FELADATKIÍRÁS

A feladatkiírást a **tanszék saját előírása szerint** vagy a tanszéki adminisztrációban lehet átvenni, és a tanszéki pecséttel ellátott, a tanszékvezető által aláírt lapot kell belefűzni a leadott munkába, vagy a tanszékvezető által elektronikusan jóváhagyott feladatkiírást kell a Diplomaterv Portálról letölteni és a leadott munkába belefűzni (ezen oldal HELYETT, ez az oldal csak útmutatás). Az elektronikusan feltöltött dolgozatban már nem kell megismételni a feladatkiírást.



Budapesti Műszaki és Gazdaságtudományi Egyetem

Villamosmérnöki és Informatikai Kar

Sereg András

Testre szabható koktéladatbázis készítése

React.js és Node.js segítségével

Konzulens

Kövesdán Gábor

BUDAPEST, 2024

Tartalomjegyzék

[Összefoglaló 6](#_Toc166706036)

[Abstract 7](#_Toc166706037)

[1 Bevezetés 8](#_Toc166706038)

[2 Felhasznált technológiák 9](#_Toc166706039)

[2.1 Backendhez használt technológiák 9](#_Toc166706040)

[2.1.1 Node.js 9](#_Toc166706041)

[2.1.2 MongoDB 9](#_Toc166706042)

[2.2 Frontendhez használt technológiák 9](#_Toc166706043)

[2.2.1 React 9](#_Toc166706044)

[2.2.2 PrimeReact 10](#_Toc166706045)

[2.2.3 React Redux 10](#_Toc166706046)

[2.2.4 React Router 10](#_Toc166706047)

[2.2.5 axios 11](#_Toc166706048)

[2.2.6 Typescript 11](#_Toc166706049)

[2.2.7 Egyéb használt könyvtárak és bővítmények (?) 11](#_Toc166706050)

[2.3 Szolgáltatások 11](#_Toc166706051)

[2.3.1 Verziókezelés 11](#_Toc166706052)

[2.3.2 Google Maps API 12](#_Toc166706053)

[3 Hasonló alkalmazások 13](#_Toc166706054)

[3.1 Untappd 13](#_Toc166706055)

[4 Tervezés 14](#_Toc166706056)

[4.1 Architektúra 14](#_Toc166706057)

[4.2 Felhasználói szerepkörök 14](#_Toc166706058)

[4.3 Adatmodellek 15](#_Toc166706059)

[4.3.1 Ingredient 15](#_Toc166706060)

[4.3.2 Drink 15](#_Toc166706061)

[4.3.3 Pub 16](#_Toc166706062)

[4.3.4 User 16](#_Toc166706063)

[4.4 REST API végpontok 16](#_Toc166706064)

[4.4.1 REST 16](#_Toc166706065)

[4.5 Végpontok 17](#_Toc166706066)

[4.6 Fontosabb funkciók 18](#_Toc166706067)

[4.6.1 Kedvencekhez adás 18](#_Toc166706068)

[4.6.2 Listák szűrése 18](#_Toc166706069)

[5 Implementáció 19](#_Toc166706070)

[5.1 -Backend: 19](#_Toc166706071)

[5.1.1 MongoDB adatmodellek 19](#_Toc166706072)

[5.1.2 Routeok 20](#_Toc166706073)

[5.2 -Frontend: 20](#_Toc166706074)

[5.2.1 Komponensek 20](#_Toc166706075)

[5.2.2 Jogosultságok 27](#_Toc166706076)

[6 Tesztelés 28](#_Toc166706077)

[7 Összegzés 29](#_Toc166706078)

[8 Köszönetnyilvánítás 30](#_Toc166706079)

[Irodalomjegyzék 31](#_Toc166706080)

[Függelék 32](#_Toc166706081)

Hallgatói nyilatkozat

Alulírott **Sereg András**, szigorló hallgató kijelentem, hogy ezt a diplomatervet meg nem engedett segítség nélkül, saját magam készítettem, csak a megadott forrásokat (szakirodalom, eszközök stb.) használtam fel. Minden olyan részt, melyet szó szerint, vagy azonos értelemben, de átfogalmazva más forrásból átvettem, egyértelműen, a forrás megadásával megjelöltem.

Hozzájárulok, hogy a jelen munkám alapadatait (szerző(k), cím, angol és magyar nyelvű tartalmi kivonat, készítés éve, konzulens(ek) neve) a BME VIK nyilvánosan hozzáférhető elektronikus formában, a munka teljes szövegét pedig az egyetem belső hálózatán keresztül (vagy hitelesített felhasználók számára) közzétegye. Kijelentem, hogy a benyújtott munka és annak elektronikus verziója megegyezik. Dékáni engedéllyel titkosított diplomatervek esetén a dolgozat szövege csak 3 év eltelte után válik hozzáférhetővé.

Kelt: Budapest, 2024. 05. 12.

...…………………………………………….

Sereg András

Összefoglaló

//TODO Valami bevezető szöveg még

A dolgozatban bemutatom az alkalmazáshoz használt különböző technológiákat és a tervezési folyamatokat. Itt részletesebben kifejtem az összes adatbázisban megtalálható típust és a felhasználói szerepköröket, valamint a főbb funkciókat.

Nem utolsó sorban az alkalmazás implementálásáról fogok írni, részletekbe menően szó esik a React segítségével készült frontendről, és a Node.js-ben írt backendről is, valamint kitérek a tesztelésre is.

Abstract

Ide jön a ½-1 oldalas angol nyelvű összefoglaló, amelynek szövege a Diplomaterv Portálra külön is feltöltésre kerül.

//TODO

# Bevezetés

Ahány ember, annyi féle preferencia. Nincs ez másképp, ha a szórakozási vagy italfogyasztási szokásokról beszélünk. Van, hogy egyedül is nehezen tudunk dönteni arról, hogy hol és hogyan töltsük el a péntek-szombat estéinket, egy több fős társasággal pedig még nehezebb dűlőre jutni.

A dolgozatom célja, hogy egy olyan alkalmazás készítsek, mellyel a felhasználók minél több szempont szerint alakíthassák a preferenciáikat, és azok alapján megtalálják Budapest számukra legmegfelelőbb vendéglátói illetve szórakoztatói egyégét.

A dolgozatom során a 2. fejezetben ismertetem az alkalmazás elkészítése során használt technológiákat, a 3. fejezetben a már létező, az alkalmazáshoz hasonló szolgáltatásokat hasonlítom össze. A 4. fejezetben a tervezési lépéseket ismertetem, az 5. fejezetben pedig az implementáció részleteit mutatom be.

//TODO kibővíteni, ha megvan a többi fejezet

# Felhasznált technológiák

## Backendhez használt technológiák

### Node.js

A Node.js egy nyílt forráskódú, szerveroldali javascript futtatókörnyezet, ami az eseményvezérelt modelljével lehetővé teszi a hatékony és skálázható alkalmazások fejlesztését.

Egyik nagy előnye - mivel a webalkalmazások túlnyomó részének a kliens oldala is javascript nyelven íródott – hogy egységesíti a kódbázist az adott projektben, így segíti a fejlesztők közötti kommunikációt és a kód megértését mindenki számára.

Az npm (Node Package Manager) segítségével könnyen felhasználhatunk már létező könyvtárakat az alkalmazásunk fejlesztése során, ezzel megannyi funkcióhoz és modulhoz jutva, melyekkel átláthatóbb és tisztább kódot írhatunk.

### MongoDB

A MongoDB egy nyílt forráskódú, dinamikus adatmodellt alkalmazó NoSQL adatbázis. JSON-szerű dokumentumokban tárolja az adatokat, ezzel skálázhatóságot és magas teljesítményt nyújtva.

Sémamentes tervezése révén könnyű és dinamikus adatmodellezést tesz lehetővé, ami ideális félig strukturált adatok kezelésére.

## Frontendhez használt technológiák

### React

A React.js egy nyílt forráskódú Javascript könyvtár, amellyel felhasználói felületeket készíthetünk, elsősorban Single Page Application (későbbiekben SPA) formájában.

Egyik főbb jellemzője a virtuális DOM, ami lehetővé teszi a hatékony és gyors tartalomfrissítéseket az alkalmazásunkban, anélkül, hogy az oldalt újra kelljen töltenünk. Ezt úgy éri el, hogy a komponensek változásakor a DOM-nak egy absztrahált változatát módosítja, ezzel csak az érintett állapotok változnak meg

A Reactban az alkalmazások úgynevezett komponensekből épülnek fel, melyek elősegítik az újrahasználhatóságot, és könnyebben karbantarthatóvá és tesztelhetővé teszi a kódot.

Lehetőségünk van különböző Hookok használatára, aminek segítségével függvényként készíthetjük el a komponenseinket, valamint lehetőséget biztosít a fejlesztőknek hogy különválasszák a megjelenítést a működési logikát.

### PrimeReact

A PrimeReact egy nyílt forráskódú, fejlett, és könnyen testreszabható UI komponenskönyvtár, amely a React alkalmazások számára kínál modern és stílusos felhasználói felületeket. A könyvtár széles skálájú kész komponenseket tartalmaz, beleértve a gombokat, űrlapokat, táblázatokat, paneleket és sok más elemet, amelyekkel gyorsan és hatékonyan építhetünk felhasználói felületeket. A PrimeReact átfogó dokumentációt kínál a komponensekről, példákkal és használati utasításokkal, amelyek segítenek a fejlesztőknek megérteni és használni ezeket a komponenseket az alkalmazásaikban.

### React Redux

A Redux egy előre kiszámítható állapotkezelő könyvtár, amely egyetlen állapotfát használ az alkalmazás állapotának tárolására és módosítására. A React Redux pedig egy olyan könyvtár, amely lehetővé teszi a Redux állapotkezelés könnyű integrálását a React alkalmazásokba. A React Redux segítségével a komponensek egyszerűen elérhetik és módosíthatják az alkalmazás állapotát, anélkül hogy közvetlenül hozzáférnének az állapothoz vagy a Redux műveletekhez. Ezáltal a kódbázis tisztább és jobban karbantartható lesz. A React Redux további előnyei közé tartozik a könnyű tesztelhetőség és az állapotmenedzsment különböző aspektusainak egyszerű kezelése, például az aszinkron műveletek vagy a globális állapot megosztása komponensek között. Az alkalmazásban betöltött pontos szerepét később részletesebben kifejtem.

### React Router

A React Router egy népszerű routing könyvtár, amely lehetővé teszi a React alkalmazások számára a kliensoldali útvonalkezelést. A React Router segítségével könnyedén hozhatunk létre dinamikus és többoldalas alkalmazásokat, anélkül hogy újratöltenénk az oldalt. A könyvtár egyszerű és intuitív API-t kínál, amely jól integrálható a React komponensekkel, lehetővé téve az alkalmazás struktúrájának és navigációjának logikus és könnyen követhető kialakítását. A könyvtár további előnyei közé tartoznak az átirányítások és a késleltetett navigáció támogatása, ami hozzájárul a felhasználói élmény javításához.

### axios

Webalkalmazások létrehozásakor gyakori feladat a HTTP protokollon keresztül való kommunikáció, amit az egyik legegyszerűbben az *Axios* könyvtárral tudunk implementálni. Használatával könnyedén küldhetünk mindenféle http kérést, mint a get, post vagy delete, valamint kezelhetjük az ezekre adott válaszokat. Az *Axios* számos előnyt kínál a Javascriptben alapból elérhető *fetch()* metódushoz képest, például automatikusan kezeli a JSON adatok átalakítását, és képes kezelni az időtúllépéseket és hibakezelést. A könyvtár támogatja az aszinkron működést, így *Promise*-okkal és *async/await* szintaxissal is használható, ami megkönnyíti az aszinkron műveletek kezelését és a kód olvashatóságát. Az Axios emellett lehetővé teszi az egyedi konfigurációkat, például alapértelmezett fejlécértékek, alap URL-ek és más beállítások megadását, amelyekkel testreszabhatjuk az alkalmazás HTTP kéréseinek viselkedését

### Typescript

A javascript webfejlesztésben való elterjedésével rohamosan derültek ki a nyelv anomáliái és hiányosságai. Ezt hivatott orvosolni a Typescript, ami a Javascript egy szigorúbb kiterjesztése. Segítségével fordítási időben is kezelhetünk hibákat, az erős típusellenőrzésnek köszönhetően pedig a legtöbb hiba már a fejlesztőkörnyezetünkben jelentkezik. Létrehozhatunk benne egyedi típusokat és interfaceket is

### Egyéb használt könyvtárak és bővítmények (?)

## Szolgáltatások

### Verziókezelés

Fejlesztés során fontos nyomon követni a fájlokban végrehajtott változtatásokat. A verziókezelés lehetővé teszi, hogy visszatérjünk korábbi változatokhoz és új verziókat hozzunk létre. Erre a legelterjedtebb eszköz az, ingyenes és nyílt forráskódú git..  
Kisebb és nagyobb projektekben is használható, egyedül inkább verziókövetésre alkalmas, de hasznos tud lenni ha különböző eszközökön fejlesztünk. Több felhasználó számára lehetőséget biztosít arra, hogy ugyanazon a kódon dolgozzanak, és könnyedén kommunikáljanak egymás között.

A git parancsait kiadhatjuk manuálisan az operációs rendszerünk parancssorából, vagy a git saját Command Line Interface-ében (CLI), de léteznek különböző asztali alkalmazások, mint például a Github Desktop, ami gyorsabbá és felhasználóbarátabbá teszi a könyvtárak verziókezelését. Emellett a legtöbb modern fejlesztőkörnyezetben elérhetők különböző bővítmények, melyek segítségével mellőzhető a CLI-k használata

A Github egy olyan szolgáltatás, ami git könyvtárak(repository-k) tárolására szolgál. Könnyen átlátható felületet biztosít a változtatásokhoz és lehetővé teszi különböző műveletek végrehajtását, mint a kódértékelés, vagy úgynevezett issue-k létrehozása, melyek segítségével hatékonyan nyilvántarthatóvá válnak az elvégzendő részfeladatok.

Mivel egyedül fejlesztettem az alkalmazást, így a git használatakor gyakran előforduló merge conflictokat nem tapasztaltam, sokkal inkább verziókövetésre használtam, illetve nagyban elősegítette a több eszközről való munkát.

### Google Maps API

A Google Cloud egy teljes körű felhőalapú szolgáltatásokat nyújtó platform, melynek segítségével könnyen implementálható megoldásokat kapunk olyan problémákra, mint az adattárolás, adatelemzés, vagy gépi tanulás.

Ezek között található a Google Maps API, aminek segítségével egy térképeket jeleníthetünk meg az alkalmazásunkban. Ezt számtalan módon testre szabhatjuk, jelölőket és útvonalakat jeleníthetünk meg a térképen, valamint kezelni tudjuk vele a felhasználói interakciókat, ezzel javítva a felhasználói élményt.

//TODO jobban kifejteni

# Hasonló alkalmazások

## Untappd

Az Untappd egy mobilon és weben egyaránt elérhető alkalmazás, ami kifejezetten a kézműves sörök piacára van kiélezve. Az alkalmazásban megtalálhatóak világszerte elérhető kocsmák és bárok, a profiljukban szerepel minden fontos információ a helyről.

A menüt a kocsmák maguk tudják változtatni, így naprakészen tartva azt hogy éppen mi elérhető náluk. A sörök valamennyi paramétere, mint a típus, származási hely, alkoholszázalék, vagy a kiszerelés mérete és ára megtalálható a menükben.

Emellett az alkalmazásban lehetőség nyílik különböző interakciókra a kocsmákkal és más felhasználókkal: be tudunk jelentkezni egy adott helyről, hogy hol, kivel, és mit fogyasztunk, és a fogyasztott terméket tudjuk értékelni, mind egy 1-5-ig terjedő skálán, mind pedig különböző tagek hozzáadásával.

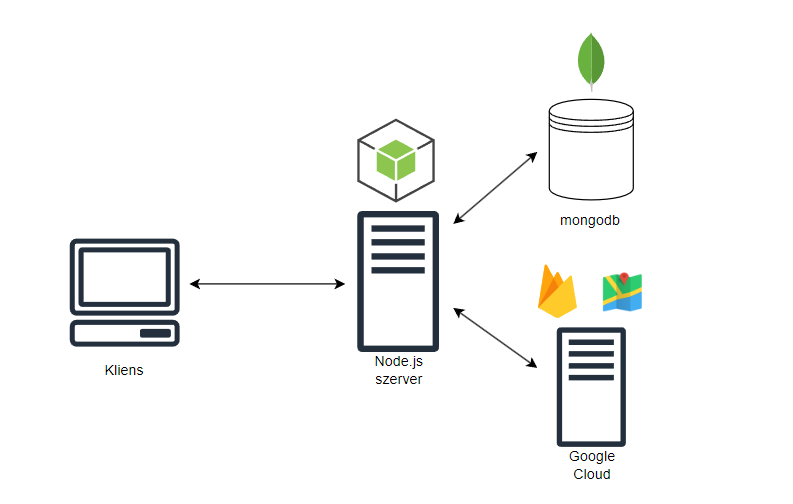
//TODO keresni még 1-2 hasonló appot/oldalt

# Tervezés

## Architektúra

//TODO átírni a nagy részét (jobban kifejtve) + új ábra mert a firebase kukázva lett.

Az alkalmazás architektúrája két fő részből áll: frontend alkalmazás és backend szerver. A kettő közötti kommunikáció HTTP protokollon, REST API segítségével történik. Emellett a szerver kommunikál egy MongoDB szerverrel, ahol az adatokat tárolja, valamint a Google Clouddal, ahonnan a térkép megjelenítéséhez használt API-t, valamint a későbbiekben használandó Firebase által nyújtott felhasználókezelő szolgáltatást veszi igénybe.



1. ábra Az alkalmazás architektúrája

## Felhasználói szerepkörök

//TODO kicsit jobban kifejteni, illetve bővíteni, ha marad idő a harmadik rolera

Az alkalmazás jelenleg két felhasználótípust különböztet meg: admint és felhasználót.

* Felhasználó

A felhasználó számára megjelenik a 3 különböző lista, ezek elemeit tudja a kedvencei közé helyezni. A kocsmák esetében lehetősége van pontozni a helyet, a sok helyen használt 1-5 csillagos skálán.

* Admin

Az admin felhasználók számára ezen felül lehetőségük van hozzáadni új elemeket bármelyik listához, valamint törölni elemeket belőlük.

A későbbiekben elképzelhető, hogy bővülni fog a felhasználói körök listája, egy köztes, „moderátor”-jellegű felhasználóval, aki csak bizonyos elemeket tud szerkeszteni vagy törölni.

## Adatmodellek

//TODO jobban kifejteni, kibővíteni az új dolgokkal

A feladat megoldása során négy különböző adatmodellt hoztam létre, hármat a 3 különböző eltárolt objektumról és egyet a felhasználók számára.

### Ingredient

A hozzávalók információit tárolja, amelyek:

* a hozzávaló neve
* a hozzávaló alkoholszázaléka
* a hozzávaló típusa

Valószínűleg nem ez a leg-letisztultabb tárolási módja a hozzávalóknak, mert az alkoholmentes üdítők vagy egyéb díszítőelemeknek felesleges alkoholszázalékot eltárolni, valamint az alkoholokat is el lehet látni több tulajdonsággal, nem csak az erősségükkel.

### Drink

Az italok információit tárolja:

* az ital neve
* az ital típusa
* az italhoz szükséges pohár
* az italhoz tartozó kép
* az italt alkotó hozzávalók tömbje

### Pub

A kocsmák információit tárolja:

* a kocsma neve
* a kocsma címe
* a kocsma szélességi és hosszúsági koordinátája
* a kocsmára érkezett értékelések tömbje
* a kocsma itallapja
* a kocsma nyitvatartási ideje

A kocsmák tárolásához létre kellett hoznom specifikus típusokat, hogy a nyitvatartási idő, és a menüben található elemek eltárolhatóak legyenek

### User

A felhasználókról tárolt adatokat tárolja. Ez az adatmodell még elég kezdetleges, mert a felhasználókezelés megvalósítása nem történt meg teljes egészében, így csak az alábbiakat tárolja:

* a felhasználó által kedvelt hozzávalók
* a felhasználó által kedvelt italok
* a felhasználó által kedvelt kocsmák

## REST API végpontok

//TODO kibővíteni a listát

### REST

A REST (Representational State Transfer) architektúra szabályok és elvárások összessége. Ezeknek az elvárásoknak való megfelelés segíti az HTTP API-k készítését, ezáltal a rendszereket kezelhetőbbé, gyorsabbá és könnyebben skálázhatóvá teszi. Az architektúra alapjait képezi a kliens-szerver modell, ahol a kommunikáció az HTTP protokollon keresztül zajlik. Emellett fontos, hogy a komponensek közötti kommunikáció egy egységes interfészen keresztül történjen, és a szerver válaszai világosak és érthetőek legyenek a kliens számára.

## Végpontok

Az alkalmazás az alábbi végpontokon keresztül kommunikál a szerverrel. Ezeken keresztül történik az erőforrások adatbázishoz adása, valamint olvasása és módosítása.

|  |  |  |
| --- | --- | --- |
| Metódus | URL | Leírás |
| GET | /routes/ingredients/ | A hozzávalók lekérdezése |
| POST | /routes/ingredients/add | Egy hozzávaló hozzáadása |
| POST | /routes/ingredients/update/:id | Egy hozzávaló módosítása |
| DELETE | /routes/ingredients/:id | Egy hozzávaló törlése |
| GET | /routes/drinks/ | Az italok lekérdezése |
| POST | /routes/drinks/add | Egy ital hozzáadása |
| POST | /routes/drinks/update/:id | Egy ital módosítása |
| DELETE | /routes/ drinks /:id | Egy ital törlése |
| GET | /routes/pubs/ | A kocsmák lekérdezése |
| POST | /routes/pubs/add | Egy kocsma hozzáadása |
| POST | /routes/pubs/update/:id | Egy kocsma adatainak változtatása |
| POST | /routes/pubs/update-rating/:id | Egy kocsma értékeléseinek változtatása |
| DELETE | /routes/pubs/:id | Egy kocsma törlése |
| GET | /routes/users/ | A felhasználók lekérdezése |
| GET | /routes/users/:id | Egy felhasználó lekérdezése |
| POST | /routes/users/add | Egy felhasználó hozzáadása |
| POST | /routes/users/update/:id | Egy felhasználó adatainak frissítése |

## Fontosabb funkciók

//TODO lesz még ide mit írni

### Kedvencekhez adás

A felhasználóknak lehetőségük van mind a három listában szereplő elemek közül a kedvenceikhez adni elemeket. Ezt a függvényt egy gomb valósítja meg ami az adott típusú elem paneljében található. A gombnyomás hatására az alkalmazás megvizsgálja, hogy a felhasználónak szerepel-e az elem a listájában. Ha szerepel, akkor kiveszi onnan, ha nem szerepel akkor pedig hozzáadja, így oldja meg a függvény, hogy egyszerre történjen a kedvencekhez adás és a kedvencekből való kivétel kezelése. Ezt követően az axios könyvtár segítségével egy POST requesttel a megfelelő URL-re elküldi a megváltoztatott, kedvenceket tároló tömböt.

### Listák szűrése

A listák szűrése során a függvény paraméterül kap egy objektumot, aminek az adattagjai megegyeznek a listában tárolt elemével, azzal a különbséggel, hogy azok az elemek amelyekre nem szűrűnk üresen maradnak. A szűrt adattagok értékeivel a Javascriptben tömbökön alkalmazható .*filter()* metódus segítségével létrehozok egy új listát, melyben csak a szűrési feltételeknek megfelelő elemek találhatók, majd az adott panel megfelelő állapotát módosítom a létrehozott tömbre.

# Implementáció

## -Backend:

//TODO kiegészíteni főleg a user-kezelés

Az alkalmazás backendjét egy Node.js szerver szolgálja. Ide érkeznek be a REST API hívások a webalkalmazásból, amik feldolgozásra és végrehajtásra kerülnek. A szerverhez kapcsolódik egy MongoDB adatbázis, amelyben a definiált sémák szerint tárolom el a beérkező adatokat.

### MongoDB adatmodellek

Az előző fejezetben kifejtett különböző típusú elemek tárolásához először létrehoztam a hozzájuk tartozó sémákat. Ezek specifikus JSON objektumok, amik meghatározzák az eltárolandó entitás adattagjainak nevét és típusát, illetve egy tömbben meg lehet adni, hogy ezek közül melyikeket kötelező specifikálni, valamint default értéket is megadhatunk.

Minden tárolandó adatnak létrehoztam egy külön *\*\_model.js* fájlt, amiből a létrehozott Schema exportálásra kerül. Itt specifikálom az adattagok nevét és típusát. A tartalmazásokat, mint például mikor az italok összetevőit tárolom el, egy string tömbbel oldottam meg, amiben az eltárolt dolgok nevei találhatóak. Itt általánosan egy jobb, de főleg időtállóbb megoldás lehetett volna az elemek azonosítóinak az eltárolása, de úgy gondoltam, hogy az adatok komplexitása nem feltétlen követeli ezt meg, így maradtam a nevek tárolásánál.

const mongoose = require("mongoose")

const Schema = mongoose.Schema

const drinkSchema = new Schema({

  name: { type: String },

  type: { type: String },

  ingredients: {type: Array},

  glass: {type: String},

  img: {type: String}

})

const Drinks = mongoose.model("Drinks", drinkSchema)

module.exports = Drinks

A felhasználók esetében viszont az entitások azonosítója került eltárolásra, a kedvencekhez adott dolgok, valamint a barát és kérelem listák esetében is.

### Routeok

Ahhoz hogy a HTTP kérések fogadását ki tudjam szervezni külön fájlokba, a strukturáltság és átláthatóság miatt, bevezettem különböző route-okat az alkalmazásba. Ezek segítségével, ha egy adott URL-en érkezik egy kérés az app felé, az átirányítja a megfelelő osztály számára.

const ingredientsRouter = require('./routes/ingredients')

app.use('/ingredients', ingredientsRouter)

## -Frontend:

//TODO frissíteni a meglévőket, kiegészíteni az új dolgokkal  
//TODO +1 alfejezet a reduxról

//TODO új képernyőképek

### Komponensek

A komponensek felépítésénél próbáltam minél jobban az újrafelhasználhatóságra törekedni, és hogy az egy csoportba tartozó komponensek szeparálva legyenek tárolva.  
Külön mappákba csoportosítottam a különböző panelek – hozzávalók, italok, kocsmák, térkép, felhasználók – komponenseit, valamint a komponensek gyökérkönyvtárában hagytam a közösen használt és általános osztályokat.

Egy panel alapvetően 4 komponensből áll:

* Az individuális elem megjelenítésért felelős komponens
* A komponenseket kilistázó és interakciókat kezelő komponens
* Egy formot és az ehhez használt függvényeket tartalmazó komponens, amit a hozzáadáskor és szerkesztéskor használok
* Egy másik formot, ami a megjelenített lista filterezését végzi

Az elemek komponensében a props-ban paraméterként átadott elem szükséges információit jelenítem meg, valamint ide vettem fel az elem szerkesztéséért felelős formot, ami egy felugró dialógus-ablakban jelenik meg. Mindenhol megjelenik az eltárolt elem neve, ezen kívül összetevő esetében a típusa, italok esetében a hozzá társuló kép és összetevők, kocsmák esetében pedig az értékelés található meg az elemet reprezentáló kártyán.

Hozzávalók és italok esetében az információk mellett a kedvencekhez adáshoz található egy gomb, a kocsmáknál ez mellett helyet kaptak gombok az értékelésre és a menü megtekintésére, valamint az egyik gomb segítségével az alkalmazás átirányít a térkép felületre, a kiválasztott kocsma megjelenítésével együtt.

//TODO két kép 1-1 item cardról

Az hozzáadást és szerkesztést ellátó űrlapokat megvalósító komponensek viszonylag egyszerűen épülnek fel. Mind egy HTML form elemet ad vissza, az adott típusú elem adataihoz tartozó és illő beviteli mezőkkel. Ezek értelemszerűen a névnél és hasonló szöveges adatoknál egy egyszerű input mező, alkoholfok vagy koordináta megadásánál számok, egyéb specifikus adatoknál pedig legördülő menüben lehet kiválasztani a kívánt értéket. Ilyen például az ital hozzáadásánál a típus vagy a hozzá tartozó pohár fajtája.

A szűrők beállításáért felelős űrlap hasonló az előzőhöz, csak itt felugró dialógus-ablak helyett a lista felett jelennek meg a beviteli mezők. Az elemek között rá tudunk szűrni a névre, típusra, italoknál arra, hogy milyen összetevőt tartalmazzon, valamint szórakozóhelyek esetében értékelés alapján is tudjuk szűkíteni a keresést.  
Itt nyilván kevesebb beviteli mezőt kellett megadni, illetve a form elrendezése a legtöbb weboldalon megszokotthoz hasonlóan vertikálisan lett megoldva.

//TODO kép egy szűrőről (?)

A térkép oldal 2 fő komponensből áll, egy ami a térkép megjelenítésért felel, és egy ami az adott pinekhez tartozó információs panelt jeleníti meg.

A kocsmákért felelős komponenseknél jobban szétbontottam a működés megvalósítását, mert az itt tárolt adatok komplexitása miatt a hozzáadáshoz és szerkesztéshez több komponenst kellett létrehoznom a formokhoz. A menü szerkesztéséhez egy külön felugró ablakot hoztam létre, ahol látható a menüben már megtalálható elemek listája, valamint egy egyszerű form segítségével, ami egy név – ár párost vár, hozzáadhatunk elemeket a menühöz. A nyitva tartás szerkesztésénél táblázat-szerűen jelennek meg a hét napjai és mellettük a hozzájuk tartozó nyitás és zárás időpontja. Az alkalmazás jelenleg úgy kezeli azt az esetet, hogy az adott napon zárva tart a hely, hogy 0:00 – 0:00-ig van megadva a nyitvatartása.

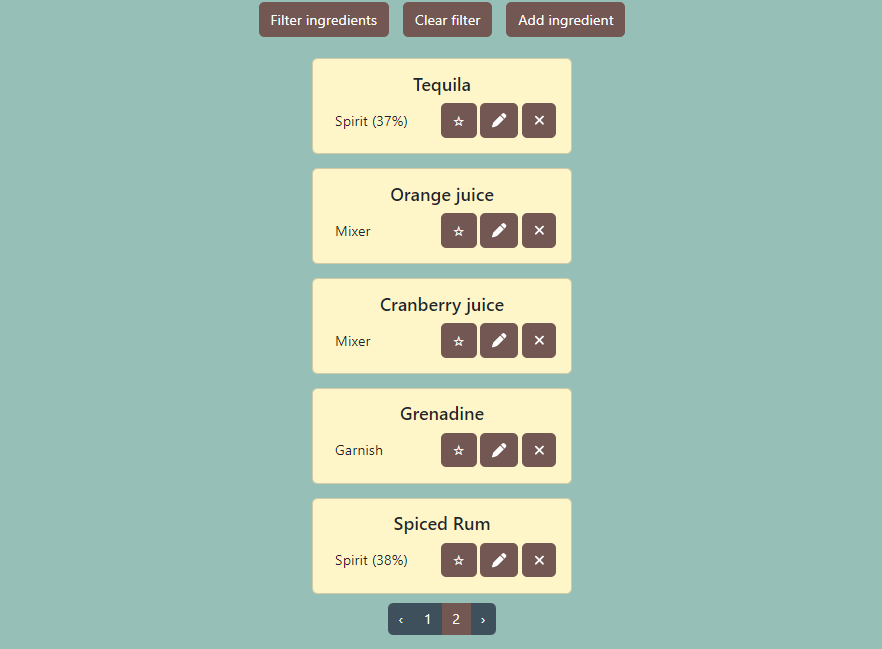
A paneleket alkotó komponensek felépítését a következőképpen valósítottam meg.  
Minden komponenshez definiáltam egy típust, ami a hozzá tartozó props elemeit határozza meg. Ezt követően a komponens arrow function-nal való létrehozása után először definiálom a felhasznált konstans változókat, szinte mindenhol a useState hook segítségével, hogy később változtatni lehessen az adatokat, és dinamikusan megjeleníteni ezeket. Aztán a komponensben felhasznált segédfüggvényeket hozom létre, majd következik a komponens által visszaadott HTML elem megalkotása.

Ez az individuális elemek komponenseiben egy React Bootstrapből importált Card elem, ami tartalmaz egy Title elemet az objektum nevével, valamint egy Stack elemet a komponenssel való interakcióhoz szükséges gomboknak. Alap esetben három gomb található a komponensekben, egy a kedvencekhez adásért felel, egy a szerkesztést, az utolsó pedig a törlést valósítja meg. Ezek mellett megtalálható még valamilyen ismertető szöveg az adott elemről, és egyes komponensek esetén egy képet is megjelenítek.



2. ábra Egy megjelenített elem

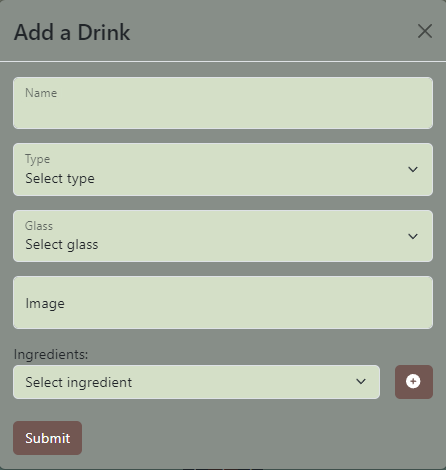
Magát a panelt kezelő komponens felépítése hasonlít az indiviudális elemekéhez, a visszaadott elem felépítésében azonban nagyban eltér. A panel legtetején találhatóak a filtert kezelő elemek. Egy gomb segítségével tudjuk megjeleníteni vagy elrejteni a kereső mezőket, és a szűrést végző gombot. Ezután az objektumok hozzáadását rejtő form Modal-ja található, de ez is értelemszerűen csak a megfelelő gomb megnyomására válik láthatóvá. Ez alatt listázódnak ki a panelhez tartozó elemek, a javascript tömbökön használható *.map()* metódussal. Végül a külön megírt Pagination osztály egy eleme található itt, ami paraméterül kapja az oldal méretet és a kilistázott lista hosszát, és ez alapján jeleníti meg az oldalak közötti váltást megvalósító gombokat.



3. ábra A hozzávalók panelje

Az elemek hozzáadásáért és szerkesztéséért felelős formok mezői az elem típusától nagyban függenek, de ahol lehetett a react-bootstrap segítségével FloatingLabel-eket helyeztem el a beviteli mezőkön, és ahol logikusnak érződött, például az italok hozzáadásánál a hozzávalók kiválasztása, ott legördülő menü segítségével hoztam létre az adatbevitelhez szükséges mezőt.

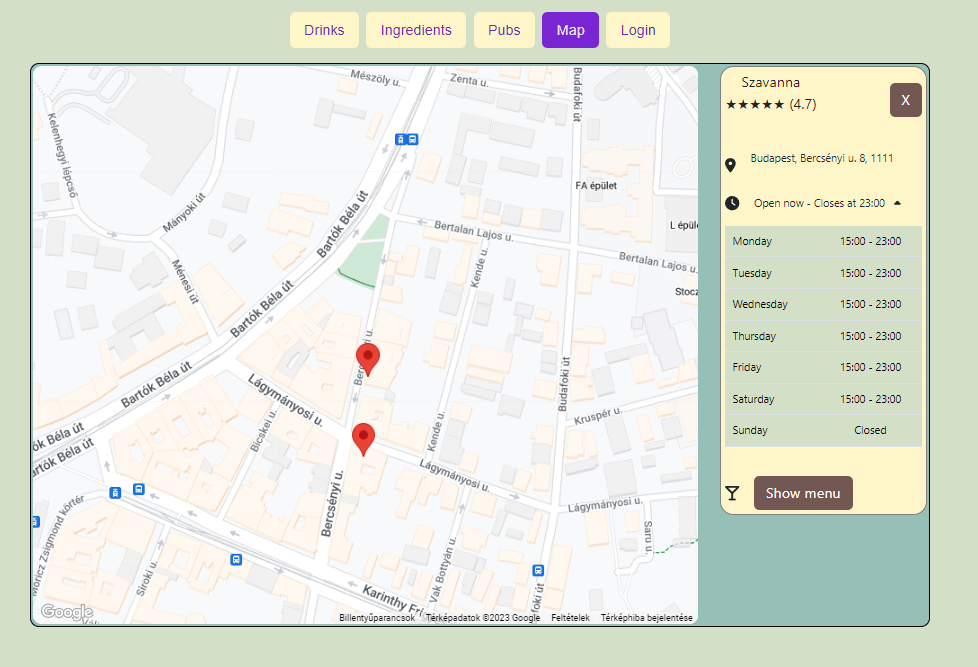
A kocsmák hozzáadása esetén, a menü és nyitvatartási idő elemeinek összetettsége miatt, ezek módosítását egy külön felugró ablakban található formban valósítottam meg, de ezekben is hasonló logika mentén jártam el.



4. ábra Az italok hozzáadását kezelő form

A térkép panel két fő részből áll, magát a térképet tartalmazó komponensből, és a kijelölt vendéglátói egység információit megjelenítő oldalsó panelből. Ez a térképen található gombostűkre kattintva jeleníthető meg illetve frissíthető a tartalma.  
A térkép egy react-google-maps könyvtárból importált komponens, ami paraméterként megkapja a komponensben korábban specifikált adatokat a térkép megjelenítéséhez. Ilyen például a térkép stílusa, ahol ki lehet kapcsolni a számunkra nem szükséges gombostűket a térképen, vagy a térkép színpalettáját konfigurálhatjuk.  
A térképen belül kilistázom az adatbázisban szereplő kocsmák helyzetét az eltárolt szélességi és hosszúsági koordináták segítségével.

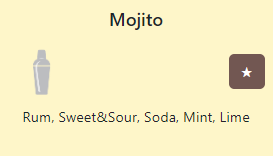
Az oldalsó panel paraméterül megkapja a kiválasztott kocsmát, és megjeleníti az arról eltárolt információkat. A menü egy felugró ablakként jelenik meg, a nyitvatartási idő pedig kattintásra lenyitható, hogy ne csak az adott napi nyitva tartást mutassa, hanem a többi naphoz tartozót is.



5. ábra A térkép panel

### Jogosultságok

Az alkalmazásban a jogosultság kezelés még elég kezdetleges, egy boolean változó felel azért, hogy éppen megjelenít-e adott komponenseket az app, vagy sem. Így az alap felhasználók nem férnek hozzá olyan metódusokhoz mint az elemek szerkesztése vagy törlése.

6. ábra Különbség a felhasználók által látott elemek között

# Tesztelés

# Összegzés

# Köszönetnyilvánítás

Irodalomjegyzék

1. Levendovszky, J., Jereb, L., Elek, Zs., Vesztergombi, Gy.: Adaptive statistical algorithms in network reliability analysis, Performance Evaluation - Elsevier, Vol. 48, 2002, pp. 225-236
2. National Istruments: LabVIEW grafikus fejlesztői környezet leírása, <http://www.ni.com/> (2010. nov.)
3. Fowler, M.: UML Distilled, 3rd edition, ISBN 0-321-19368-7, Addison-Wesley, 2004
4. Wikipedia: Evaluation strategy, <http://en.wikipedia.org/wiki/Evaluation_strategy> (revision 18:11, 31 July 2012)

Függelék