

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.



M Ű E G Y E T E M 1 7 8 2

Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
XXX Tanszék

Sereg András

TESTRE SZABHATÓ KOKTÉLADATBÁZIS KÉSZÍTÉSE

React.js és Node.js segítségével

KONZULENS

Kövesdán Gábor

BUDAPEST, 2023

Tartalomjegyzék

Összefoglaló	6
Abstract.....	7
1 Bevezetés	8
2 Felhasznált technológiák	9
2.1 Backendhez használt technológiák	9
2.1.1 Node.js	9
2.1.2 MongoDB	9
2.2 Frontendhez használt technológiák.....	9
2.2.1 React	9
2.2.2 React-bootstrap	10
2.2.3 axios.....	10
2.2.4 Typescript	10
2.3 Szolgáltatások	11
2.3.1 Verziókezelés.....	11
2.3.2 Google Maps API	11
3 Hasonló alkalmazások	12
3.1 Untappd.....	12
4 Tervezés	13
4.1 Architektúra	13
4.2 Felhasználói szerepkörök.....	13
4.3 Adatmodellek.....	14
4.3.1 Ingredient.....	14
4.3.2 Drink	14
4.3.3 Pub	15
4.3.4 User.....	15
4.4 REST API végpontok	15
4.4.1 REST.....	15
4.5 Végpontok.....	16
4.6 Fontosabb funkciók.....	17
4.6.1 Kedvencekhez adás.....	17
4.6.2 Listák szűrése.....	17

5 Implementáció	18
5.1 -Backend:	18
5.1.1 MongoDB adatmodellek	18
5.1.2 Routeok	18
5.2 -Frontend:	19
5.2.1 Komponensek	19
5.2.2 Jogosultságok	24
6 Telepítési és használati útmutató.....	25
7 Összegzés.....	26
8 Köszönetnyilvánítás	27
Irodalomjegyzék.....	28
Függelék.....	29

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, 2023. 12. 07.

.....
Sereg András

Összefoglaló

Abstract

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

1 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 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.

2 Felhasznált technológiák

2.1 Backendhez használt technológiák

2.1.1 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.

2.1.2 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.

2.2 Frontendhez használt technológiák

2.2.1 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 újrahasználatóságot, 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.

2.2.2 React-bootstrap

A React Bootstrap a Bootstrap grafikus keretrendszer React-komponensekkel való kiegészítése, melynek segítségével hatékony módon vihetünk különböző UI elemeket az alkalmazásunkba, amelyek követik a Bootstrap stílusrendszerét.

Egyes komponensek megjelenítését függővé tehetjük a kijelző méretétől, így néhány sornyi kóddal több különböző eszközre optimalizált kódot írhatunk.

A könyvtár használatával számos előre összerakott komplex komponenseket is tudunk használni, mint például különböző űrlap elemek, vagy felugró ablakok

2.2.3 axios

Webalkalmazások létrehozásakor gyakori feladat a HTTP protokollon keresztül való kommunikáció. A böngészők képesek fogadni HTTP kéréseket a beépített fetch() metódusukkal, ezek küldését pedig az Axios javascript könyvtár segítségével lehet megoldani.

2.2.4 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

2.3 Szolgáltatások

2.3.1 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.

2.3.2 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 térképet 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.

3 Hasonló alkalmazások

3.1 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 bárók, a profiljukban szerepel minden fontos információ a helyről.

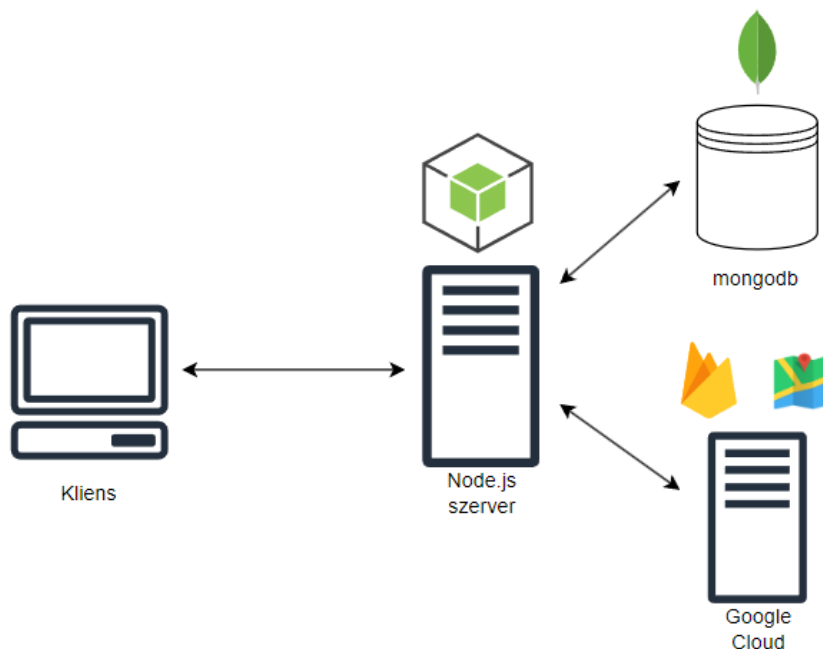
A menüt a kocsmák maguk tudják változtatni, így naprakészen tartva azt hogy éppen mi elérhető náluk. A sörök valamennyi paramétere, mint a típus, származási hely, alkoholszázalék, vagy a kiszereles mérete és ára megtalálható a menükben.

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.

4 Tervezés

4.1 Architektúra

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

4.2 Felhasználói szerepek

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.

4.3 Adatmodellek

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

4.3.1 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.

4.3.2 Drink

Az italok információit tárolja:

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

4.3.3 Pub

A kocsmák információit tárolja:

- a kocsmá neve
- a kocsmá címe
- a kocsmá szélességi és hosszúsági koordinátája
- a kocsmára érkezett értékelések tömbje
- a kocsmá itallapja
- a kocsmá nyitvatartási ideje

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

4.3.4 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

4.4 REST API végpontok

4.4.1 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.

4.5 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 kocsmát hozzáadása
POST	/routes/pubs/update/:id	Egy kocsmának adatainak változtatása
POST	/routes/pubs/update-rating/:id	Egy kocsmának értékeléseinek változtatása
DELETE	/routes/pubs/:id	Egy kocsmát 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

4.6 Fontosabb funkciók

4.6.1 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.

4.6.2 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 szűrünk 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.

5 Implementáció

5.1 -Backend:

5.1.1 MongoDB adatmodellek

Az előző fejezetben kifejtett adatmodellek tárolásához létre kellett hoznom úgynevezett Schema-kat, ami az adatok strukturált és egységes kezelését segítik elő a MongoDB adatbázisokban.

A megfelelő importok után létrehoztam külön mind a négy tárolandó objektumnak egy-egy *_model.js* fájlt, amiben létrehoztam a hozzá tartozó schemát. Itt egy objektumban kell definiálnunk a tárolandó adatok strukturáját, ami egy *név : típus* párosok sorozata. Ezt követően létrehozzuk a modellt a kívánt kollekciónévvel. Ahhoz hogy referálni tudjunk az adatmodellre, ki kell exportálnunk a fájlból.

```
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
```

5.1.2 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)
```

5.2 -Frontend:

5.2.1 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 gyökörkönyvtárában hagytam a közösen használt és általános osztályokat.

Egy panel alapvetően 4 komponensből áll:

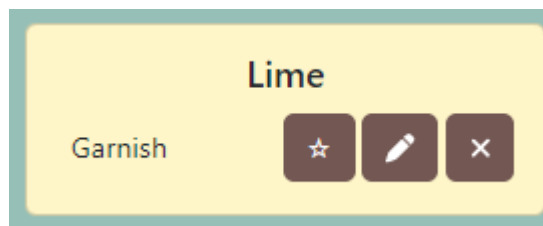
- Az individuális elem megjelenítésért felelős komponens
- A komponenseket kilistázó és interakciókat kezelő komponens
- Egy formot és az ehhez használt függvényeket tartalmazó komponens, amit a hozzáadáskor és szerkesztéskor használok
- Egy másik formot, ami a megjelenített lista filterezését végzi

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.

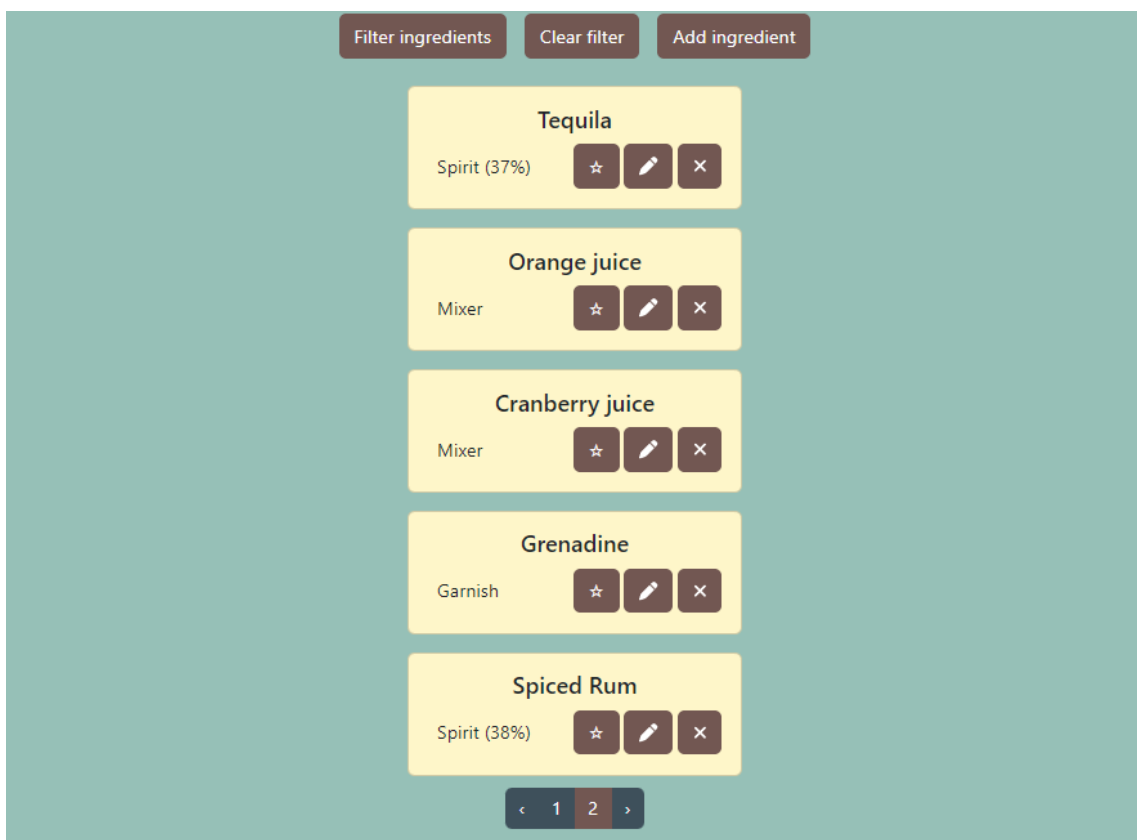
A paneleket alkotó komponensek felépítését a következőképpen valósítottam meg. Minden komponenshez definiáltam egy típust, ami a hozzá tartozó props elemeket határozza meg. Ezt követően a komponens arrow function-nal való létrehozása után először definiálom a felhasznált konstans változókat, szinte mindenhol 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

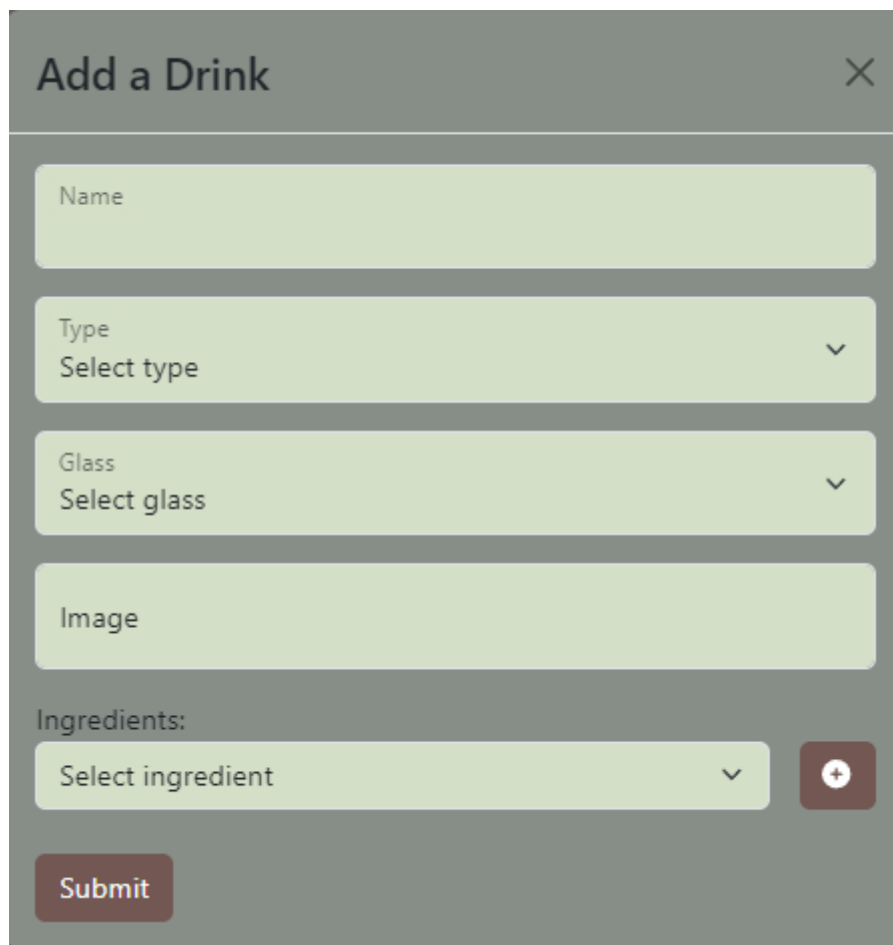
Magát a panelt kezelő komponens felépítése hasonlít az individuális elemekéhez, a visszaadott elem felépítésében azonban nagyban eltér. A panel legtetején találhatóak a filtert kezelő elemek. Egy gomb segítségével tudjuk megjeleníteni vagy elrejtetni a kereső mezőket, és a szűrést végző gombot. Ezután 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 koktélok 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.



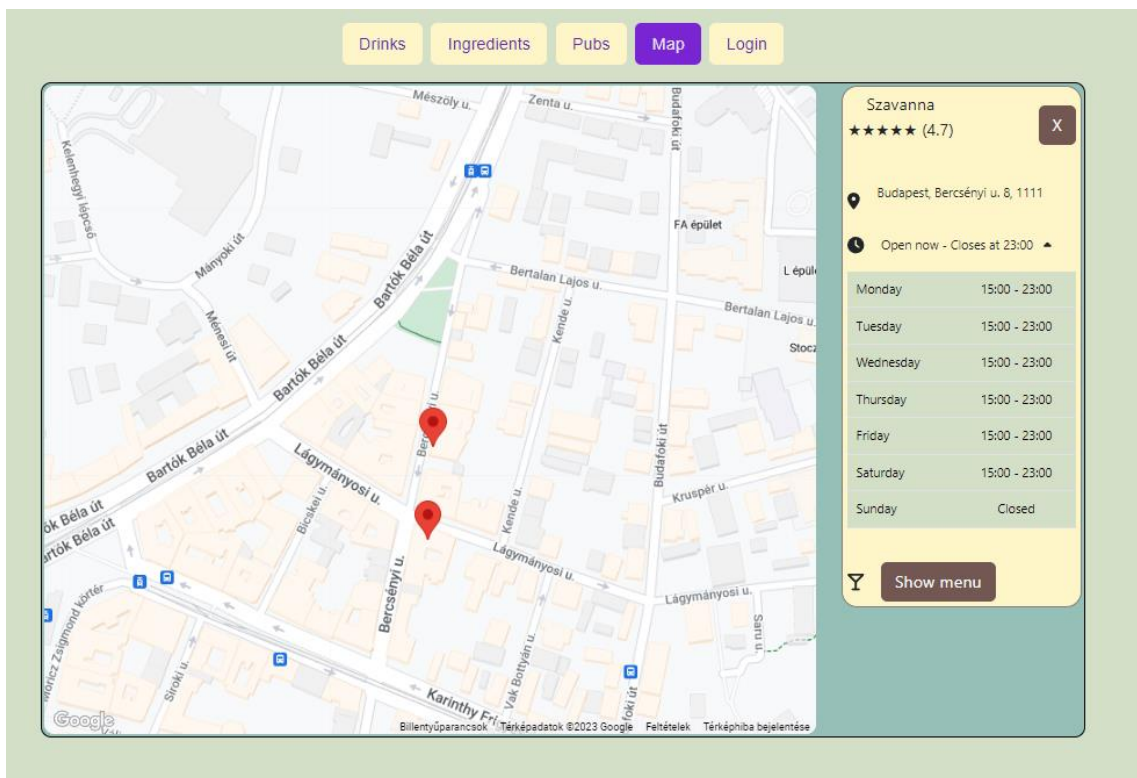
The image shows a web form titled "Add a Drink" with a close button (X) in the top right corner. The form contains the following fields and controls:

- Name:** A text input field with a light green background and a floating label.
- Type:** A dropdown menu with a light green background, a floating label, and a downward arrow icon.
- Glass:** A dropdown menu with a light green background, a floating label, and a downward arrow icon.
- Image:** A text input field with a light green background and a floating label.
- Ingredients:** A section with a label "Ingredients:" followed by a dropdown menu with a light green background, a floating label, and a downward arrow icon. To the right of the dropdown is a dark red button with a white plus sign (+).
- Submit:** A dark red button with white text located at the bottom left of the form.

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.

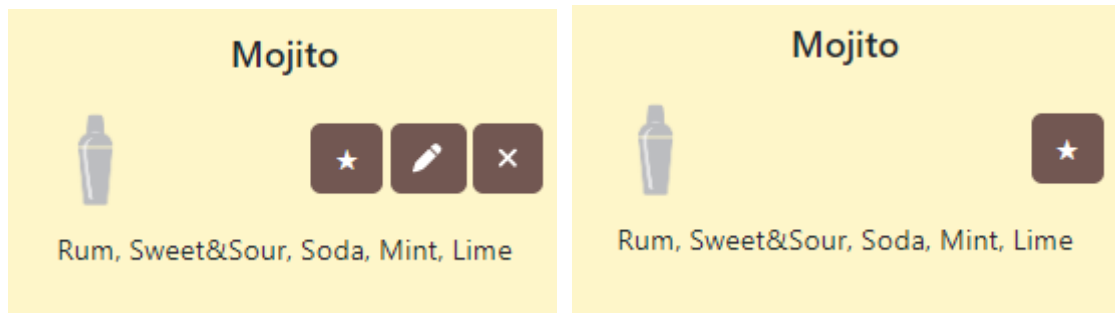
Az oldalsó panel paraméterül megkapja a kiválasztott kocsmát, és megjeleníti az arról eltárolt információkat. A menü egy felugró ablakként jelenik meg, a nyitvatartási idő pedig kattintásra lenyitható, hogy ne csak az adott napi nyitva tartást mutassa, hanem a többi naphoz tartozót is.



5. ábra A térkép panel

5.2.2 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

6 Telepítési és használati útmutató

7 Összegzés

8 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 Instruments: *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