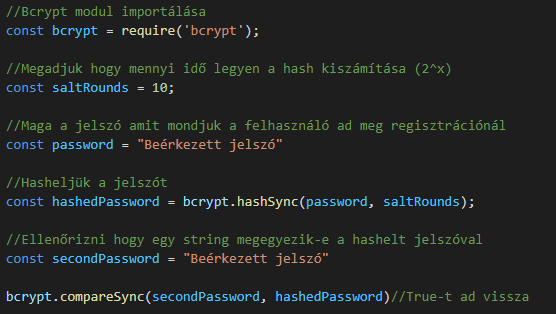
### Nodemailer

A Nodemailer egy olyan modul ami lehetővé teszi az e-mail-ek küldését a szerverről. Lehet saját email szerverről(Simple Mail Transfer Procotol – SMTP-vel) és más email szolgáltatásokkal(például: Gmail) emailt küldeni.



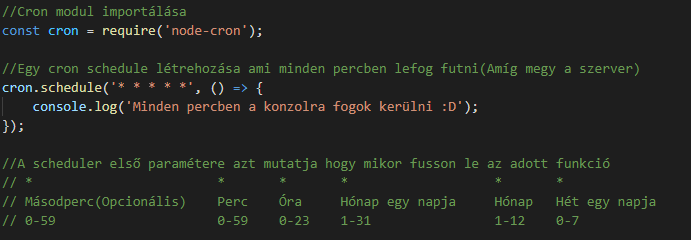
### Bcrypt(Node Package)

Maga a Bcrypt az egy jelszó hashing funkció ami egy elég megbízható titkosítást biztosít a jelszavaknak. Én a bcrypt Javascript implementálását használom.



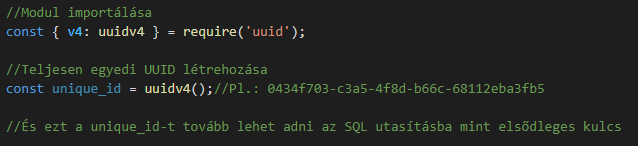
### Node Cron

Node Cron lehetővé teszi hogy bizonyos időpontokban lefuttassunk autómatikusan funkciókat. A modul szintaktikája a GNU operációs rendszer crontabján alapszik, ami mint a Node Cron, lehetővé teszi hogy bizonyos időpontban parancsok, fájlok fussanak le.



### Uuid

A Universal Unique Identifier(UUID) egy nagy valószínűséggel teljesen egyedi azonosító kódot jelent, amit sok különböző tényezőből például a létrehozásnál lévő időpont, hálózati cím segítségével hoz létre egy algoritmus. Én ennek az algoritmusnak a Node.Js beli implementációját használom(uuid Ami a node package nek a neve) az adatbázisban tárolt pár adatnak az azonosításához.



## Adatbázis

### MariaDB

MariaDB egy open source relációs adatbázis, eredetileg a MySQL egy továbbfejlesztett változata. Nagyon magas a MySQL-el a kompatibilitása, ez látható abban is hogy a backend szerveren én igazából egy MySQL drivert használok a MariaDB eléréséhez és minden jól működik és általában jobban skálázhatóbb és gyorsabb mint mondjuk a MySQL.

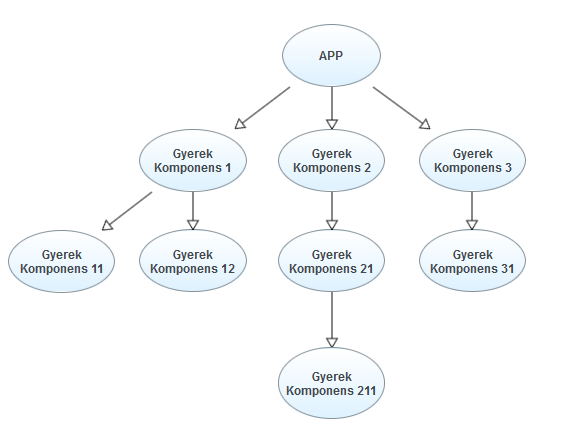
## Frontend

### React

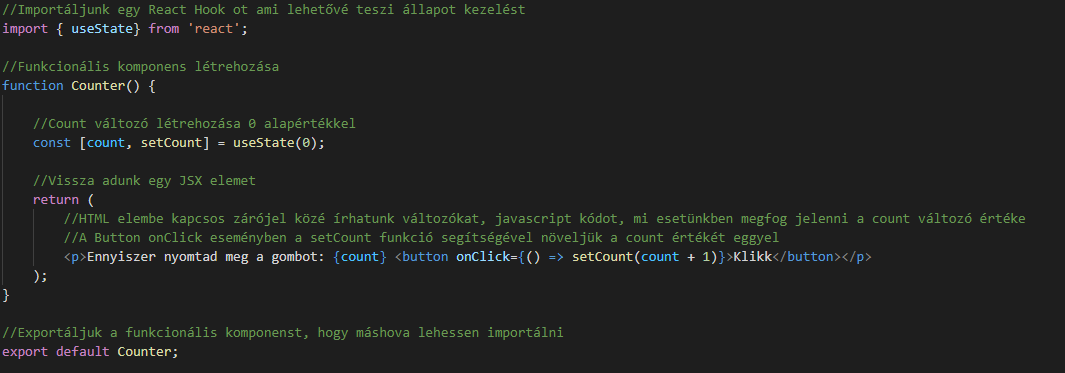
React az egy javascript könyvtár felhasználói interfacek létrehozásához. Komponens alapú ami annyit jelent hogy valamilyen logikát, html-elemeket egy komponensbe összegyűjt, és lehet meghívni/felhasználni a kívánt helyen.

Ezeknek a komponenseknek van saját állapota is, vagyis a komponensen belül lehet azokat az adatokat tárolni, amiket felhasznál, de nem csak adatokat tárolhat hanem funkciókat is, amik megváltoztathajták a komponens állapotát, kinézetét.

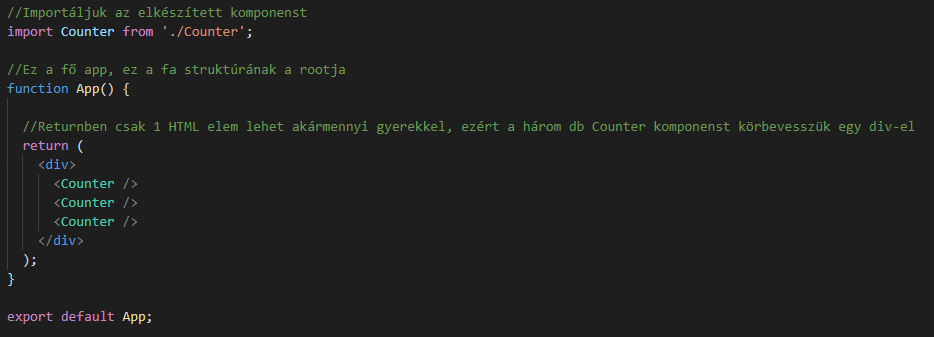
Ezek a komponensek mindig valamilyen más komponensnek a gyermekei és minden egy fő komponensből indul ki(gyökér komponens) és fa struktúrában helyezkednek el.



Egy gyors példa a React komponens rendszerére



A fenti kód létrehoz egy komponenst aminek saját állapota van és azt kezeli(count változó).



Itt pedig az elkészített komponenst importáljuk az App komponensbe, ami az applikációnak a gyökér eleme.



És ha elindítjuk a kliens szervert, lehet látni a 3 komponenst amit a gyökér komponensbe beletettünk és ha rákattintunk az egyik klikk gombra, akkor a mellette lévő számláló növekedni fog.

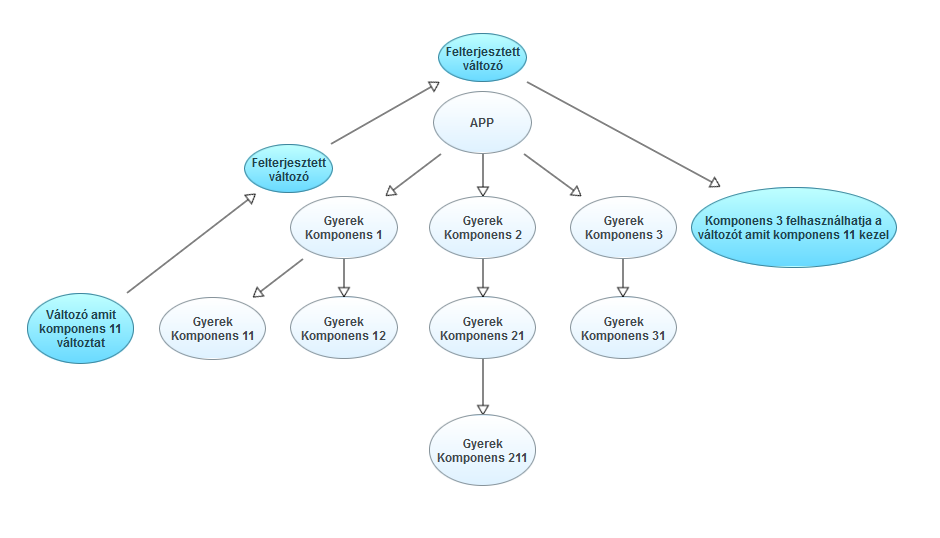
Ez csak egy bevezetés volt a React komponens rendszerébe, de sokkal több funkciót is ellehet vele végezni, mint például feltételes megjelenítést, ami annyit jelent hogy valamilyen feltételek hatására, a komponens mást fog megjeleníteni.

### Redux

Maga a Redux globális állapot tárolásra használható, amivel könnyen lehet úton követni a programnak a változásait és debuggolni. Ez bármilyen Javascript applikációhoz használható, de a React-hoz van egy hivatalos React Redux változat is, ami lehetővé teszi a komponenseknek a store-al(Az a hely ahol tároljuk az applikációnak az adatait, állapotát) való kommunikációt.

Redux alapból immutable(változhatatlan) vagyis nem direkt változtatjuk az adatok állapotát, hanem egy másolatot készítünk az adott állapotról amit változtathatunk és utána a másolattal felülírjuk az eredeti állapotot

Hogy miért kell egy globális állapot manager? Azért, mert a React-ban a komponensek fa struktúrában helyezkednek el és ha egy komponensből átakarunk vinni egy másik komponensbe egy adatot, akkor azt felkell terjeszteni a szülőbe, ami majd a másik komponensnek átadja az adatot, ez a megoldás kicsi utakon, kicsi projektben még használható, de ha nagy a projekt, vagy az út amit megkell tennie az adatnak, akkor jobban járunk használjuk ezt a globális állapot megoldást.

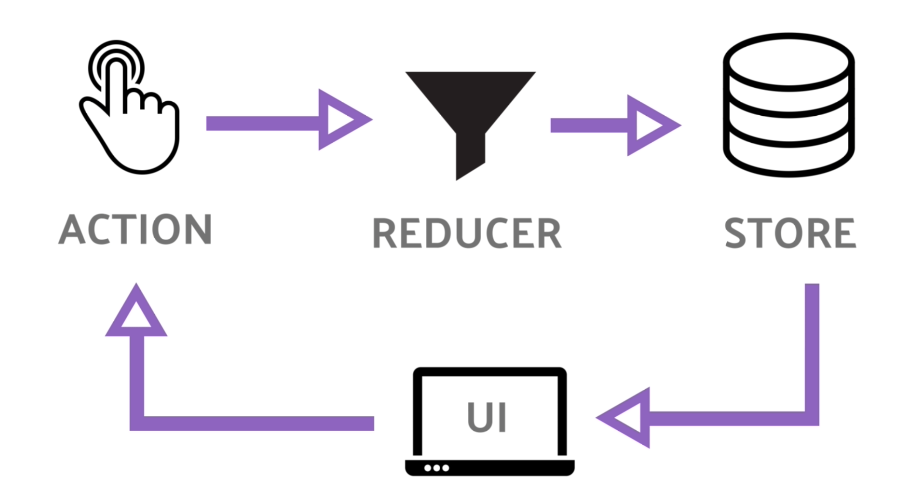


Ahogy a példán látható a gyerek komponens(gyk) 11 ben változtatjuk a változó értékét, de hogy elérje egy másik komponens, felkell terjesztenünk a szülőbe ezt a változót, ami igazából annyi jelent hogy a szülő komponens ben is létezik ez a változó, de a gyk-nak továbbadjuk ezt a változót, és a változtatáshoz szükséges metódusokat. Ez a megoldás az érintett komponenseken végig viszi a változót, míg a Redux segítségével egy közös, globális helyen kezelhetjük azokat az adatokat, amiket 1 vagy több komponens is használni akar.

Redux 3 fő elemből áll

* Store
* Action
* Reducer

A store tartalmazza az adatokat, és a User Interface innen szerzi az adatokat amiket megjeleníthet vagy feldolgozhatja. Az action az igazából egy funkció amit meghívhatunk, adhatunk neki adatot amit feldolgoz és a legfontosabb része az hogy a reducer-nek átadja ezt az adatot. A reducer meg a storeban lévő adatokat változtatja meg legtöbbször az action-től kapott adatra.



A Redux adat folyama a fönti ábrán látható módon működik. A felhasználói felületen autómatikusan vagy valamilyen esemény hatására, meghívunk egy actiont, ami valamilyen adat feldolgozást csinálhat(pl.: kapott adat + 1) és utána az action akár több reducer-t is meghívhat amik megváltoztatják a storeban lévő értékeket, amiknek a változása megjelenik a felhasználói felületen, ha éppen megjeleíti azt.

### Axios

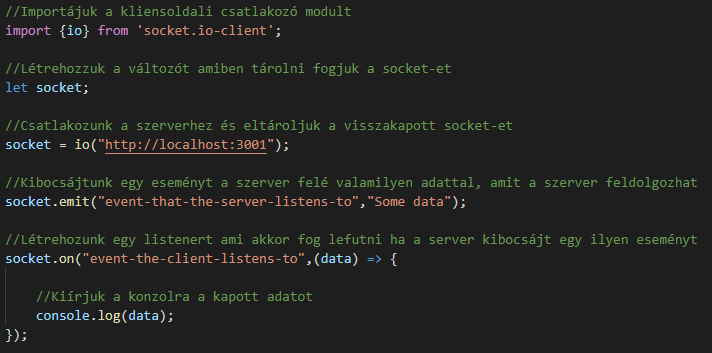
Axios az egy promise(ígéret) alapú http kliens amivel lehet http kéréseket küldeni böngészőből és nodejs alkalmazásból. Javascriptnek a Promise API-ját használja ezért az aszinkron kéréseket lehet kezelni .then és .catch megoldással, vagy az await operátorral hogy csak akkor történjen adat felhasználás, ha a kérésre érkezett válasz.

Van még több funkciója is a http kérések indításán kívül például a kimenő kéréseket interceptálja és új adatokat ad hozzá a kéréshez pl.: Minden kéréshez hozzáadja a Header-be az authentikációhoz szükséges tokent.

### Socket Io Client

Ez a szerveren használt socket es kommunikációnak a kliens oldali fele, itt is egy socketen lehet létrehozni eseményeket amire figyel és lefutnak azok a funkciók amiket akarunk ha a szerverről érkezik egy olyan esemény vagy lehet eseményeket küldeni a szerver felé.

Ahhoz hogy ezt a socket-et megszerezzük ami tartalmazza az információkat ahhoz hogy a szerverrel kommunikáljunk csatlakozni kell a szerverhez és a visszakapott socket-et tároljuk.



### Typescript

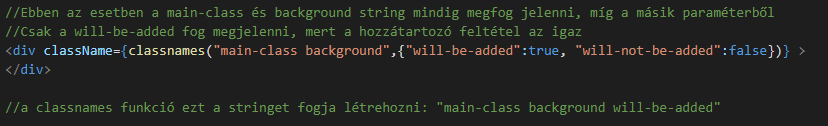
TypeScript az egy programozási nyelv, igazábol javascript csak lehetővé teszi hogy típusokat használjunk amik csökkenthetik a javascript gyengén típusos szerkezetéből adandó hibákat.

Lehetővé teszi még hogy a funkcióknak megadjuk milyen milyen típusú paramétereket várjon, milyen típusú értéket adjon vissza. Lehet névtereket(namespace), enumerátorokat és saját objektum sablonokat is könnyen létrehozni. A saját készítésű típusokat például enumerátorok, objektum sablonokat is lehet hozzárendelni változókhoz, függvényekhez.

A TypeScript azért is jó mert compileolásnál ellenőrzi hogy az adott típusok jó fajta adatot kapnak, és ha érzékeli hogy valamilyen változó rossz fajta értéket kap, vagy olyan funkciót akarunk elvégezni rajta amit a típusán nem lehetne, azt jelzi a TypeScript

### Classnames

Ez egy Javascript modul amit html elemek osztályneveinek feltételes összekapcsolás könnyítésére hozták létre. Igazából ez a modul csak annyit csinál, hogy a megadott stringekből létrehoz egy új stringet amivel a html css osztályokat lehet hozzárendelni egy html elemhez. A feltételes összekapcsolást a stringekhez tartozó feltétel típus kiértékelésével végzi el, vagyis ha egy stringhez tartozik egy feltétel akkor az akkor fog megjelenni a végső stringbe ha a feltétele igaz.



## Proxy

### http-proxy-middleware

Ezzel a Node.JS modullal létre lehet hozni egy proxy middleware-t amit egy http proxy működtetésére lehet felhasználni. Egy http proxy kettő vagy több szolgáltatás között létezik, és rajta keresztül kommunikálnak ezek a résztvevők.

Főként a kliens és a szerver között helyezkedik el, ami annyit jelent hogy a kliensről érkező kéréseket továbbítja a megfelelő szolgáltatásnak, ami lehet saját szerver vagy külső szolgáltatás is(Valamilyen API).

De a proxy végezhet valamilyen műveletet is ezekkel a kérésekkel mielött továbbadja, lehet az logolás, érvényesség ellenőrzése, header vagy body módosítása. De fellehet használni akár arra is, hogy megfelelő módon elossza a terhet a különböző szerverek között, hogy ne terhelődjenek aránytalanul.

Az applikációnkat biztonságosabbá is lehet tenni azzal hogy elrejtjük azokat az erőforrásokat amiket felhasználunk, így nem fog kikerülni a nagyvilágba hogy milyen szolgáltatásokat használunk, azokat hol lehet elérni és a legfontosabb, hogy a szolgáltatáshoz használt kulcsokat, authentikációhoz adatokat nem a kliensen tároljuk hanem a proxy-n amihez a kliens nem fér hozzá.

# Specifikáció

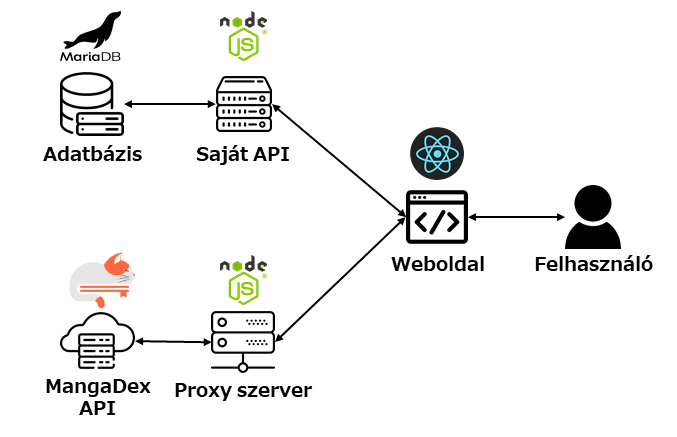
## Funkcionális követelmények

* Mangák keresése
* Mangák olvasása
* Mangák értékelése
* Manga írók, fordítók megjelenítése
* Mangák gyűjteményekbe gyűjtése
* Manga alá kommentelés
* Manga fejezet alá kommentelés
* Fórum thread létrehozása
* Fórum thread alá kommentelni
* Kommentek értékelése
* Bejelentkezés
* Regisztráció
* Profil adatok változtatása
* Barátok felvétele
* Barátokkal chatelés
* MangaDex-ről hotlinkelés elkerülése

## Nem funkcionális követelmények

* Mangákat MangaDex API használatával lekérdezni
* Backend szerver Node.Js API szerver
* Adatbázis MySQL(MariaDB)
* Frontend React-ban íródjon

## Rendszer architektúra



A weboldalról két felé lehet kéréseket indítani, a MangaDex felé és a saját API felé. A MangaDex felé olyan kérések mennek amik a mangákkal kapcsolatos például keresés, olvasás, míg a saját API felé főként a szociális funkciókért indít kéréseket az oldal például bejelentkezés, kommentelés, chat barátokkal.

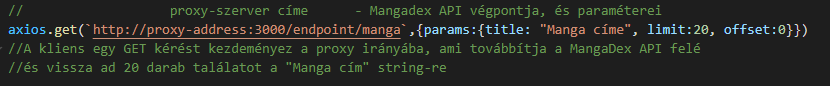
### Kommunikációk rétegek között

Legtöbbször a rétegek közötti kommunikációt a kliens kezdeményezi, ez lehet felhasználó által indított esemény miatt, vagy autómatikus, például Manga keresésnél a felhasználó beír egy címet amire keresni akar, vagy maga az oldal a betöltésnél http kéréseket küld a MangaDex API felé hogy kapjon Manga részleteket, borítóképet amit megjelenít a felhasználói felületen.

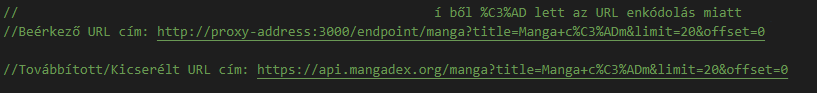
/\*Azért legtöbbször a kliens kezdeményezi a kommunikációt, mert a chat esetében a beérkező üzenet a szervertől jön, igaz, hogy ahhoz hogy beérkező üzenet jöjjön, egy másik kliensnek ezt kezdeményeznie kell, de a vizsgált architektúrában nincs jelen más felhasználó.\*/

#### Frontend és MangaDex API között

A frontend és a külső API között a kommunikáció általában úgy halad, hogy a proxy szerverre küldünk egy http requestet az Axios node csomag segítségével úgy hogy a proxy URL után a megfelelő MangaDex API végpont címet, és megfelelő paramétereket írjuk.



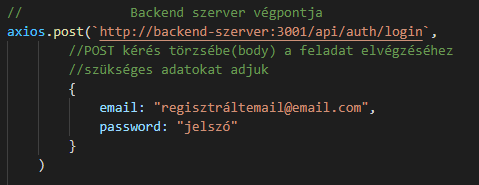
Miután elküldjük a kérést a proxynak, a proxy a beérkező URL-nek a proxy címét lecseréli a megfelelő MangaDex API szerver címére.



Majd a továbbított URL címről visszakapott választ visszaadja a kliensnek, ami megfelelően feldolgozza azt.

#### Frontend és Backend API között

A frontend és a saját backend API közötti kommunikáció annyiban egyszerűsödik, hogy nem megy keresztül egy proxyn az egész folyamat, hanem egyből a szerverre küldi a kéréseket.



És miután megkapja a kliens a kérés eredményét, megfelelően feldolgozza azt.

#### Frontend és Backend Socket között

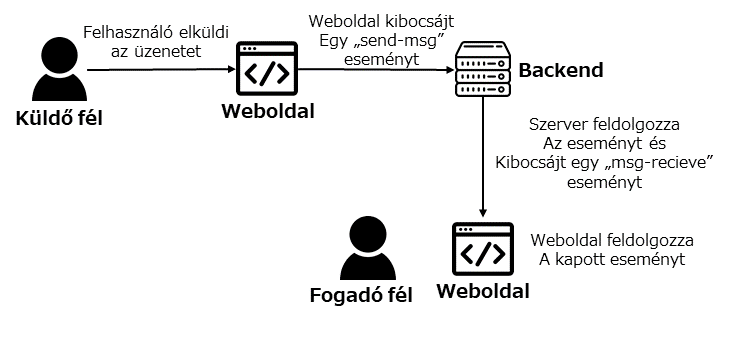
A kliens és backend nem csak http API módon kommunikál hanem egy kétfolyamú valós idejű socket kommunikációt biztosít kliens és szerver között. Ezt a fajta kommunikációt a Socket.IO node csomag biztosítja.

A mi alkalmazásunkban akkor történik kommunikáció a szerverrel amikor a felhasználó bejelentkezik és csatlakozik a szerverhez, amikor a kliens üzenetet kap, és amikor a kliens üzenetet küld.

Csatlakozás során a kliens küld egy „add-user” eseményt a szerver irányába, amit az lekezel és hozzáadja a kapcsolatot egy globális listához ami az aktív felhasználókat tartja nyílván.

Amikor üzenetet küld a kliens, akkor a szervernek küld egy „send-msg” eseményt, amit a szerver feldolgoz és továbbítja a megfelelő kliensnek.

Üzenetet akkor kap egy kliens, ha egy másik kliens küld neki egy üzenetet. A másik kliens küldött üzenetét a szerver feldolgozza, és továbbítja a megfelelő kliensnek az üzenetet egy „msg-recieve” esemény kibocsátásával, amit a fogadó kliens megkap, és feldolgozza az üzenetet



### MangaDex API

### Frontend

### Backend

### Adatbázis

### Proxy

# Tesztek

## Backend

## Frontend

# Továbbfejlesztési lehetőségek

# Telepítési útmutató

# Felhasználói dokumentáció

Irodalomjegyzék

|  |  |
| --- | --- |
| [1] | T. De Smedt and W. Daelemans, “Pattern for python.,” *The Journal of Machine Learning Research,* vol. 13, no. 1, pp. 2063-2067, 2012. |
| [2] | „Záróvizsga információk,” [Online]. Available: https://mik.uni-pannon.hu/index.php/hu/oktatas/zarovizsga.html. [Hozzáférés dátuma: 04 03 2022]. |
| [3] | D. J. Wetherall és A. S. Tanenbaum, Computer networks, Pearson Education, 2013. |

Mellékletek

Mappaszerkezet

+chatbot

| backen.bat

| backend.py

| files.doc

| fixedlinks.json

| ipcheck.py

| linkek.json

| linkfix.py

| log.txt

| sqlwriter.py

| textprocessor.py

|

+---backend

| pyvenv.cfg

+---lara

| | entities.py

| | nlp.py

| | parser.py

| | stemmer.py

+---static

| +---css

| | chat.css

| |

| \---js

| chat.js

|

+---templates

| index.html

|

[PÉLDA!!! Megjegyzés: A Python csomagkezelője által telepített fájlok, illetve a különböző cache fájlok a fenti listából kimaradtak, mivel ezekkel indokolatlanul és aránytalanul hosszú lenne a fenti felsorolás. A beadott fájlok között azonban a teljesség kedvéért szerepelnek ezek a fájlok is.]

Ábrajegyzék

[1. ábra: ábrafelirat (ha szükséges, akkor a forrás megjelölésével) [1] 1](#_Toc97890941)

Táblázatjegyzék

[1. táblázat (forrás megjelölésével) [2] 1](#_Toc97890942)