

**Szegedi Tudományegyetem
Informatikai Intézet**

Receptmegosztó weboldal fejlesztése

Szakdolgozat

Készítette:
Virágh Vivien
programtervező
informatikus BSc
szakos hallgató

Témavezető:
Dr. Mingesz Róbert
adjunktus

Szeged
2023

Feladatkiírás

Manapság a papír alapú receptkönyvek helyett sokan inkább az online receptmegosztó oldalakhoz fordulnak, mivel használatukkal hatalmas választék érhető el, könnyen megtalálhatjuk a számunkra tökéletes receptet, és sok lehetőségünk van azok rendszerezésére. A hallgató feladata egy weboldal elkészítése JavaScript alapú technológiával, ahova a felhasználók különböző recepteket tölthetnek fel, azokat egyszerűen és hatékonyan böngészhetik és kezelhetik. Az oldal emellett a mindennapi sütés-főzés megtervezésében is ki tudja segíteni a felhasználókat.

Szükséges funkciók:

- Felhasználókezelés
- Receptek feltöltése, böngészése, és rendszerezése
- Bevásárlólista kezelése
- Heti menü ajánlása a felhasználónak

Tartalmi összefoglaló

- ***A téma megnevezése:***

Egy receptmegosztó weboldal kifejlesztése.

- ***A megadott feladat megfogalmazása:***

Olyan weboldal elkészítése, melyre a felhasználók feltölthetik saját receptjeiket, illetve a mások által feltöltötteket könnyen böngészhetik. A recepteket a felhasználó kedve szerint rendszerezni tudja, továbbá bevásárlólistát tud vezetni a szükséges hozzávalókról. Az oldal továbbá tud heti menüt ajánlani a felhasználónak, ezzel elősegítve az előre tervezést. Mindennek megvalósítása JavaScript alapú technológiák felhasználásával.

- ***A megoldási mód:***

JavaScript alapú keretrendszerek megismerése a megfelelőek kiválasztásához, szükséges funkciók konkrét viselkedésének kigondolása, a rendszer felépítésének és megjelenésének megtervezése, megvalósítása, majd tesztelése.

- ***Alkalmazott eszközök, módszerek:***

JavaScript, Node.js, Express.js, MySQL, Prisma, Docker, Vue.js, Vite, HTML, SCSS (Sassy Cascading Style Sheets), Bootstrap, Pinia, Controller-Service-Repository tervezési minta

- ***Elért eredmények:***

Letisztult, felhasználóbarát, reszponzív weboldal elkészítése modern, fejlesztést megkönnyítő technológiák felhasználásával.

- ***Kulcsszavak:***

javascript, express, vuejs, mysql, prisma, reszponzív, receptek

Tartalomjegyzék

Feladatkiírás	2
Tartalmi összefoglaló	3
Tartalomjegyzék.....	4
Bevezetés.....	6
1. Felhasznált módszerek, eszközök	7
1.1 JavaScript	7
1.2 Adatbázis	8
1.2.1 MySQL.....	8
1.2.2 Prisma.....	8
1.2.3 Docker	10
1.3 Backend	11
1.3.1 Node.js.....	11
1.3.2 Express.js.....	11
1.3.3 Controller-Service-Repository tervezési minta	14
1.4 Frontend	15
1.4.1 HTML.....	15
1.4.2 SCSS.....	16
1.4.3 Vue.js és az MVVM tervezési minta	16
1.4.4 Vite	19
1.4.5 Axios	20
1.4.6 Bootstrap	21
1.4.7 Pinia.....	21
2. Receptmegosztó weboldal.....	22
2.1 A rendszer felépítése	22
2.1.1 Szerver oldal.....	22
2.1.2 Kliens oldal	23

2.1.3 Az adatbázis struktúrája	24
2.2 Jogosultságok	25
2.3 Regisztráció	27
2.4 Bejelentkezés, kijelentkezés	29
2.5 Recept feltöltése	29
2.6 Recept megtekintése.....	32
2.6.1 Keresés és filterek	32
2.6.2 Recept oldal vendégként	33
2.6.3 Recept oldal bejelentkezett felhasználóként.....	34
2.7 Kedvencek oldal.....	35
2.7.1 Összes kedvenc tab.....	35
2.7.2 Csoportok tab	36
2.8 Profil oldal.....	37
2.9 Heti menü oldal	38
2.10 Bevásárló lista	40
2.10.1 Kategória nézet.....	41
2.10.2 Lista nézet	41
2.11 Admin oldal.....	42
2.11.1 Felhasználók, receptek, és kommentek kezelése.....	42
2.11.2 Az alkalmazás belső adatainak kezelése	43
2.11.3 Statisztikák	44
2.12 Reszponzivitás.....	45
3. Összefoglaló	46
Irodalomjegyzék.....	47
Nyilatkozat	48
Köszönetnyilvánítás	49
Mellékletek.....	50

Bevezetés

Napjainkban a sütni és főzni vágyó emberek körében a számos receptkönyv vásárlása helyett elterjedtebbé vált a különböző receptmegosztó oldalak használata. Ilyen oldalak például magyar körökben a „Nosalty” és a „Mindmegette”, angol nyelvű pedig többek között az „Allrecipes”, jelen alkalmazás pedig a „Bittersweet” nevet kapta.

A fenti oldalakról az alkalmazásom fejlesztése során mintát vettem, azonban célom volt egy ezeknél átláthatóbb, rendezettebb weboldal megalkotása, melynek használata könnyű és intuitív. Az oldal a receptek böngészésén és feltöltésén kívül további kiegészítő funkciókkal is rendelkezik, mint például kedvenc receptjeink elmentése, azoknak csoportosítása, és bevásárlólista vezetése, melyre a receptek hozzávalóit, vagy akár saját egyéb termékeinket is felírhatjuk. Emiatt természetesen az oldalnak reszponzívnak kellett lenniük, különböző méretű monitorokon túl mobiltelefonon és táblagépen is kényelmesen használhatónak, így például bevásárlás közben is nézhetjük telefonunkról. Mindezen túl az oldal heti menüt is tud ajánlani a felhasználónak, figyelembe véve az illető esetleges allergiáit, diétáját, és hogy mennyi időt és pénzt hajlandó a főzésre ráfordítani. Ez bár bizonyos fentebb említett weboldalakon is szerepel, de a menüt az oldal kollégáinak, a webmagazin íróinak kell összerakni, itt viszont a menük automatikusan generáltak és teljesen személyre szabottak, külső beavatkozást nem igényelnek.

A teljes rendszer JavaScript keretrendszerek használatával lett megalkotva. Szerver oldalon az Express.js van alkalmazva, egy Node.js alapú JS backend keretrendszer, mely webalkalmazások és alkalmazásprogramozási interfészek (API) létrehozására készült. Kliens oldalon pedig a Vue.js frontend keretrendszert használja az alkalmazás, mellyel úgynevezett egyoldalas alkalmazásokat (single-page application, SPA) lehet készíteni, és nagyban megkönnyíti a frontend oldali fejlesztést. A reszponzivitást a Bootstrap kliens oldali keretrendszerrel és további *media query*-kkel valósítottam meg. Az alkalmazás MySQL adatbázist használ, annak kezelése azonban nem közvetlen SQL parancsokkal történik, hanem a Prisma ORM (Object-Relational Mapping, objektum-relációs leképezés) segítségével. A Prisma lehetőséget nyújt az adatbázis tábláinak egyszerű létrehozására, és a CRUD (create, read, update, delete) műveletek végrehajtására JavaScript és TypeScript nyelveken.

A fent felsorolt modern technológiák nagyban elősegítik a könnyebb fejlesztési folyamatot, és segítségükkel a weboldal továbbfejlesztése is könnyen megvalósítható lehet a jövőben.

1. Felhasznált módszerek, eszközök

1.1 JavaScript

A JavaScript nyelvet eredetileg Brendan Eich fejlesztette ki, neve azonban kezdetleg Mocha, majd LiveScript volt, a JavaScript elnevezést csak később kapta meg. 1997-ben a nyelvet az ECMA International szabványosította ECMAScript néven. Az ECMA-262 szabvány jelenleg is érvényes, és a JavaScript 1.5-nek felel meg.

A JavaScript egy weboldalakon elterjedten használt objektumorientált szkriptnyelv. A szkriptnyelvekről azt érdemes tudni, hogy úgynevezett interpretált nyelvek, azaz a forrásfájl nem szükséges lefordítani, hanem azt egy interpreter értelmezi. A futtatáshoz így bár mindig szükségünk van egy ilyen értelmezőre, de cserébe a nyelv általában platformfüggetlen tud lenni.

JavaScriptnél kliens oldalon ez az értelmező a böngészőben lévő JavaScript Engine szokott lenni, mint például a Google Chrome és Microsoft Edge által használt V8. A böngészőben futó JavaScript többek között módosítani tudja az oldal DOM szerkezetét és stílusát, sütiket és eseményeket képes kezelni (pl. `onLoad`, `onClick`), adatokat tud tárolni a `LocalStorage` felhasználásával, és még további számos műveletre képes. A nyelv teljesen integrált a HTML és CSS nyelvekkel. HTML-ben a bárhova elhelyezett `<script></script>` tagek közé tehetjük JavaScript kódunkat, amit a böngésző azonnal értelmezni és futtatni fog. Hosszabb, összetettebb kódoknál érdemes inkább az `src` attribútum használata, mellyel külső JavaScript állományt (`.js`) tudunk betölteni. Fontos megemlíteni, hogy bár a nyelv szabványosítva lett, a különböző böngészők mégis részben eltérően implementálják a JavaScriptet, illetve a régebbi verziójú böngészők nem biztos, hogy kompatibilisek bizonyos parancsokkal.

A JavaScript kód futtatásához azonban nincsen feltétlen böngészőre szükségünk, a nyelv szerver oldalon is használható, például a Node.js szerver oldali platformfüggetlen szoftverrendszer segítségével, mely szintén a V8 motorra épül, továbbá Windows alapú környezetben is futtathatók a `wscript.exe` és a `cscript.exe` segítségével.

Az idők során sok olyan programozási nyelv jött létre, mely köztes nyelvként JavaScript kódot generál. Ezeknek a nyelveknek a célja az hatékonyság és a megbízható kód írásának elősegítése. Ilyen nyelv például a TypeScript, a CoffeeScript, és a Dart, melyeknek bár a JavaScripttől eltérő a szintaktikájuk, de ugyanolyan alap JavaScripté tudnak átfordulni, így a böngészők értelmezni tudják őket. [1, 2]

1.2 Adatbázis

1.2.1 MySQL

A MySQL egy nyílt forráskódú, SQL (Structured Query Language) alapú relációs adatbázis-kezelő rendszer, melynek alapítói David Axmark és Michael Widenius voltak. A MySQL eredetileg a MySQL AB nevű svéd cég tulajdonában volt, melyet később a Sun Microsystems vásárolt fel. 2010-ben a Sun az Oracle részévé vált, és a MySQL a mai napig is az ő tulajdonukban van.

Egy relációs adatbázisban az adatok táblákba vannak rendezve, és az adatok között kapcsolatok lehetnek, amik az adatbázis struktúráját határozzák meg. Egy MySQL adatbázissal a felhasználók SQL nyelven interaktálhatnak, és különböző adatlétrehozó, szerkesztő, törlő, és lekérdező műveleteket hajthatnak végre, továbbá az egyes felhasználók jogosultságait is lehet kezelni. Az adatbázis-kezelő az egyes programnyelvekből egyedi illesztőfelületekkel érhető el, ilyen például PHP-nál a *mysqli*, Java-nál a JDBC, C és C++ programozási nyelvekben pedig az ODBC. A JavaScript esetén is számos mód van az adatbázissal való kommunikációra, mint például a MySQL Cluster vagy a JSDB. Jelen alkalmazás erre a Prisma nevű ORM-et alkalmazza, melyről a következő alfejezetben esik bővebben szó. [3, 4]

1.2.2 Prisma

A Prisma egy nyílt forráskódú ORM (Object-Relational Mapping/Mapper). Segítségével meg lehet határozni az adatbázis sémáját, és különféle CRUD (Create, Read, Update, Delete) műveleteket hajthatunk végre bármilyen Node.js vagy TypeScript backend alkalmazásban, továbbá számos adatbázist támogat, mint például a MySQL, PostgreSQL, SQLite, és a MongoDB.

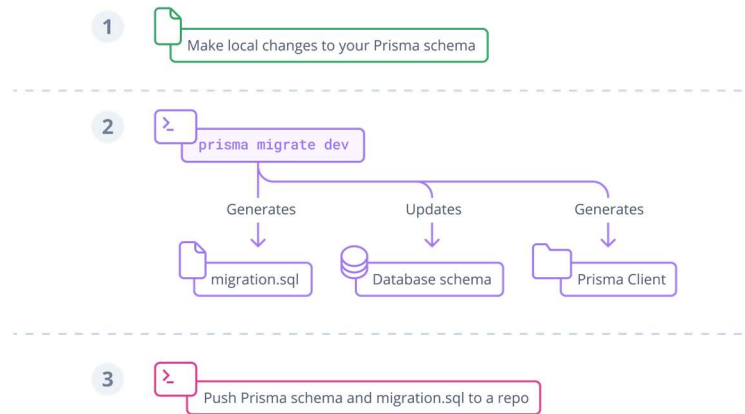
Az ORM-ek magas szintű adatbázis absztrakciót biztosítanak, és objektumorientált programozási felületet teremtenek az adatbáziskezeléshez, elrejtve annak a komplexitását. Általánosan egy ORM használatakor a modellek class-ok által vannak definiálva, és ezek vannak az egyes táblákhoz hozzárendelve. A Prisma azonban ettől egy kicsit eltér, mivel itt a modelleket egy „Prisma schema” -ban kell meghatározni, amiket utána az alkalmazásban a Prisma Client-en keresztül lehet használni az adatbázis műveletekhez, ezzel egyszerű típusbiztos adatkezelést biztosítva.

A Prismát alapvetően két fő részre lehet osztani, a Prisma Migrate-re, és a Prisma Client-re. A Prisma Migrate-et az adatbázis struktúrájának meghatározására és változtatására alkalmazzuk. A Prisma schema-ban ezt a következőképpen lehet végrehajtani:

```
1
2 generator client {
3   provider = "prisma-client-js"
4 }
5
6 datasource db {
7   provider = "mysql"
8   url       = env("DATABASE_URL")
9 }
10
11 model User {
12   id      Int      @id @default(autoincrement())
13   name    String?
14   email   String   @unique
15   recipes Recipe[]
16 }
17
18 model Recipe {
19   id          Int      @id @default(autoincrement())
20   name        String
21   description String
22   uploader    User     @relation(fields: [uploaderId], references: [id])
23   uploaderId  Int
24 }
25
```

1.1 ábra: Prisma schema példakód

A fenti 1.1-es ábrán egy egyszerű példa adatbázis sémája látható. Ez a kód, és az összes további kódrészlet, ahol nincs külön forrás megnevezve, saját készítésű. A generator rész (2-4. sorok) jelzi a Prismának, hogy Prisma Client-et szeretnénk generálni, a `datasource` rész (6-9. sorok) pedig meghatározza az adatbázis kapcsolatot. Ezután létrehozunk egy *User* és egy *Recipe* táblát autoinkrementált *id*-kkel, néhány további adatmezővel, és közöttük meghatározunk egy 1-N kapcsolatot. Az adattáblák tényleges létrehozásához már csak a `prisma migrate dev` parancsot kell futtatnunk a terminálban. A Prisma Client segítségével ezután már szabadon hajthatunk végre adatbázis műveleteket az alkalmazásunkból. Az alábbi 1.2-es ábrán ez a folyamat látható vizualizáltan:



1.2 ábra: Általános Prisma Migrate workflow [5]

A Prisma Client számos beépített függvénnyel rendelkezik az adatbázisunk adatainak manipulálására, mint például a `.findMany`, `.findUnique`, `.create`, `.update`, `.delete`, és még sok más. Az összes ilyen lekérdezés és művelet ezután sima JavaScript objektumokkal tér vissza. Egy egyszerű adatbeillesztés kódja a fentiekben létrehozott adatbázisba a következőképpen néz ki:

```

8
9  const user = await prisma.user.create({
10    data: {
11      name: "Demo User",
12      email: "user@demo.com",
13      recipes: {
14        create: {
15          name: "Demo recipe",
16          description: "Demo",
17        }
18      }
19    }
20  });
21

```

1.3 ábra: Prisma create példakód

A fenti 1.3-as ábra kódja a *User* táblába illeszt be egy új felhasználót, illetve a *Recipe* táblába egy hozzá kapcsolt új receptet. [5]

1.2.3 Docker

A Docker egy olyan számítógépes szoftver, amely operációs rendszer szintű virtualizációt végez konténerek futtatásával. A Docker, Inc. fejlesztette ki, és először 2013-ban jelent meg. A Docker előnye a virtuális gépekkel szemben, hogy sokkal kevesebb erőforrást fogyasztanak, mivel minden konténert egyetlen operációs rendszer kernel működtet. A konténereket a Docker

Engine host-olja. Az egyes konténerek egy úgynevezett image-ből vannak felépítve, ami az alkalmazások tárolására szolgál. Ezeket a Docker repository-ból lehet letölteni, és segítségével akár egy MySQL szervert vagy sok más adatbázis szervert és további alkalmazásokat felhúzhatunk, melyek teljesen elkülönülnek egymástól, és egymással jól meghatározott csatornákon kommunikálhatnak.

A Docker egy hasznos eszköze még továbbá a Docker Compose, ami egy Python-ban írt eszköz. Segítségével akár több konténerből álló docker környezeteket is fel tudunk húzni. Egy ilyen környezet leírásának mindig a `docker-compose.yml` fájlban kell szerepelnie. [6, 7]

1.3 Backend

1.3.1 Node.js

A Node.js egy nyílt forráskódú serveroldali JavaScript futtató környezet, mely a Google V8 motorján fut, és feladata a JavaScript kódok böngészőn kívüli futtatása, főként webszervereken. A Node.js-t Ryan Dahl hozta létre 2009-ben, és a további fejlesztését is ő vezette a Joyent támogatásával, manapság pedig az OpenJS Foundation irányítja ezt.

A Node.js segítségével dinamikus weboldalakat tudunk létrehozni, és nem csak a kliens oldalon, de a serveroldalon is JavaScript nyelven programozni, ahelyett, hogy ezekhez eltérő nyelveket kéne használnunk. Aszinkron I/O műveletekre képes és felépítése eseményalapú, mely callback műveleteket használ a folyamatok elvégéztekor, így elősegítve a skálázhatóságot és a túlterhelés minimalizálását.

Számos beépített modullal rendelkezik a gyakori műveletek kezelésére, mint például I/O műveletek, hálózatkezelés, kriptográfia, és még sok más, de mindezen túl még több ezer függvénykönyvtár érhető el hozzá, melynek többsége az npm [8] weboldalról elérhető. Az npm egy beépített Node.js csomagkezelő, mellyel a fent említett programokat lehet telepíteni parancssoron keresztül (`npm install <csomag neve>`), így egységesítve a rengeteg külső Node.js könyvtár kezelését. A szakdolgozatban tárgyalt alkalmazás is számos ilyen csomagot használ, mint például az express, bcrypt, multer, nodemailer, és a node-schedule. [9]

1.3.2 Express.js

Az Express.js egy nyílt forráskódú, gyors és személyre szabható Node.js keretrendszer backend webalkalmazások és azon belül API-k (Application Programming Interface) fejlesztésére. Az Express.js-t TJ Holowaychuk alapította, legelső verziója 2010-ben jelent meg. Az Express.js fő funkciója, hogy robosztus routing-ot szolgáltat az API-k létrehozásához, az alap Node.js-hez

képest sokkal gyorsabban lehet vele az alkalmazást lekódolni, és a végpontokat könnyebb kezelni. Egy egyszerű webszerver a következőképpen tud kinézni:

```

1
2 // Express könyvtár importálása
3 const express = require('express');
4
5 // App inicializálása
6 const app = express();
7
8 // GET útvonal létrehozása, mely válasznak egy szöveget küld vissza
9 app.get('/', function (req, res) {
10   res.send("Hello World!")
11 });
12
13 // Az app a 3000-es porton fut
14 app.listen(3000, () => {
15   console.log('Listening at Port 3000');
16 });
17

```

1.4 ábra: Express webszerver példakód

Ha a fenti 1.4-es ábrán szereplő alkalmazást elindítjuk, egy olyan szervert fogunk kapni, ami a 3000-es porton fut, és ott várja a kéréseket. Az alap GET kérésre pedig egy „200 OK” státuszú választ ad vissza, annak törzsében a „Hello World!” szöveggel. Ha a 10. sorban látható `res.send()` helyett `res.json()`-t használunk, és annak átadunk egy JS objektumot, akkor azt JSON formában lehet visszaküldeni, és ezt utána a kliens oldalon könnyen vissza is alakíthatjuk. A receptmegosztó alkalmazás is ezt a módszert alkalmazza, az adatátvitel kliens és szerver között JSON formátumú JS objektumokkal történik.

Úgynevezett „middleware” -ek használatával a beérkező kéréseket lehet kezelni. A middleware-ek olyan callback függvények, melyeknek hozzáférése van az alkalmazás kérés-válasz kommunikációjához, és ennek segítségével műveleteket hajthatunk végre a kérés beérkezése, és annak tényleges végrehajtása között. Ezt többek között az autorizáció ellenőrzésére is szokás használni. Bizonyos végpontokhoz nem lehet mindenkinek hozzáférése, hiszen akkor például bárki végrehajthatna admin műveleteket alkalmazásunkon. Azonban, ha middleware-t használunk, ellenőrizhetjük, hogy a kérés küldőjének van e joga az adott művelethez, és visszautasíthatjuk azt, mondjuk egy „401 Unauthorized” válasz visszaküldésével. Erre alább, az 1.5-ös ábrán láthatunk egy leegyszerűsített példát, mely az 1.4-es ábrán lévő kódot egészíti ki:

```

10 // Middleware admin jog ellenőrzésére
11 function adminAuthMiddleware(req, res, next) {
12   try {
13     // Aktuális user adatainak betöltése (pl. adatbázisból)
14     const user = getCurrentUser()
15
16     // Ha a felhasználó nem admin, hibát dobunk
17     if(!user.admin) {
18       throw 'Admin role needed.'
19     }
20
21     // Egyébként tovább mehetünk a következő funkcióra
22     next();
23   } catch (error) {
24     // 401-es hibakódot küldünk vissza az error üzenettel
25     res.status(401).send(error);
26   }
27 }
28
29
30 app.get('/', adminAuthMiddleware, function (req, res) {
31   res.send("Hello World!");
32 });
33

```

1.5 ábra: Middleware alkalmazása példakód

Amikor a futó webszerverre elküldjük a GET kérést, először az `adminAuthMiddleware` nevű middleware fog lefutni, mely lekéri az aktuális felhasználó adatait (ezt például lekérhetjük az adatbázisból miután a session-ből beazonosítottuk őt), majd ellenőrzi, hogy rendelkezik-e admin joggal. Ha nem, akkor 401-es hibakóddal fog válaszolni, egyébként pedig tovább engedi a route paramétereiben felsorolt következő függvényre. Middleware-ből természetesen nem csak egy lehet, a paraméterekben többet is írhatunk egymás után, és olyankor azok egymás után futnak le, amíg végig nem érünk rajtuk, valamelyikben visszaküldjük a választ, vagy visszatérünk.

Természetesen nagyobb alkalmazásoknál nem szerencsés, ha minden végpontunkat egy fájlban soroljuk fel, gyorsan átláthatatlanná tud válni, ezért érdemes a route-okat külön fájlokba szervezni. Ilyenkor létre kell hozni egy router modult az `express.Router()` meghívásával, majd azt az `app` helyére írjuk a route-ok meghatározásánál. A fájl végére elhelyezzük a `module.exports = router` utasítást, aminek köszönhetően az elkészített router fájlt a fő JS fájlban importálhatjuk, és az `app.use('/<route>', <routeName>)` paranccsal beköthetjük. Erre az alábbi képen láthatunk példakódot:



```

1  const express = require('express');
2  const router = express.Router();
3
4
5  router.get('/', function (req, res) {
6    res.send('Hello World!');
7  });
8
9  router.get('/hungarian', function (req, res) {
10   res.send('Helló, világ!');
11 });
12
13 module.exports = router;
14

```

```

1  const express = require('express');
2  const app = express();
3
4
5  const helloWorldRouter = require('./helloWorldRouter');
6
7  app.use('/hello-world', helloWorldRouter);
8
9  app.listen(3000, () => {
10   console.log('Listening at Port 3000');
11 });
12

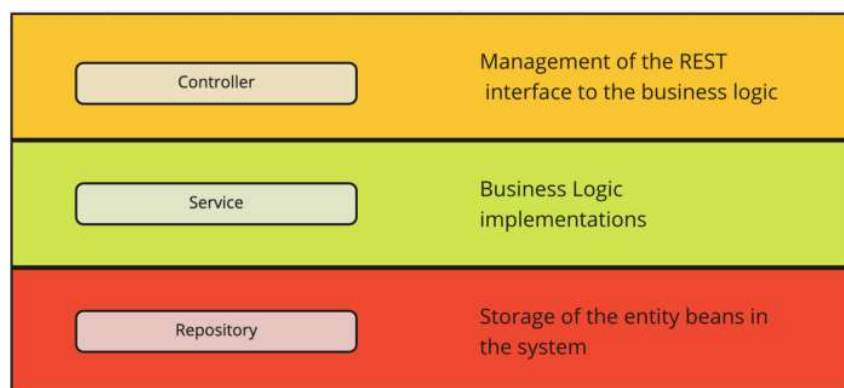
```

1.6 ábra: Router külön fájlban példakód

Az 1.6-os ábrán bal oldalon látható a `helloWorldRouter.js` router-fájl, jobb oldalon pedig az app fájl. Az alkalmazás így a `/hello-world` és a `/hello-world/hungarian` GET kéréseket képes kezelni. [10, 11, 12]

1.3.3 Controller-Service-Repository tervezési minta

A Controller-Service-Repository tervezési minta fő célja, hogy a kódot három rétegre bontsa annak feladata szerint. Ezek a minta nevében is szereplő Controller, Service, és Repository rétegek. Az egyes rétegek feladatai az 1.7-es ábrán láthatóak.



1.7 ábra: A Controller-Service-Repository minta rétegei [13]

A Controller réteg feladata csupán annyi, hogy a funkcionalitást felfedje, kívülről elérhetővé tegye, és továbbadja a feladatot az azt ténylegesen megoldó rétegeknek. A Service rétegben található minden üzleti logika, ez határozza meg magának az alkalmazás működésének a logikáját, szabályait. A Repository réteg pedig kizárólag az adatok tárolásával és lekérdezésével foglalkozik.

Ezen három réteg teljesen független egymástól, semmit nem tudnak a többinek a működéséről, csakis a saját feladataikra koncentrálnak. Ha az Service rétegnek valamilyen adatkezelésre van szüksége, akkor meghív egy Repository-t, de az nem érdekli, hogy az hogyan kezeli az adatot, és a Repository-t sem érdekli, hogy honnan hívták, ő csak megcsinálja, amit

kértek tőle. A Controller pedig csak továbbadja a feladatokat. A rendezettségén túl ez azért is jó, mert így sokkal könnyebben tesztelhető az alkalmazás, egyszerűbb beazonosítani, hogy a kód melyik részében lehet a hiba. A függetlenség továbbá az újrafelhasználhatóságot is elősegíti. [13]

1.4 Frontend

1.4.1 HTML

A HTML (HyperText Markup Language) egy jelölőnyelv, melynek célja weboldalak tartalmának kialakítása, és a tartalom strukturálása. Fontos, hogy a formázás nem feladata, azt majd a CSS fogja megtenni, a logikát pedig többek között a beágyazott JavaScript kód adhatja. A nyelv mára már internetes szabvánnyá vált a W3C támogatásával.

A HTML kódunkban úgynevezett tageket használunk, amivel tartalmi elemeket (pl. szövegek, képek, hivatkozások), illetve strukturális elemeket (pl. fejléc, lábléc, menüsor) tudunk elhelyezni az oldalon egy sima szöveges állomány segítségével, melyet a böngésző értelmez és megjelenít. Az aktuális HTML5 szabvány további újdonságokat is tartalmaz, mint például hang és videó lejátszó, grafikus és szemantikus elemek, új űrlapmező típusok stb. HTML-t jeleníthet még meg levelező program is, hiszen manapság már elterjedtek a HTML e-mailek, illetve aural böngészők, melyek felolvassák a weboldal tartalmát, vagy braille olvasók.

Minden HTML tagnek van egy típusa, illetve lehetnek további attribútumai. Az alábbi 1.8-as ábrán egy képet szúrunk be a weboldalra, hiszen típusa `img`, adunk neki egy `id`-t, azaz egy egyedi azonosítót (ez később az elemek szelektálásánál tud majd segíteni, például CSS írásakor), megmondjuk a kép forrását az `src` attribútummal, illetve egy alternatív szöveget, ha a képet nem tudnánk elérni az `alt` segítségével.

```

```

1.8 ábra: Kép beszúrása példa

Léteznek továbbá páros tagek is, mint például a `<p>...</p>` elem, mely egy paragrafust jelöl, és annak tartalma a két címke közé kerül. A HTML5-nek mondhatjuk előnyének, de hátrányának is, hogy bizonyos szinten megengedi a szabványtól való eltérést, a pontatlanságokat, így például attól, hogy egy záró taget elhagyunk, az oldal még meg fog jelenni. Ennek hátránya, hogy a kód programmal nehezebben feldolgozhatóvá válik, és romlik az átláthatósága. A szabvány szabályait erősen ajánlott betartani, és számos validátor létezik a

helyesség ellenőrzésére. Ezzel szemben az XHTML szabvány, mely a HTML4 XML alapú „újraírása”, szigorú szabályokkal rendelkezik. [14, 15, 16]

1.4.2 SCSS

Az SCSS (Sassy CSS) a Sass (Syntactically Awesome Style Sheets) CSS preprocesszor egyik szintaxisa. A Sass egy olyan stíluslap nyelv, mely segítségével használhatunk változókat, beágyazott szabályokat, függvényeket, és még sok más dolgot, ami az alap CSS-ben nem feltétlen létezik. Persze mivel ezeket a böngésző ilyen formában nem tudja értelmezni, ezért a stíluslap sima CSS-é fordul át.

Az eredeti Sass szintaxis indentációt használ a kód blokkok elválasztására, és új sor karaktereket a szabályok között, míg az újabb SCSS a már CSS-ből megszokott kapcsos zárójeleket és pontosvesszőket. Ez többek között azért is jó, mert kompatibilis az alap CSS-el, minden CSS kód egyben valid SCSS kód is.



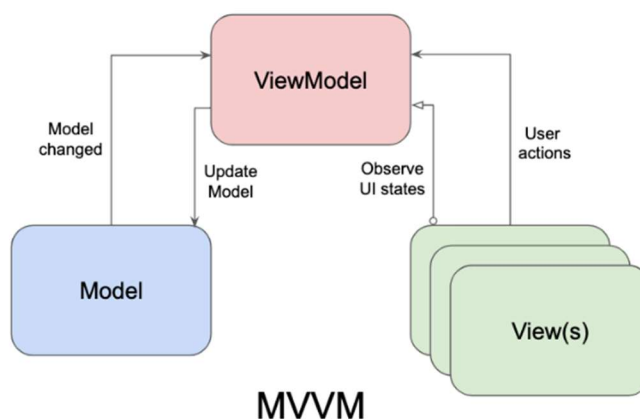
1.9 ábra: SCSS kód, és annak CSS megfelelője [17]

Az 1.9-es ábrán bal oldalon látható az SCSS kód beágyazott szabályokkal, és jobb oldalt a CSS kód melyre az átfordul, ezt fogja a böngésző megkapni. Itt is ugyanúgy használhatjuk a CSS3-ban megszokott szelektorokat (pl. #id-name, .class-name, body), pseudo-osztályokat (pl. :hover, :first-child) és szabályokat (pl. background-color, margin). [17, 18]

1.4.3 Vue.js és az MVVM tervezési minta

A Vue.js egy MVVM (Model-View-Viewmodel) tervezési mintájú frontend JavaScript keretrendszer mely felhasználói felületek létrehozására, egyoldalas alkalmazások felépítésére használható. Megalkotója Evan You, aki 2014-ben adta ki először a keretrendszert. A projekt Vue 3-at használ, mely a legújabb és legtámogatottabb verzió.

Az MVVM tervezési minta három részre osztja az alkalmazást. A Model reprezentálja az alkalmazás adatait, illetve az üzleti logikát, függvényeket. A View kizárólag a megjelenítéssel foglalkozik, az oldal elrendezését, struktúráját, és kinézetét határozza meg. Megjeleníti a Model adatait, és fogadja a különböző külső eseményeket, például kattintás, billentyű lenyomás. Ezt a két elemet pedig a View-Model köti össze, a kettő közötti automatikus kommunikációt teszi lehetővé, emiatt Binder-nek is szokás nevezni. A tervezési minta fő célja, hogy a View réteget lehessen önállóan kezelni [19]. Az MVVM egyes részei közötti kapcsolat az alábbi 1.10-es ábrán is jól látható:



1.10 ábra: Az MVVM tervezési minta [20]

A Vue.js a HTML, CSS, és JavaScript alapokra épít, és ezek fölé deklaratív és komponens alapú programozási modellt biztosít. A deklaratív rendereléssel a HTML kimenetet a JavaScript állapota szerint automatikusan meg lehet változtatni. Ha egy változó új értéket kap a JS kódban, akkor az azonnal megváltozik a HTML-ben is, és hogyha például egy input mező értéke össze van kötve (`v-model`) egy változóval, akkor a bevitt érték változásakor a változó értéke is az új érték lesz. A meglévő adatok szerint egy elemet vagy komponens többször is megjeleníthetünk, más-más hozzá tartozó értékekkel (`v-for`), vagy egy logikai kifejezés értéke szerint megjeleníthetjük, vagy sem (`v-if`, `v-show`), és még sok más lehetőség áll rendelkezésünkre. Eseménykezelésben is sok segítséget kapunk, hiszen például a kattintás figyeléséhez csupán az `@click`-et kell elhelyeznünk a kívánt tagben, és ennek értékeként megadhatjuk, hogy mi a teendő (ennek általában egy függvényt szokás megadni, de alapvetően bármilyen kód lehet).

A Vue nagy fókuszot fektet az egyfájlos komponensek (Single File Components, SFC) használatára. Lényege, hogy az oldal sablonja (HTML), stílusa (CSS), és logikája (JavaScript) mind egyetlen fájlban (`.vue`) legyenek. Ez bár egy tradicionális webfejlesztői nézetből furcsa

lehet, de a gondolat mögötte az, hogy az ugyanarra vonatkozó dolgokat tartsuk egy helyen, és bár a HTML, CSS, és JS kódoknak más a formátuma, de mind ugyanarra a komponensre vonatkoznak a webalkalmazásban. Ez sok esetben egyszerűbb, átláthatóbb kódot eredményezhet, de használata nem kötelező. Egy példa egy egyszerű egyfájlos komponensre az alábbi 1.11-es ábrán látható:

```

2   <template>
3     <p class="helloWorld">{{helloText}}</p>
4     <button @click="changeToWelcome">Click me!</button>
5   </template>
6
7   <script>
8     export default {
9       name: "Demo",
10      data(){
11        return {
12          helloText: "Hello World!"
13        }
14      },
15      methods: {
16        changeToWelcome(){
17          this.helloText = "Welcome World!"
18        }
19      }
20    }
21  </script>
22
23  <style>
24    .helloWorld {
25      font-size: 16px;
26    }
27  </style>

```

1.11 ábra: SFC komponens példakód

Ahogy látható, komponensünkben egy bekezdés, és egy gomb található. A bekezdés tartalma a *helloText* változó értéke lesz, mely kezdetben a „Hello World!”. Ha a gombra rákattintunk, lefut a *changeToWelcome* metódus, mely megváltoztatja a változó értékét „Welcome World!” -re, és így a bekezdés tartalma is innentől kezdve ez lesz. A bekezdésre továbbá egy kis stílust is raktunk a `<style></style>` tagek között.

A Vue kétfajta API stílussal rendelkezik, az Options és a Composition API-al. Az Options API esetén, melyet jelen project is használ, egy komponens logikáját egy nagy objektumon keresztül adhatjuk meg (gyakorlatilag beépített *opcióink* vannak), például az adatokat a *data* kulccsal, a metódusokat a *methods*-al, és így tovább. Az objektum elemeit azon belülről a *this*-el érhetjük el. A fenti 1.11-es ábrán látható kód is ilyen. Ezzel szemben a Composition API beimportált API függvényekre hagyatkozik a logika megadásához. [21, 22]

A Vue önmagában nem biztosít lehetőséget routing-ra, ehhez a Vue Router csomagra van szükségünk. A Vue Router dinamikus útvonalkezelésre ad lehetőséget, kezelni tudjuk vele

a route paramétereit, egymásba beágyazva adhatjuk meg őket, és még sok más hasznos funkcióval rendelkezik. Az alábbi 1.12-es ábrán egy route definiálása látható:

```

1  import {createRouter, createWebHistory} from "vue-router";
2
3  const routes = [
4    {
5      name: "Home",
6      path: "/home",
7      component: () => import('@/views/Home'),
8    },
9  ],
10
11 export default createRouter({
12   history: createWebHistory(),
13   routes
14 })

```

1.12 ábra: Route létrehozása Vue Router-rel

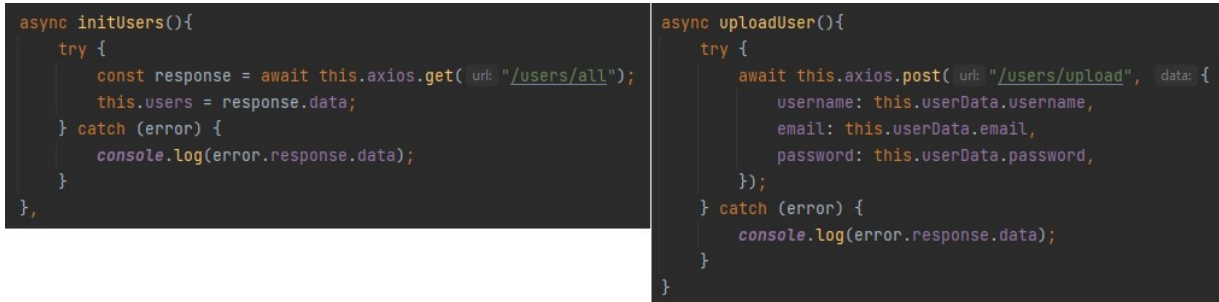
Ennek hatására a `/views/Home.vue` komponensünk a „/home” útvonallal lesz elérhető, továbbá pedig a 12. sor hatására létrejönnek a szokványos HTML5 előzmények. Ahhoz viszont, hogy komponenseink ténylegesen megjelenjenek amikor rájuk navigálunk, ez még nem elég. A `<router-view>` tag mindig azt a komponenst fogja megjeleníteni, ami az URL-ben szerepel. Ezt bárhova el lehet helyezni, de leggyakrabban az alkalmazás fő komponensébe (`App.vue`) szoktuk rakni. Komponensek közötti navigálásra a `<router-link>` taget lehet használni. Ez nagyon hasonló az `<a>` taghez, de csak komponensekhez használhatjuk. Ennek köszönhetően a Vue Router meg tudja változtatni az URL-t az egész oldal újra töltése nélkül (mely egyoldalas alkalmazásnál hacsak lehet elkerülendő). Ugyanilyen navigálást a JS kódból is végre tudunk hajtani a `$router.push()` parancs segítségével. [23]

1.4.4 Vite

A Vite egy lokális fejlesztési szerver, melyet a Vue megalkotója, Evan You fejlesztett ki 2020-ban, és a Vue projektekkel alapértelmezetten használatos. A Vite a JavaScripten kívül a TypeScriptet és a JSX-et is támogatja. Az egyik fő funkciója a Hot Module Replacement (HMR). Ennek köszönhetően, amikor az alkalmazás egy oldalát szerkesztjük, majd a fájlt elmentjük, nem kell az egész alkalmazásnak újra fordulnia, csak a szerkesztett fájlnak. A HMR-t natív ESM (ECMAScript Modules) fölött végzi el. A Vite a szerver indítását is fel tudja gyorsítani, például a függőségeket és a forráskódot külön veszi, külön-külön optimalizálva azok kezelését. Mindezekon túl a Vite még számos gyorsító funkcióval rendelkezik, hogy megkönnyítse a fejlesztés folyamatát. [24, 25]

1.4.5 Axios

Az Axios egy Node.js csomag, amivel http kéréseket lehet küldeni kliens oldalon a böngészőből vagy Node-ból.



```

async initUsers(){
  try {
    const response = await this.axios.get( url: "/users/all");
    this.users = response.data;
  } catch (error) {
    console.log(error.response.data);
  }
},

async uploadUser(){
  try {
    await this.axios.post( url: "/users/upload", data: {
      username: this.userData.username,
      email: this.userData.email,
      password: this.userData.password,
    });
  } catch (error) {
    console.log(error.response.data);
  }
}

```

1.13 ábra: Axios GET és POST kérések küldése

Az 1.13-as ábrán bal oldalon egy egyszerű axiosos GET kérés, jobb oldalt pedig egy POST kérés látható. Az Axios Promise alapon működik, ezért használni kell a JS async és await funkcióit, hogy megvárjuk a választ, különben csak egy Promise objektumot fogunk visszakapni.

GET kérés esetén az `axios.get()`-et kell meghívni, és annak átadni az URL-t ahova a kérést küldeni szeretnénk. Itt azonban azt láthatjuk, hogy nem egy teljes URL van megadva. Ez azért lehetséges, mivel az `axios.default.baseUrl`-nek értékül adhatunk egy stringet, mely utána a függvényhívásoknál automatikusan hozzáfűződik a megadott URL elejére. Miután megvártuk a válaszukat, azt visszaadja a függvény. A válasz törzse a `.data` adattagban található, de más részek is elérhetőek, mint például a fejléc (`.headers`) vagy a státuszkód (`.status`). A válasz törzsét ezután egyenesen változóba menthetjük, mivel az Axios automatikusan JS objektummá alakítja a kapott JSON objektumot.

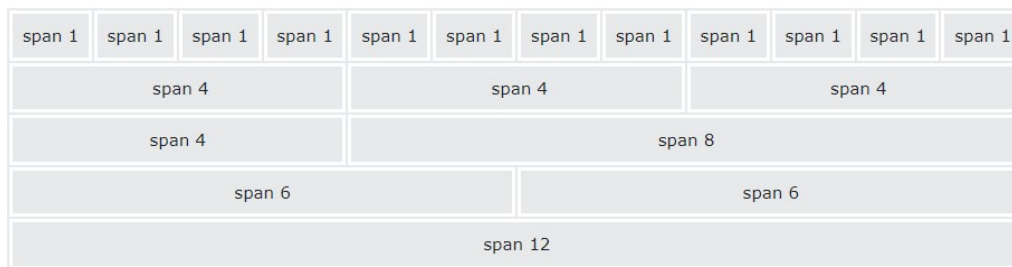
A POST kérésnél már a törzset is meg kell adnunk. Ezt egy objektum formájában tehetjük meg, mely küldéskor JSON-é fog alakulni. Természetesen `multipart/form-data` formátumban és meg lehet adni az adatokat. Ilyenkor létre kell hozni egy új `FormData` objektumot (`new FormData()`), majd ahhoz hozzáfűzni a kívánt névvel az adott adatokat (`formData.append('field_name', 'field_value')`). Ezután át kell adnunk ezt az objektumot az Axiosnak a szokásos módon, és beállítani a kérés fejlécében a `Content-Type`-ot.

Az Axios kéréseknek konfigurációt lehet meghatározni, szintén objektum formájában, melyet a hívás paramétereiben kell elhelyezni. Ha valamiért szükséges, akkor ezen keresztül is megadhatjuk a kérés típusát, az URL-t, POST esetében az adatokat, fejléc beállításokat stb. [26]

1.4.6 Bootstrap

A Bootstrap egy nyílt forráskódú CSS keretrendszer, mely a reszponzív frontend webalkalmazások fejlesztését segíti, a mobiltelefonokat előtérbe helyezve. Legfrissebb verziója a Bootstrap 5, melyet ez az alkalmazás is használ. HTML-ből, CSS-ből, és opcionálisan JavaScript-ből áll, egyedi grid rendszerrel rendelkezik a reszponzivitás elősegítéséhez, és számos előre megalkotott reszponzív sablon elemmel tud szolgálni, legyen az tipográfiai, űrlap, gomb, navigáció, legördülő menü, vagy modal, és még számtalan további lehetőség áll rendelkezésünkre. Rengeteg előre definiált osztály létezik, melyet a HTML elemünk `class` attribútumába elhelyezve az megkapja a kívánt stílust.

A Bootstrap grid rendszere 12 egyenlő oszlopra osztja az oldalt. Az oldalon lévő adott elemek esetén class-ok segítségével megadhatjuk, hogy hány oszlopot foglaljon az el. Az oszloprendszer elképzelésében az 1.14-es ábra is segíthet:



1.14 ábra: Bootstrap Grid oszlopok [27]

Az oszlopokat megadhatjuk általánosan minden képernyőre, vagy képernyőméretek szerint változóan is. Ha csak az alap `col-*` osztályt használjuk (ahol a `*` helyén az oszlopok kívánt száma van), akkor minden képernyőn az adott arányok fognak érvényesek lenni. Vonatkozhatnak ezek azonban különböző képernyőméretekre is. Ezen osztályok a `.col-*` ($<576\text{px}$), `.col-sm-*` ($\geq 576\text{px}$), `.col-md-*` ($\geq 768\text{px}$), `.col-lg-*` ($\geq 992\text{px}$), `.col-xl-*` ($\geq 1200\text{px}$), és a `.col-xxl-*` ($\geq 1400\text{px}$). Persze nem szükséges mindig mindet megadni, ilyenkor az imént leírt relációk fognak érvényesülni, például, ha csak a `.col-md-*` osztályt alkalmazzuk, akkor minden 768px széles vagy annál nagyobb képernyőn érvényesülni fog a szabály. Ha nem adunk meg oszlopszámot, akkor az elemek automatikusan egyenlő szélességűek lesznek. [27, 28]

1.4.7 Pinia

A Pinia egy Vue-hoz készült keretrendszer, mely intuitív, típusbiztos, és rugalmas tárolót biztosít a Vue alkalmazásoknak. A Pinia segítségével globális tárhelyet tudunk létrehozni az weboldalunkon, melyben tárolni tudjuk az app állapotát az oldalak és komponensek felett.

Támogatja továbbá a Hot Module Replacement-et, a tárolók az oldal újra töltése nélkül frissülnek. Az alábbi 1.15-ös ábrán egy Pinia tárolóra láthatunk példát:

```

1  import {defineStore} from "pinia";
2
3  export const useUserStore = defineStore("user", {
4    state: () => {
5      return {
6        loggedIn: false,
7      }
8    },
9
10   actions: {
11     login(){
12       this.loggedIn = true;
13     },
14
15     logout(){
16       this.loggedIn = false;
17     }
18   }
19 })

```

1.15 ábra: Pinia tároló példakód

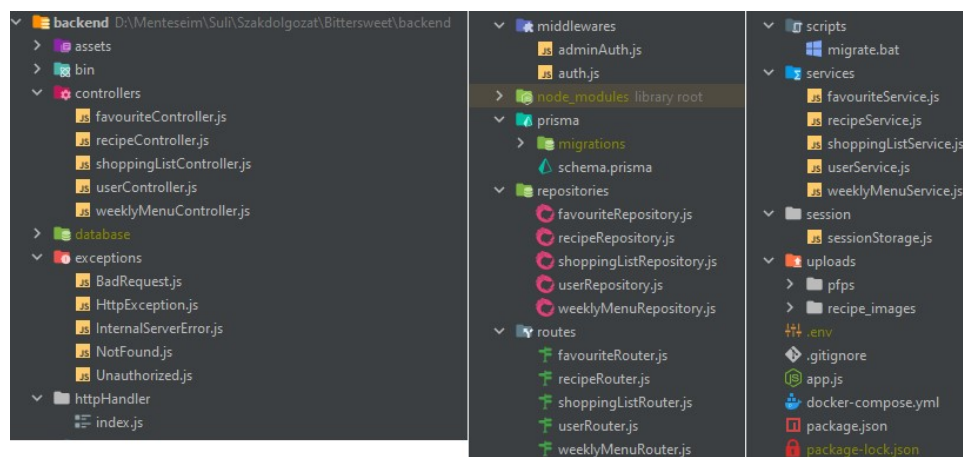
Az tároló állapotát a *loggedIn* változó értéke fogja adni (itt persze több változónk is lehetne), az *actions*-ön belül pedig globálisan használható függvények vannak. A tárolót ezután már csak importálni kell az adott komponensben, és onnantól kezdve elérhetővé válik. [29]

2. Receptmegosztó weboldal

2.1 A rendszer felépítése

2.1.1 Szerver oldal

Az alkalmazás szerver oldali felépítése az alábbi 2.1-es képen látható:



2.1 ábra: Alkalmazás szerver oldali felépítése

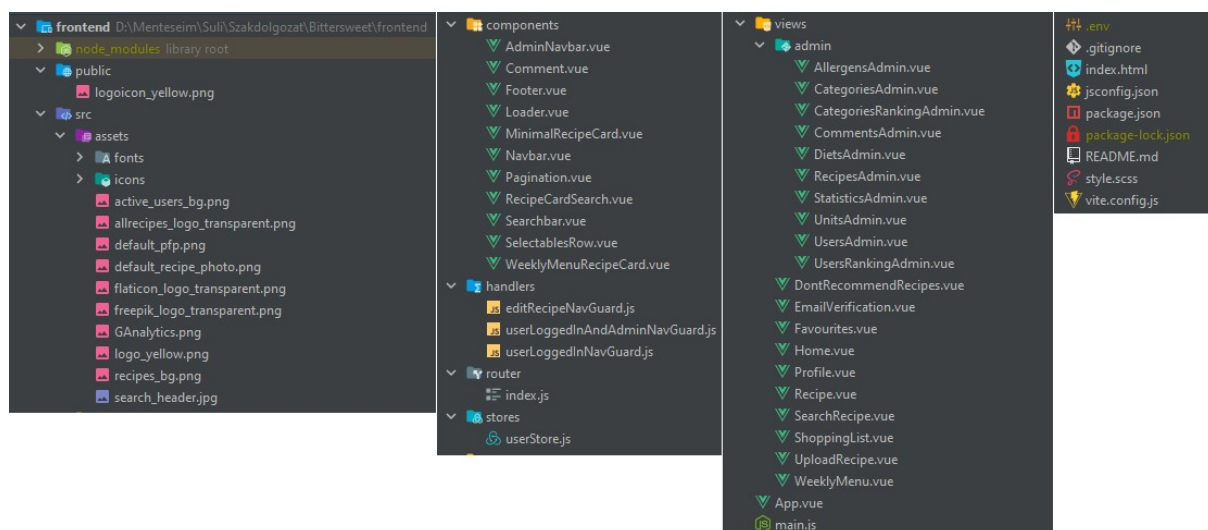
Ahogy láthatjuk és már az 1.3.3-as fejezetben tárgyaltuk, a backend Controller-Service-Repository tervezési mintát alkalmaz. Az egyes API végpontok funkciójuk szerint külön router fájlokba vannak kategorizálva a `/routes` mappában, a `/controllers`, `/services`, és `/repositories` mappák pedig a tervezési minta adott rétegeihez tartozó fájlokat tartalmazzák.

Mindezen kívül még az `/exceptions` mappában az egyes http hibaüzeneteknek megfelelő osztályokat találhatunk. Ezek rendkívül egyszerűek, fő céljuk, hogy tudják magukról a válaszban visszaküldendő státuszkódot, így azt fejlesztéskor nem kell mindig külön megadni, elég csak a hibaüzenetet átadni a konstruktornak. További segítséget nyújt még ebben a `/httpHandler/index.js` fájlban található `sendHttpException` függvény, mely a `res` (response) objektumot, és egy ilyen exception objektumot vár, és automatikusan elküldi ennek megfelelően a választ.

A `/middlewares` mappában a felhasználó autentikációhoz szükséges middleware-ek találhatók, a `/session/sessionStorage.js` pedig a session kezelésben nyújt segítséget. Az `/uploads` mappában a felhasználók által feltöltött képek találhatók. A mappákon kívül az `/app.js` fájl a szerver fő JavaScript fájlja, ebben vannak magához az alkalmazáshoz többek között a route-ok is hozzákapcsolva, továbbá sütikezelő, CORS beállítások stb. A `/docker-compose.yml` fájlban található a MySQL adatbázis konténerének beállításai, mely alaphoz a 3036-os porton fut (a backend szerver portját a `.env` fájlban kell megadni). Az adatbázis tábláinak létrehozásához szükséges Prisma séma a `/prisma/schema.prisma` fájlban található.

2.1.2 Kliens oldal

Az alkalmazás kliens oldali felépítése az alábbi 2.2-es ábrán látható:

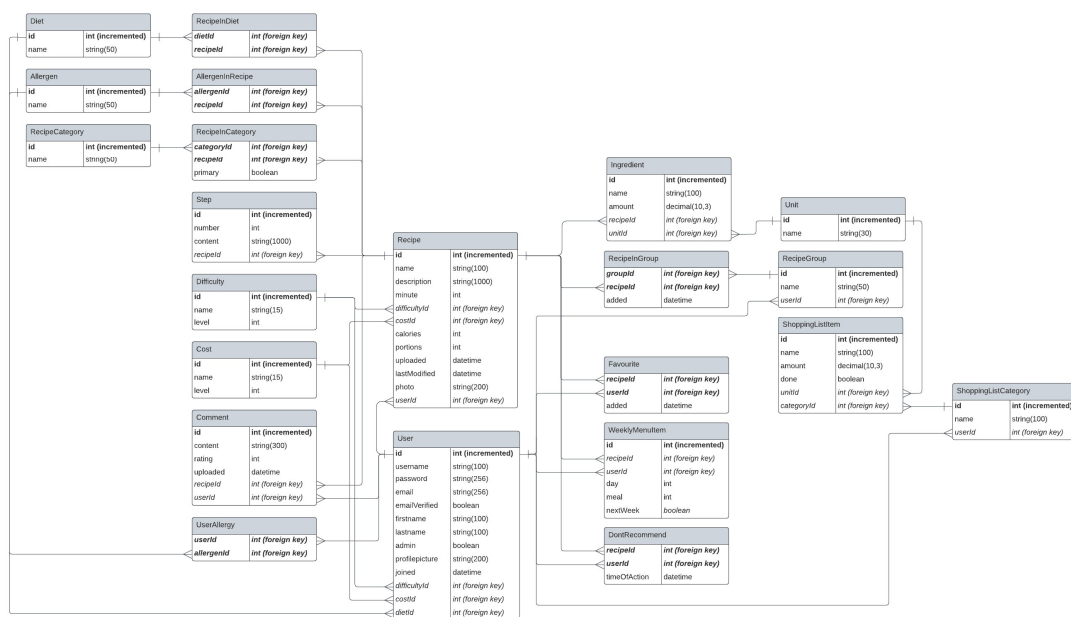


2.2 ábra: Alkalmazás kliens oldali felépítése

Az alkalmazás fő vue fájlja a `/src/App.vue`. Ebbe kerül mindig beágyazásra az aktuális URL-hez kapcsolódó view a korábban tárgyalt `<router-view>` segítségével. Ezek az `/src/views` mappában találhatóak, továbbá az ezekben felhasznált külön komponensek az `/src/components` mappában láthatók. A fő, mindent összefogó JS fájl az `/src/main.js`. Ez többek között hozzáköti az alkalmazáshoz a használt külső könyvtárakat, sűtiket kezel, tartalmazza az axios beállításait, és végül mount-olja az appot. Az `/src/router` mappában az a Vue Router-es fájl található, mely a weboldal útvonalait definiálja, és hozzájuk köti az adott view-kat. Az `/src/handlers` mappa úgynevezett nanguard-okat tartalmaz, melyek szabályokat határoznak meg a felhasználók oldalakhoz való hozzáféréssel kapcsolatban, megakadályozzák, hogy valaki olyan URL-re navigáljon, amihez nincs joga. Az `/src/stores/userStore.js` egy Piniás tároló, mely tartalmazza az aktuálisan bejelentkezett felhasználó adatait, így azokat nem kell minden oldalon, ahol szükség van rá, külön lekérni, hanem globálisan elérhetőek. Végül pedig, az `/src/assets` mappában találhatóak a felhasznált képek, ikonok, és betűtípusok. Az ikonok a Flaticon [30] weboldaltól származnak, a fejléc kép (search_header.jpg) a Freepik [31] oldalról, a betűtípusok pedig a Google Fonts [32] könyvtárból. Az alkalmazás képernyőtervei Figma [33] használatával készültek.

2.1.3 Az adatbázis struktúrája

Mivel az alkalmazás MySQL adatbázist használ, így egy relációs adatbázisséma határozza meg az adatbázis struktúráját. Az alábbi 2.3-as ábrán az ez által meghatározott adattáblákat, és a közöttük lévő kapcsolatokat láthatjuk:



2.3 ábra: Az adatbázis struktúrája (elsődleges kulcsok félkövéren, külső kulcsok dölten)

Ahogy látható, az adattáblák főként a Recipe és a User táblák köré épülnek, ezek számos adatmezővel és sok kapcsolattal rendelkeznek. A Difficulty és a Cost táblák tartalma fix, csupán az ismétlés elkerülését szolgálják, a különböző szintek nevét így elég csak egyszer eltárolni. Hasonló táblák még továbbá a Unit, Diet, Allergen, és RecipeCategory, annyi különbséggel, hogy bár ezek globálisan minden felhasználónál ugyanazok lesznek (pl. select input opciói), az admin felhasználóknak hozzáférésük van hozzájuk, tudnak újat létrehozni, szerkeszteni, vagy törölni, ha szükségesnek látják. Az utóbbi három tábla N:M kapcsolatban áll a Recipe táblával, illetve az Allergen a User táblával is, így ezek között összekapcsoló táblák vannak.

A Recipe táblához az Ingredients tábla is kapcsolódik, és az adott hozzávalókhöz a mértékegységeket a Unit tábla tárolja. Az ebben a táblában található mértékegységek érhetőek el a bevásárlólista elemeinél is, így ez a ShoppingListItem táblához is kapcsolódik.

A receptek az egyes felhasználók által létrehozott csoportokba is tartozhatnak, így a Recipe és RecipeGroup táblák között is N:M-es kapcsolat van, segédtáblával. Maga a Favourite tábla is tulajdonképpen csak egy összekapcsoló tábla a Recipe és User között, mely tárolja, hogy melyik felhasználó melyik receptet kedvencezte be, mint ahogy a DontRecommend tábla is, melybe akkor kerül be egy recept-felhasználó páros, ha a felhasználó nem szeretné a heti menüjében az adott receptet megkapni. A WeeklyMenuItem ettől annyiban tér el, hogy azt is tárolnia kell, hogy melyik hét melyik napjának melyik étkezéséhez tartozik a rekord, így a *userId*, *nextWeek*, *day*, és *meal* attribútumok négyese egyedi.

Az alkalmazáshoz mellékelt adatbázisban a példareceptek az Allrecipes [34] weboldalról származnak.

2.2 Jogosultságok

Az alkalmazásban alapvetően 4 jogosultsági szint különböztethető meg, bár kettő közülük eléggé hasonló. Attól függően, hogy az adott felhasználó melyikbe tartozik, különböző oldalak és funkciók lesznek elérhetőek. Ezen jogosultságok a *guest*, *unverified*, *verified*, és az *admin*.

A *guest* tulajdonképpen egy vendég felhasználót jelent. A receptek böngészéséhez és megtekintéséhez nincs szükség regisztrációra és bejelentkezésre, a vendég felhasználóknak erre van joga. Feltölteni azonban már nem tud receptet, illetve az oldal összes többi fő funkciója is bejelentkezett felhasználóhoz kötött. Ha esetleg egy ilyen felhasználónak már van profilja, de elfelejtette hozzá a jelszót, akkor kérhet újat az email címe megadásával.

Az *unverified*, avagy nem hitelesített felhasználó még hasonló a vendéghez. Egy vendég felhasználó regisztráció után válik ilyenné. Ekkor a profil már létrejön az adatbázisban, de amíg

a megadott email cím nincs hitelesítve, addig a bejelentkezés vissza lesz utasítva. Ilyenkor a felhasználó a kapott emailben lévő link segítségével hitelesítheti magát.

A *verified* felhasználók azok, akik regisztráltak, email címüket hitelesítették, és bejelentkeztek az alkalmazásba. Előttük már a weboldal legtöbb funkciója feltárul. Tölthetnek fel új receptet az oldalra, és sajátjaikat később szerkeszthetik vagy törölhetik, az egyes recepteket kedvenceik közé rakhatják és azokat csoportosíthatják, a receptekre kommenteket, értékeléseket írhatnak, vezethetik bevásárló listájukat, szerkeszthetik profiljukat és a heti menüvel kapcsolatos preferenciáikat, melyet az oldal hetente legenerál nekik. A heti menüben utána további változtatásokat is végezhetnek.

Az *admin* felhasználók rendelkeznek minden joggal, amivel a *verified* felhasználók is, ezen felül pedig további adminisztrátori jogosultságokkal is. Hozzáférésük van az Admin oldalhoz, amin keresztül az alkalmazás felhasználóit, receptjeit, és hozzászólásait kezelheti, szerkesztheti, törölheti. Ezek nagyrészt a moderálás lehetőségét szolgálják. Az admin felhasználó tud továbbá más felhasználókat adminná tenni, vagy ezt a jogot esetleg megvonni, illetve ugyanezt a verifikáció állapotával is megteheti. Mindezen túl kezelni tudja az alkalmazás globális mértékegységeit, diétáit, allergénjeit, és recept kategóriáit, tud újat hozzáadni, vagy létezőt szerkeszteni és törölni. A weboldal statisztikáit is megtekintheti.

Ezen jogosultságokat mind szerver, mind kliens oldalon ellenőrizni kell műveletek végrehajtása előtt. A backend-en ezt middleware-ek végzik el, melyek az adott végpont meghívásakor először ellenőrzik a jogosultságokat, és ez alapján vagy tovább engedik a kérést, vagy hibát küldenek válaszul. Frontend oldalon a navigáció szempontjából is fontos az ellenőrzés, melyekre *navguard*-ot használunk. Mielőtt egy route-ra rálépünk, lefut az aktuális `beforeRouteEnter(to, from, next)` függvény, mely nem megfelelő jogosultság esetén közbelép, és az eredeti cél helyett a főoldalra navigál át. Az alkalmazásból egy *navguard* és egy *middleware* az alábbi 2.4-es ábrán látható:

```

1 import {useUserStore} from "@stores/userStore.js";
2
3 export const beforeRouteEnter = (to, from, next) => {
4   if(!useUserStore().loggedIn) {
5     next({name: 'Home'})
6   } else {
7     next()
8   }
9 }

```

```

3 const {session} = require("../session/sessionStorage");
4 const Unauthorized = require("../exceptions/Unauthorized");
5
6 module.exports = (req, res, next) => {
7   try {
8     const sessionToken = req.headers.authorization;
9     if(!session[sessionToken]){
10       throw new Unauthorized("User not logged in.");
11     }
12
13     const storedSessionToken = session[sessionToken];
14     if(storedSessionToken.isExpired()){
15       delete session[sessionToken];
16       throw new Unauthorized("Token expired.");
17     }
18
19     next();
20   } catch (exception){
21     res.status(exception.code).json({errorMessage: [exception.message]})
22   }
23 }

```

2.4 ábra: Bejelentkezett felhasználó ellenőrzése (bal: navguard, jobb: middleware)

Mivel az, hogy a felhasználó be van a jelentkezve, a userStore-ban van eltárolva, így a navguard csupán ezt ellenőrzi. A middleware viszont magát a session-t nézi, először ellenőrzi, hogy az létezik e, majd ha igen, akkor azt is, hogy lejárt e már. Ha úgy találja, hogy a felhasználó nincs bejelentkezve, akkor hibát dob, egyébként pedig tovább engedi a kérést.

2.3 Regisztráció

Az oldalra való regisztrációhoz a vendég felhasználónak meg kell adnia egy felhasználónevet, email címet, és egy jelszót. Mind a felhasználónévnek, mind az email címnek egyedinek kell lennie az adatbázisban, már létezővel nem lehet regisztrálni. A felhasználónév maximum 100 karakter, az email cím pedig 256 lehet, az utóbbi mivel ez egy általánosan elfogadott felső határ egy email cím hosszára [35]. Az email cím formátumának helyességét továbbá regex ellenőrzi. A jelszónak legalább 6 karakter hosszúnak kell lennie.

A regisztrációs adatok frontend és backend oldali sikeres validálása után a User táblába kerülnek. A jelszó a bcrypt jelszó hash-elő segítségével van titkosítva, melyet jelen project a bcrypt npm csomag segítségével vesz igénybe.

```

const salt = await bcrypt.genSalt(10);
userData.password = await bcrypt.hash(userData.password, salt);

```

2.5 ábra: Jelszó titkosítása bcrypt segítségével

A fenti 2.5-ös ábrán látható egy felhasználói jelszó titkosítása, mielőtt az az adatbázisba kerülne. A titkosításhoz először salt-ot generálunk, majd elvégezzük a hash-elést.

Miután a regisztráció megtörtént, a felhasználónak hitelesítenie kell a megadott email címét, mielőtt bejelentkezhetne új fiókjába. Egy sikeres regisztráció után a rendszer emailt küld az adott címre, melyben egy 15 percig életben lévő hitelesítő link található. Erre kattintva

verifikálható az email cím, illetve maga a felhasználói fiók. Az alábbi 2.6-os ábrán is ez a folyamat látható:



2.6 ábra: Email hitelesítés folyamata

A hitelesítő linkek JSON Web Token-t használnak, melynek segítségével JSON objektumokat tudunk titkosított formában elhelyezni az URL-ben. A titkosításhoz meg kell adni egy úgynevezett secret-et, aminek segítségével később vissza fogjuk tudni fejteni a titkosított token-ből az eredeti objektumot. A secret természetesen minél hosszabb, annál jobb, jelen esetben 42 karakter hosszú. Az alkalmazásban a token-eknek egy 15 perces lejáratú idő is meg van adva. A token tartalmazza az adott felhasználó `userId`-jét, ami alapján tudni lehet, hogy melyik felhasználói fiókot lehet a linkkel verifikálni. Ha a token már lejárt a linkre kattintáskor, akkor a felhasználói fiók automatikusan törlődik, hogy újra lehessen regisztrálni a rendszerbe.

Maga az email a *nodemailer* nevű npm csomag segítségével kerül elküldésre. Ehhez először létre kell hoznunk egy *transporter* objektumot a `nodemailer.createTransport()` függvény segítségével. Itt adhatjuk meg az email szolgáltatás típusát (pl. gmail), illetve a használni kívánt email címet és az ahhoz tartozó jelszót. Ezeket jelen esetben a `.env` fájlban kell elhelyezni `EMAIL_USER` és `EMAIL_PASS` néven. Ezután ezen objektumon keresztül már el is küldhetjük az emailt az adott címre a `sendMail()` funkcióval, mely akár HTML email is lehet. Az alkalmazáshoz készült saját email cím, mely a `bittersweetRecipes@gmail.com`.

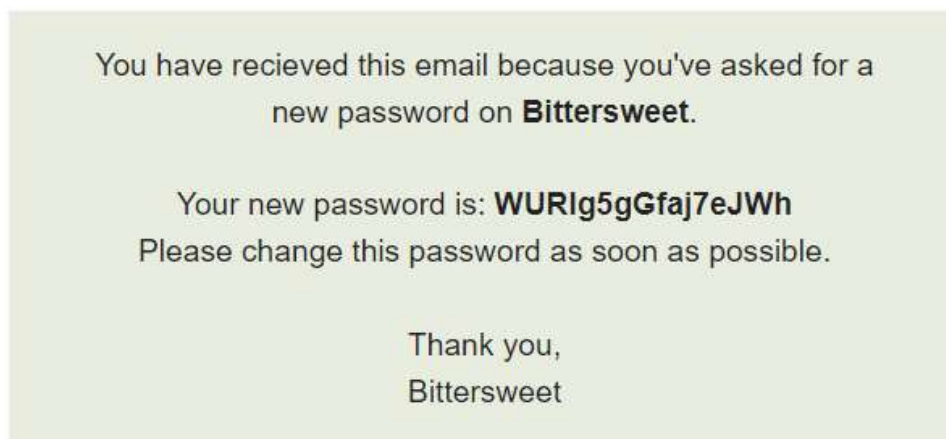
Ha esetleg valamilyen hiba folytán az email nem kerül elküldésre, vagy érkezik meg, akkor is van lehetőségünk újra regisztrálni. A szerver oldalon 15 percenként automatikusan lefut egy kód, ami töröl minden legalább már 15 perce regisztrált, de még nem verifikált felhasználót, hiszen ők már biztos nem lesznek hitelesítve. Ez a *node-schedule* npm csomag segítségével történik, mellyel elvégzendő job-okat tudunk ütemezni crontab megadásával.

A regisztrációt végrehajtó szerver oldali kód az [A] mellékletben található, ebben jól látható a különböző külső könyvtárak használata, illetve a backend Controller-Service-Repository rétegei is jól megfigyelhetőek rajta.

2.4 Bejelentkezés, kijelentkezés

Regisztráció és hitelesítés után az oldalra már be lehet jelentkezni, a felhasználónév és jelszó megadásával. Ilyenkor létező felhasználónév esetén lekérjük a hozzá tartozó titkosított jelszót, melyről a bcrypt meg tudja mondani, hogy megegyezik-e a felhasználó által beírttal. Ha ennek az adatok megfelelnek, akkor megtörténhet a beléptetés. Legenerálunk egy *sessionToken* nevű random UUID-t a *uuid* npm csomag segítségével, és azt elhelyezzük egy globális *session* objektumba a hozzá tartozó *userId* és lejáratí idővel együtt. A kliens oldal a visszakapott *sessionToken*-t ezután sütiben tárolja, illetve a http kérések fejlécébe rakja „Authorization” néven, mely alapján a szerver azonosítani tudja a felhasználót. Ha egy request küldésekor azt látjuk, hogy a session hamarosan lejár (1 órán belül), akkor azt megújítjuk. Egy session az alkalmazásban frissítés nélkül 24 óráig él, ezután az első kérésnél törlődik mindkét oldalon. A manuális kijelentkezés során is ez történik, először szerver oldalon töröljük a header-ben kapott session-t, majd a kliensen a sütiket és az Authorization header-t.

Jelszó elfelejtése esetén a felhasználó kérhet újat a regisztrált email címe megadásával. Ilyenkor generálódik számára egy új, 16 karakter hosszú jelszó, melyet a szerver elküld erre a címre. Egy ilyen email az alábbi 2.7-es képen látható. Ezzel már beléphet a fiókjába és megváltoztathatja a jelszavát tetszés szerint.




2.7 ábra: Elfelejtett jelszó email

2.5 Recept feltöltése

Bejelentkezett felhasználóként a menüsorban található „Upload recipe” gombbal juthatunk el a receptfeltöltő oldalra, melynek törzse a 2.8-as képen látható:

Upload recipe



Upload image

0/1000

Ingredients

Amount

Unit

Ingredient

×

+

Add ingredient

Steps

1.

Step description

×

+

Add step

Submit recipe

Additional information

Time required: :

Difficulty:

Cost:

Portions: portion(s)

Calories: kcal/portion

Primary category:

Other categories:

Diets:

Allergens:

2.8 ábra: Recept feltöltés oldal

Az „Additional information” szekcióban található adatok megadása nem kötelező, az azon kívüliek pedig igen. Kötelező adatok tehát a recept címe, egy kép az elkészült ételről, a szükséges hozzávalók, és az elkészítés lépései. A kép mérete legfeljebb 1 MB, és PNG vagy JPG kiterjesztésű lehet. Szerver oldalon a feltöltött képet az `/uploads/recipe_images` mappába mentjük a *multer* nevű npm csomag segítségével. Ez a kód az alábbi 2.9-es képen látható:

```

10 const storage = multer.diskStorage({
11   destination: (req, file, cb) => {
12     const path = './uploads/recipe_images';
13     fs.mkdirSync(path, { recursive: true })
14
15     cb(null, path);
16   },
17   filename: (req, file, cb) => {
18     const extension = file.originalname.split('.')[1];
19
20     cb(null, req.params.id + '.' + extension);
21   },
22 })
23
24 const fileFilter = function (req, file, cb) {
25   const allowedTypes = ["image/jpeg", "image/png"];
26   const fileSize = parseInt(req.headers["content-length"])
27
28   req.fileValidationErrors = [];
29
30   if(!allowedTypes.includes(file.mimetype)){
31     req.fileValidationErrors.push('Incorrect file type. ');
32   }
33
34   if(fileSize > 1024000){
35     req.fileValidationErrors.push("File can't be bigger than 1MB.");
36   }
37
38   if(req.fileValidationErrors.length > 0){
39     return cb(null, false);
40   }
41
42   cb(null, true);
43 }
44
45 const multerUpload = multer({
46   storage: storage,
47   fileFilter,
48 })
49
50 const uploadFile = function (req, res, next) {
51   const upload = multerUpload.single('image');
52
53   upload(req, res, function (err) {
54     if (err instanceof multer.MulterError) {
55       console.log("MulterError occured: ");
56       console.log(err);
57     } else if (err) {
58       console.log("Unknown error occured: ");
59       console.log(err);
60     }
61     next()
62   })
63 }

```

2.9 ábra: Recept kép feltöltése multer segítségével

A `multer.diskStorage()` segítségével megadhatjuk a mentési útvonalat, callback formájában. Ahogy a 13. sorban láthatjuk, ha az útvonalban szereplő mappák még nem léteznek, akkor létrehozuk őket. Itt határozhatjuk meg továbbá a mentett fájl nevét is. A `fileFilter` a fájlok szűrésére szolgál, itt van végrehajtva a fent említett kitételek ellenőrzése (ez küldés előtt frontend oldalon is megtörténik). Ha ezeknek a feltöltött kép nem felel meg, akkor az nem kerül mentésre. Az `uploadFile` függvény pedig lényegében csak meghívja a már konfigurált multert, és gondoskodik a hibakezelésről.

Egy hozzávalót mennyiség, mértékegység, és hozzávaló neve részekben lehet megadni, de ezek közül csak a név kitöltése kötelező. Minden recepthez legalább egy hozzávaló, és egy lépés megadása kötelező, különben a feltöltés vissza lesz utasítva.

Ezeket túl még további extra információkat is meg lehet adni a receptekhez, melyek a keresést is nagyban segíthetik. Ezek a következők: szükséges idő az elkészítéshez, nehézség, költség, adagok száma, kalóriák, elsődleges kategória, további kategóriák, diéták melyeknek megfelel a recept, és tartalmazott allergének. A nehézséget és a költséget adott fokozatok közül lehet kiválasztani, és a kategóriák is adottak. Utóbbiak kiválasztásával különböző tulajdonságokkal jelölhetjük meg a receptünket, mint például „reggeli”, „marhahús”, „csípős”, és még számos más kategória áll rendelkezésre, melyek közül többet is választhatunk. Az elsődleges kategória („Primary category”) csupán annyiban kap kiemelt szerepet, hogy a

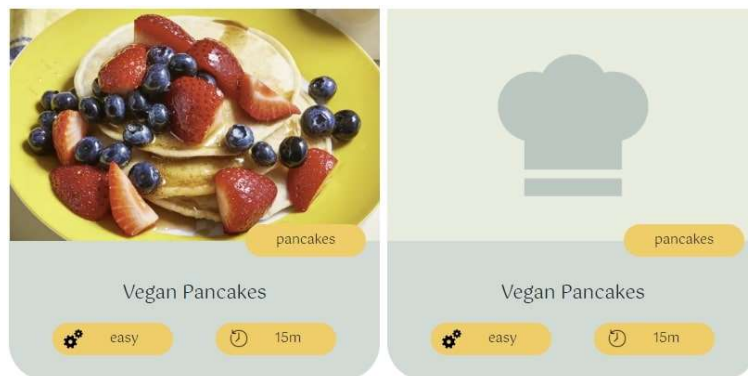
főoldalon és kereséskor a megjelenő receptkártyán az a kategória lesz látható. Ezen az oldalon, és az alkalmazás összes többi oldalán is a select beviteli mezők a `@vueform/multiselect` npm-ről letölthető külső komponens segítségével lettek létrehozva.

A „Submit recipes” gombra kattintva tölthetjük fel a receptet. Ilyenkor először kliens, majd szerver oldalon levalidáljuk az adatokat, majd azok bekerülnek a hozzájuk tartozó adatbázis táblába, külső kulcsokon át összekapcsolva, illetve a kép is feltöltődik és `<receptId>.<kiterjesztés>` néven kerül mentésre.

2.6 Recept megtekintése

2.6.1 Keresés és filterek

A főoldalon feltöltési idő szerint csökkenő sorrendben láthatók az oldalra feltöltött receptek. A receptek paginálva jelennek meg, egy oldalon 12. A pagináció megvalósításához a *Vue Awesome Paginate* külső könyvtárát alkalmazza a projekt kliens oldalon, a szerveren pedig a Prisma skip és take opció vannak használva. Egy receptkártya megjelenését a 2.10-es képen láthatjuk, bal oldalon fotóval, jobb oldalon pedig placeholder képpel, mely akkor jelenik meg, ha valamiért a képet nem sikerült elérni:



2.10 ábra: Receptkártya megjelenése

A receptek konkrétabb böngészéséhez a „Search” mezőt használhatjuk, mely a receptek nevében fog keresni, és listázza a találatokat. A talált recepteket a különböző tulajdonságaik alapján rendezhetjük, vagy a nagyobb pontosság érdekében tovább szűrhetjük. A Filters gombra kattintva megjelenik egy dropdown a szűrési lehetőségekkel, ahogy a 2.11-es képen látható:

Sort by: Uploaded ↓ Filters

Categories

Diets

Exclude allergens

Portions

portion(s)

Time

From: HH : MM

To: HH : MM

Calories

From: kcal

To: kcal

Difficulty

☐ Very easy

Cost

☐ Very cheap

Clear

2.11 ábra: Szűrési opciók lenyíló elem

A szűrést az „Apply filters” gombbal lehet megtenni. Ilyenkor a szűrési beállítások az URL-en keresztül kerülnek átadásra a keresési oldalnak amire újra ránavigálunk, így akár link alapján is meg lehet tekinteni a receptek adott listázását, és az előzményeken keresztül visszalépve is megőrzi az oldal a keresést.

2.6.2 Recept oldal vendégként

Ha találtunk egy receptet, melyet részletesebben is megtekintenénk, a receptkártyára kattintva a recept oldalon találjuk magunkat. Ez vendég felhasználóként a 2.12-es képnek megfelelően néz ki:

Churros

★★★★☆ (1 ratings)


20m

medium

very cheap

4

691 kcal/portion



Ingredients

- 1 cup(s) of water
- 2.5 tbsp white sugar
- 0.5 tsp salt
- 2 tbsp vegetable oil
- 1 cup(s) of all-purpose flour
- 2 quart(s) of oil for frying
- 0.5 cup(s) of white sugar
- 1 tsp ground cinnamon

Allergens

gluten

Steps

- Combine water, 2 1/2 tablespoons sugar, salt, and 2 tablespoons vegetable oil in a small saucepan and place over medium heat. Bring to a boil and remove from the heat. Stir in flour, stirring until mixture forms a ball.
- Heat oil for frying in a deep fryer or deep pot to 375 degrees F (190 degrees C).
- Transfer the dough to a sturdy pastry bag fitted with a medium star tip. Carefully pipe a few 5- to 6-inch strips of dough into the hot oil; work in batches so you don't crowd the fryer. Cook until golden; use a spider or slotted spoon to transfer churros to paper towels to drain.
- Combine 1/2 cup sugar and cinnamon. Roll drained churros in cinnamon and sugar mixture.

Test2

Recipes: 14

Description

Churros (Mexican fritters) are very common at fairs. In my border hometown, the line at this stand is always overwhelming. People wait hours in line just to get a taste of these churros. I have run across several recipes but this is the best one by far.

Uploaded: 30/12/2022

Last modified: 05/01/2023

Comments (1)

Average rating:

★★★★☆ 4.00

Vivi2222

07/01/2023

★★★★☆

Turned out almost perfect!

2.12 ábra: Recept oldal (oldal teteje és alja egymás mellé helyezve)

Ahogy látható, az oldal megjeleníti a recept minden megadott adatját, továbbá annak feltöltőjét és az ő receptjeinek számát, a recept feltöltésének és legutóbbi módosításának dátumát, értékelését, és a hozzá tartozó hozzászólásokat. Egy hozzászólásban láthatjuk a felhasználót, aki írta, az értékelést egy 1-től 5-ig terjedő skálán (csillagok formájában), az értékeléshez írt megjegyzést, és a küldés dátumát. A recept értékelése a felhasználók által írt értékelések átlagából jön ki.

A kép szerveroldalon a Node.js-be beépített *File System Module* segítségével töltődik be, majd *base64*-es kódolással kerül elküldésre a frontendnek. Ennek kódja a 2.13-as képen található:

```

98 module.exports.getRecipeImage = async (req, res) => {
99   try {
100     let dir = __dirname.substring(0, __dirname.lastIndexOf(path.sep));
101     let img = dir + '/uploads/recipe_images/${req.params.filename}';
102
103     fs.readFile(img, function (err, content) {
104       if (err) {
105         sendHttpException(res, new NotFound(["Recipe image not found."]));
106       } else {
107         res.writeHead(200, {"Content-type": "image/jpg"});
108         res.end(content.toString('base64'));
109       }
110     })
111   } catch (exception) {
112     if (exception instanceof HttpException){
113       sendHttpException(res, exception);
114       return;
115     }
116     sendServerErrorResponse(res, exception.message);
117   }
118 }
119 }

```

2.13 ábra: Kép elküldése a File System Module segítségével

2.6.3 Recept oldal bejelentkezett felhasználóként

Ha bejelentkezett felhasználóként tekintünk meg egy receptet, akkor elérhetővé válnak bizonyos funkciók a megtekintésen túl is.

A „Favourite” gombra kattintva hozzáadhatjuk a receptet a kedvenceinkhez, vagy ha már köztük van, akkor eltávolíthatjuk közülük. A gomb kinézetéből tudni lehet, hogy szerepel e már a recept a kedvenceinkben, hiszen olyankor a gombban lévő szív ikon kitöltött sárgává válik, és a gomb keretének is más színe lesz. Amikor éppen hozzáadásra használjuk ezt a gombot, akkor a felugró modal segítségével arra is lehetőségünk van, hogy a receptet berakjuk egy már létező csoportba, vagy új csoportot hozzunk neki létre.

A „Put on weekly menu” gomb segítségével az adott receptet a heti menüre helyezhetjük, és a megjelenő modalon kiválaszthatjuk, hogy melyik hétre, napra, és étkezésre szeretnénk berakni azt. Mivel desszerteket az alkalmazás nem napi szinten, hanem heti szinten ajánl, ezért abban az esetben, ha desszertet választunk, nem adhatunk meg napot, csak hetet, ezt azonban maga az alkalmazás sem engedi, hogy megtegyük.

A „+ Shopping list” gombbal a recept kívánt hozzávalóit írhatjuk fel a bevásárló listánkra egy gombnyomással. Ehhez először ki kell választanunk, hogy mely hozzávalókat szeretnénk felírni a hozzávalók neve melletti checkbox-ok segítségével, vagy egyszerűen az „All” checkbox bepipálásával, majd a gombra kattintva ezek a listára kerülnek. Ilyenkor a bevásárló listán létre jön egy új kategória a recept nevével, és a hozzávalók abba kerülnek.

A „Rate this recipe” gomb egy olyan modalt nyit fel, mellyel értékelést és kommentet írhatunk a recept alá. A receptet egy 1-től 5-ig terjedő skálán értékelhetjük, az alapján, hogy hány csillagot adunk neki. Az értékeléshez kommentet nem kötelező írni, azt üresen hagyhatjuk. Minden felhasználó egy recepthez csak egy értékelést írhat, azt viszont elküldés után szerkesztheti vagy törölheti az „Edit comment” és „Delete comment” gombokkal, melyek a „Rate this recipe” helyett jelennek meg miután már írtunk a recepthez értékelést. Egy felhasználó a saját receptjét nem értékelheti.

Az említett modaltok az alábbi 2.14-es ábrán láthatóak, említés szerinti sorrendben:



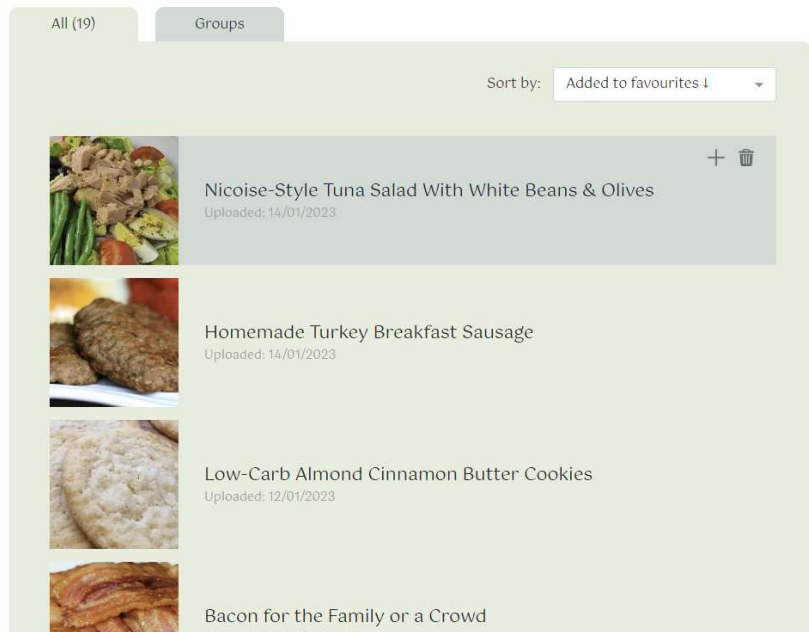
2.14 ábra: A recept oldal fontosabb modaljai

2.7 Kedvencek oldal

A menüsorban a profilképünkre kattintva további választható menüpontok jelennek meg. Ezek között a „Favourites” gombra kattintva a kedvencek oldalra navigálhatunk. Az oldal kettő tabból (fülből) áll, az egyik az „All” tab, a másik a „Groups”. Alapértelmezetten az „All” fülön vagyunk.

2.7.1 Összes kedvenc tab

Amennyiben az „All” tab van kiválasztva, akkor az összes kedvenc receptünkből álló listát láthatunk, a 2.15-ös ábrán szereplő formában:

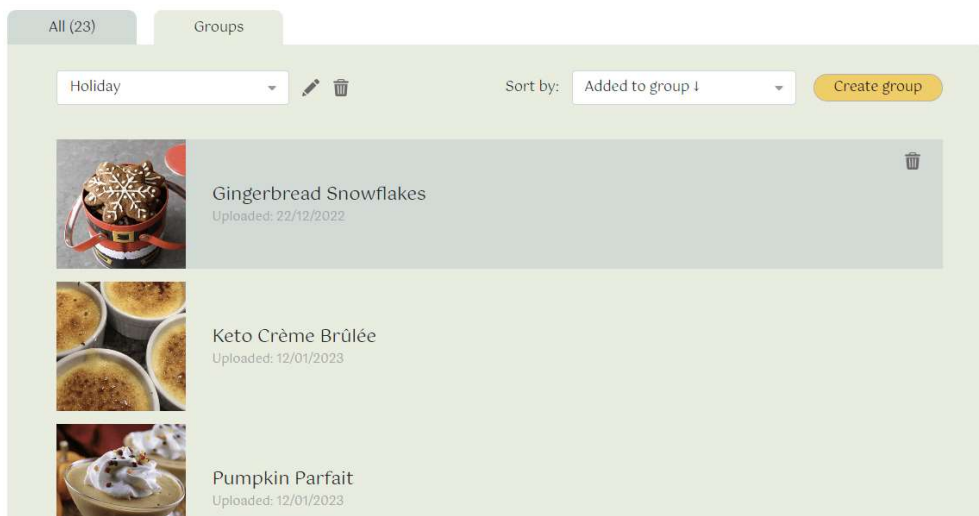


2.15 ábra: Kedvencek oldal, „All” fül

A recepteket rendezhetjük többféle módon, például a kedvencekhez adás ideje szerint növekvő vagy csökkenő sorrendben. A lista tabulálva jelenik meg, egy oldalon maximum 10 receptkártya van egyszerre. A kártyára kattintva a recept oldalára navigálhatunk, viszont, ha csak rávisszük egyre az egeret, akkor további opciók jelennek meg. A plusz gombbal egy már létező csoporthoz adhatjuk hozzá a receptet, a szemetes ikonnal pedig törölhetjük a kedvencek közül. Ilyenkor, ha a recept már egy csoport része, akkor onnan is törlődik.

2.7.2 Csoportok tab

A „Groups” fülre kattintva a csoportok tab jelenik meg. Itt a recept csoportjainkat kezelhetjük, rendszerezhetjük, és böngészhetjük. A 2.16-os ábrán láthatjuk, ez hogyan néz ki:



2.16 ábra: Kedvencek oldal, „Groups” fül

A bal felső sarokban egy select inputot találunk, itt választhatjuk ki, hogy melyik csoport receptjeit szeretnénk megjeleníteni. Amint egy csoportot kiválasztunk, a hozzá tartozó receptek azonnal megjelennek. Az input mellet lévő ceruza ikonra kattintva szerkeszthetjük a csoport nevét, a szemetes ikonnal pedig törölhetjük a csoportot. Ilyenkor a receptek a kedvencek között maradnak, csupán az aktuális csoportból törlődnek. Egy recept több csoportban is lehet, ilyenkor azokban is megmaradnak.

A jobb oldalon lévő „Create group” gombbal új receptet hozhatunk létre a nevének megadásával. Létrehozás után automatikusan az új csoport kerül kiválasztásra, ami ekkor még üres. A gomb mellett található még a receptlista rendezésére szolgáló input is. A receptek alapértelmezetten a csoporthoz adás ideje szerint csökkenő sorrendben jelennek meg.

A receptkártyák nagyon hasonlóak az összes recept fülön láthatókhöz, és ugyanúgy tabulálva jelennek meg. Az egérrel egyre rámutatva egy opció jelenik meg, a recept csoportból való törlése.

2.8 Profil oldal

A menüben a „Profile” gombra kattintva megtekinthetjük a felhasználói profilunk adatait, mint például email cím és a regisztráció dátuma, illetve, hogy hány receptet töltöttünk fel az oldalra. A felület a 2.17-es képen látható:

The screenshot shows a user profile page with a light green background. On the left, there is a circular profile picture placeholder and the username 'Vivi2222' with 'Recipes: 61' below it. To the right, there is a form with the following fields: 'First name:' with the value 'Vivien', 'Last name:' with 'Virágh', 'Email:' with 'nova@gmail.com', and 'Member since:' with '30/11/2022'. Below these fields are two buttons: 'Change password' (grey) and 'Edit profile' (orange). A horizontal line separates this section from the 'Preferences for weekly menu' section below. This section contains four settings: 'Exclude allergens:' with a minus sign, 'Diet:' with 'vegan', 'Max. recipe difficulty:' with 'medium', and 'Max. recipe cost:' with 'average'. An 'Edit preferences' button (orange) is located at the bottom right of the preferences section.

First name:	Vivien
Last name:	Virágh
Email:	nova@gmail.com
Member since:	30/11/2022

[Change password](#) [Edit profile](#)

Preferences for weekly menu

Exclude allergens: -

Diet: **vegan**

Max. recipe difficulty: **medium**

Max. recipe cost: **average**

[Edit preferences](#)

2.17 ábra: Felhasználói profil

A „Change password” gombra kattintva megnyílik egy modal, melyben megváltoztathatjuk a jelszavunkat. Ehhez természetesen a jelenlegi megadása is szükséges. Az „Edit profile” gombbal pedig a profilszerkesztő modal nyílik meg, mellyel megváltoztathatjuk felhasználónevünket, vezeté- és keresztnévünket, illetve beállíthatjuk a profilképünket. Az email címünk megváltoztatására itt nincs lehetőség, annak szerkesztéséhez egy adminnal kell felvenni a kapcsolatot.

A profil adatok alatt láthatjuk a heti menü preferenciáinkat. Ezek azok a kitételek, melyeknek a generált heti menü meg fog felelni. A kijelölt allergéneket nem fogják tartalmazni a receptek, csak a megadott diétának megfelelőek lesznek ajánlva, és nem lesznek nehezebbek vagy drágábbak a kívánnál. Ezeket a preferenciákat az „Edit preferences” gombbal elérhető modallal változtathatjuk meg. Amikor változtatásainkat elmentjük, a heti menü újra generálódik az új feltételeknek megfelelően. Az említett modalok az alábbi 2.18-as képen láthatóak:

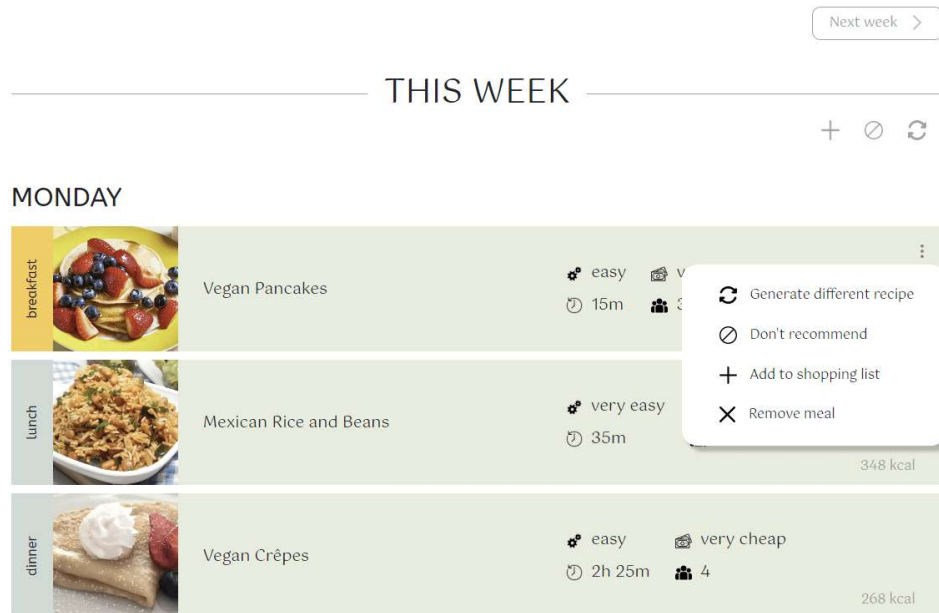
2.18 ábra: A profil oldal modaljai

A lap alsó részén a felhasználó által feltöltött recepteket láthatjuk listázva, paginált formában, különböző rendezési lehetőségekkel. Egy receptre mutatra az egérrel megjelennek a szerkesztés és törlés gombok. A szerkesztésre kattintva a receptfeltöltő oldalon találjuk magunkat, viszont a recept adatai alapján az már ki van töltve. Itt elvégezhetjük a kívánt változtatásokat, majd elmenthetjük azokat. Az oldalra navigáláskor navguard ellenőrzi, hogy a felhasználó valóban olyan receptet próbál-e szerkeszteni, mely hozzá tartozik, vagy ha nem, akkor admin-e. A törlés gombbal minden recepttel kapcsolatos dolog törlődik, beleértve a hozzá írt kommenteket és további adatokat is, és a recept képe is törlésre kerül a szerveren.

2.9 Heti menü oldal

A „Weekly menu” menüpontra kattintva egy felhasználó megtekintheti a számára generált heti menüt. Az alkalmazás egy hétre előre is ajánl recepteket, emiatt van egy eheti ajánlás oldal

(„This week”), illetve egy jövőheti („Next week”). A hét minden napjára hétfőtől vasárnapig 3-3 recept van felkínálva, egy reggeli, egy ebéd és egy vacsora. A lap alján továbbá kettő desszert ajánlás is található, melyek az egész hétre vonatkoznak. Az automatikusan generált menüben minden receptnek meg kell felelnie a felhasználó által beállított preferenciáknak. A felhasználó által manuálisan beállított receptekre ez természetesen nem vonatkozik.



2.19 ábra: Heti menü oldal

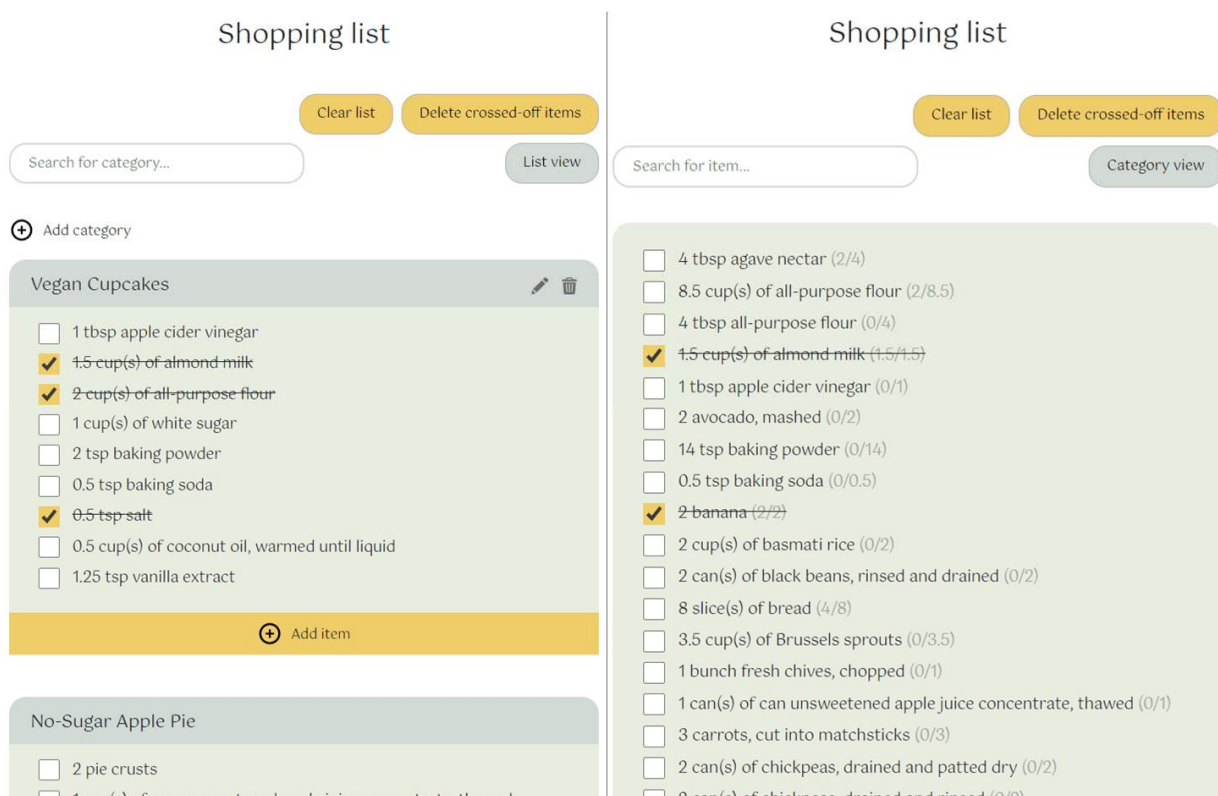
Ahogy a 2.19-es képen látható, a menü megtekintése mellett számos további funkció is rendelkezésünkre áll. A jobb felső sarokban lévő ikonok között a pluszjel ikonnal az éppen megjelenített hét összes receptjének hozzávalóit felírhatjuk a bevásárló listára. A mellette lévő áthúzott körre kattintva megjeleníthetjük azokat a recepteket, amiket korábban letiltottunk a heti menünkről. Ezek listázva jelennek meg, és ha akarjuk, visszavonhatjuk a letiltást. Receptet letiltani a „Don’t recommend” opcióval lehet, ilyenkor az alkalmazás nem fogja a receptet többé ajánlani (visszavonásig). Tovább haladva a fenti ikonok között, a körkörös nyilakra kattintva újra generáltathatjuk az egész hetet.

Az egyes étkezésekre vonatkozóan is vannak további opcióink. A „Generate different recipe” gombbal az aktuális étkezésre kérhetünk új recept ajánlatot. A „Don’t recommend” -re kattintva a már korábban említett recept letiltást lehet megtenni, az „Add to shopping list” gomb pedig csak annak az egy receptnek a hozzávalóit adja hozzá a listánkhoz. A „Remove meal” gombbal megmondhatjuk az oldalnak, hogy arra az étkezésre a héten nem szeretnénk főzni, szóval ott ne szerepeljen semmilyen recept. Ilyenkor a slot üressé válik, csupán egy „You have removed this meal for the week” felirat fogja jelezni, hogy miért nem szerepel ott recept.

Két alkalom van, amikor a rendszer automatikusan generál heti menüt a felhasználónak. Az egyik alkalom, amikor regisztráció után sikeresen hitelesíti az email címét, a másik pedig minden hétfő éjjélkor történik. Ilyenkor a jövőheti menüből az eheti lesz, és új jövőheti menü generálódik. A job ütemezése itt is szintén a *node-schedule* csomag segítségével történik. A menü generálás bár részben randomizált, az alkalmazás igyekszik elkerülni az ismétlődéseket, például egy nap ne legyen kétszer ugyanaz a recept, vagy ne az legyen, ami előző nap is volt. Ilyet csak akkor enged legenerálni, ha nincs más választása, mert mondjuk csak egy reggeli van az adatbázisban, ami megfelel minden kritériumnak, így kénytelen azt ismételni. Ha pedig nem talál olyan receptet, ami megfelelne, akkor üresen hagyja az étkezés mezőjét, és csak egy „Sorry, we have no recipe to recommend here” üzenet lesz a helyén. Az alkalmazás egy receptről onnan tudja, hogy melyik étkezéshez illik, hogy léteznek az egyes étkezéseknek megfelelő recept kategóriák (pl. „breakfast”), amit a recept feltöltője megadhat.

2.10 Bevásárló lista

A menüben a „Shopping list” gombra kattintva a felhasználó megtekintheti saját bevásárló listáját. Ez az oldal kettő nézettel rendelkezik, a kategória nézettel (Category view), és a lista nézettel (List view), melyek között a hozzájuk tartozó gombbal tudunk váltogatni. A nézetek az alábbi 2.20-as ábrán láthatóak:



2.20 ábra: Bevásárlólista oldal (bal: kategória nézet, jobb: lista nézet)

2.10.1 Kategória nézet

A kategória nézetben a bevásárló listára felírt elemek kategóriájuk szerint jelennek meg. Ha egy recept hozzávalóit írjuk fel ide annak az oldaláról, vagy a heti menü oldalról, akkor létrejön egy kategória az adott recept nevével, és abba kerülnek a hozzávalói. Saját kategóriát is létrehozhatunk az „Add category” gombbal, majd a kategória nevének megadásával. Ha egy létező kategóriára rávisszük az egeret, akkor megjelenik a kategória (névének) szerkesztése és a kategória törlése opció is. A kategóriák pagináció nélkül, egymás alatt jelennek meg, létrehozás ideje szerint csökkenő sorrendben. A kategóriák között keresni is lehet a fenti keresőmező segítségével, a szűrés azonnal történik, minden billentyűnyomásnál a bemenetnek megfelelően változik a megjelenített lista is. Egy kategóriába új elemet az „Add item” gombbal lehet felvenni, mely hatására a receptfeltöltő oldal hozzávaló megadás mezőjéhez nagyon hasonló input mezők jelennek meg. Itt ugyanúgy megadhatjuk az elem nevét, illetve opcionálisan a mennyiségét és mértékegységét is. Új elemet csak egy már létező kategórián belülre lehet felvenni. Ha egy elem mellett a hozzá tartozó checkbox-ot bepipáljuk, akkor az elem áthúzva fog megjelenni a listán, ez jelezheti például, hogy a termék már a kosarunkban van vásárlás során. Amikor egy elemet kihúzzunk, akkor az azonnal mentésre kerül az adatbázisba is. A „Delete crossed-off items” gombbal az összes ilyen áthúzott elemet törölhetjük a listáról, a „Clear list” gomb pedig teljesen kiüríti azt, minden kategória törlődik a bennük lévő elemekkel együtt.

2.10.2 Lista nézet

A lista nézet célja a könnyebb bevásárlást elősegítése. Ebben a nézetben nem jelennek meg a kategóriák, hanem helyette csak az elemek láthatóak, ábécé sorrendben listázva. Azok a termékek, melyek ugyanazzal a névvel és mértékegységgel szerepelnek, összesítésre kerülnek, egy tételként jelennek meg. Az ábécé sorrendnek köszönhetően a különböző mértékegységű, de megegyező termékek egymás alá kerülnek. Keresésre itt is van lehetőség, magukra a lista tételeire kereshetünk rá, mennyiségével, mértékegységével együtt is. Ha egy listaelemet ebben a nézetben húzzunk le, akkor minden őt alkotó elem (melyeknek összege azt adja) lehúzásra kerül az adatbázisban, és így a kategória nézetben is. Ellenkező esetben, ha a kategória nézetben húzzunk le valamit, ami a lista nézetben egy összesítés részeként szerepel, akkor az a lista nézetben nem kerül lehúzásra, viszont a tétel mögött zárójelben látható, hogy az összesített mennyiségből már mennyi került lehúzásra, pl. „8 szelet kenyér (4/8)” azt jelenti, hogy a felírt

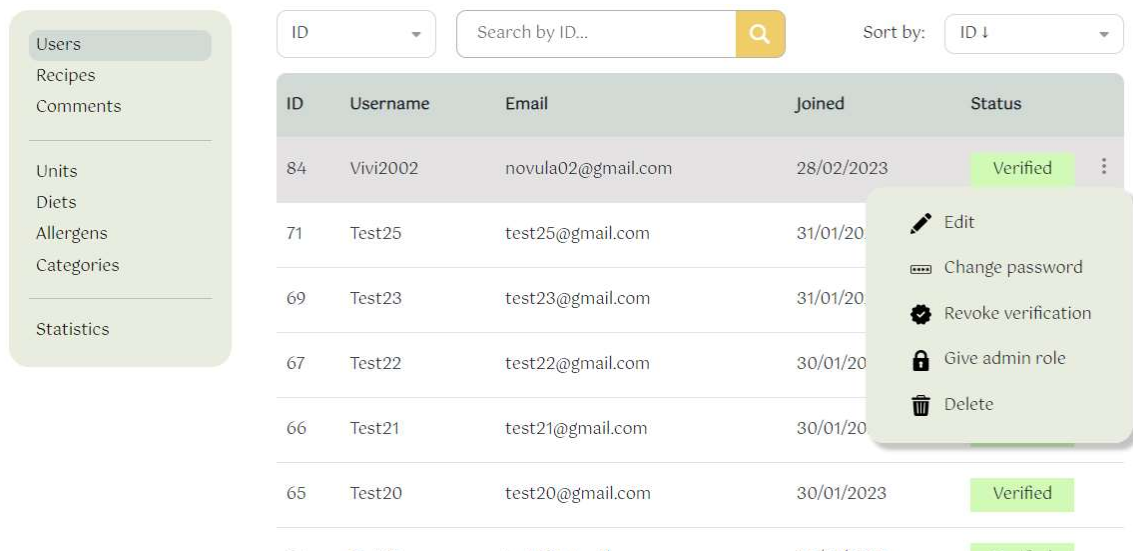
8 szelet kenyérből már 4 lehozásra került. Ha az összesített tételnek minden öt alkotó eleme lehozásra kerül, akkor már ő is áthúzza fog megjelenni.

2.11 Admin oldal

Az admin oldal a különböző adminisztrációs funkciókkal rendelkező oldalakat fogja össze. A kliensen az oldalakhoz szigorúan csakis admin joggal rendelkező felhasználók férhetnek hozzá, melyről navguard gondoskodik, és az admin műveletekhez tartozó API végpontokat is csak ők hívhatják meg, melyet middleware ellenőriz.

2.11.1 Felhasználók, receptek, és kommentek kezelése

A felhasználók, receptek, és kommentek listájának megtekintésére, azok kezelésére és moderálására egy-egy külön oldal szolgál. Az adatok táblázatos, paginált formában jelennek meg, és azok között a táblázat bizonyos oszlopai szerint lehet keresni, és az adatokat rendezni.



ID	Username	Email	Joined	Status
84	Vivi2002	novula02@gmail.com	28/02/2023	Verified
71	Test25	test25@gmail.com	31/01/20	
69	Test23	test23@gmail.com	31/01/20	
67	Test22	test22@gmail.com	30/01/20	
66	Test21	test21@gmail.com	30/01/20	
65	Test20	test20@gmail.com	30/01/2023	Verified

2.21 ábra: Admin oldal, felhasználók tábla

A 2.21-es ábrán a felhasználói adatok kezelésére szolgáló oldal látható. A másik kettő oldal is nagyon hasonlóan néz ki, csak a megjelenített mezők és az egyes rekordokkal kapcsolatban elérhető opciók különböznek.

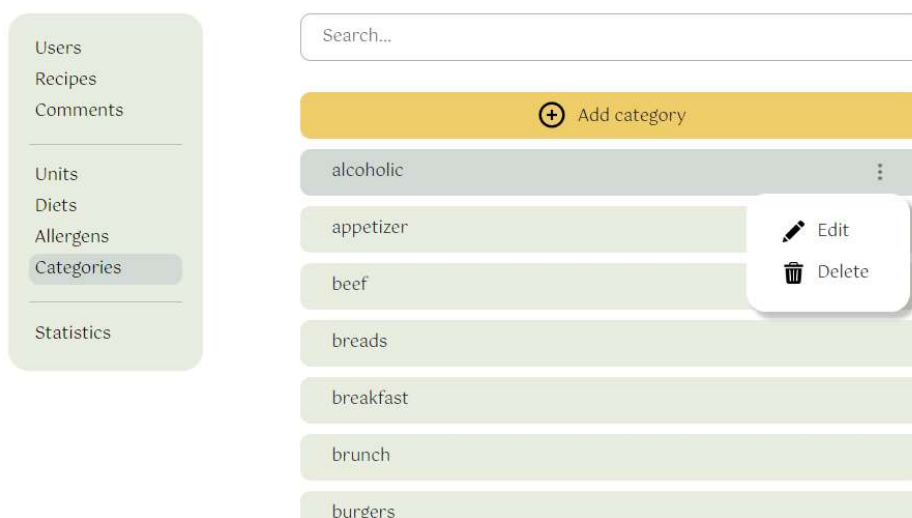
A felhasználók oldalán található a legtöbb funkció. Egy felhasználó adatait szerkeszthetjük, beleértve az email címét is, illetve a jelszavát is megváltoztathatjuk (itt természetes már nem kell az eddigi jelszót ehhez megadni), vagy akár törölhetjük is a fiókját. A felhasználó jogosultságait is megváltoztathatjuk: a hitelesítést és az admin szerepet manuálisan megadhatjuk vagy megvonhatjuk tőle. Ha egy nem hitelesített felhasználónak

admin jogot adunk, akkor automatikusan hitelesítetté is válik, ha pedig egy adminnak megvonjuk a hitelesítését, akkor azzal az admin jogát is elveszti.

A receptek és a kommentek oldalán a rekordokon végrehajtható műveletek a szerkesztés és a törlés, mely ezen adatok moderálására nyújt lehetőséget. A recept szerkesztése a már korábban tárgyalt (2.8 Profil oldal), a receptfeltöltő oldallal megegyező felületen történik. Egy admin felhasználó minden recept szerkesztéséhez hozzáfér. Mivel egy recepthoz nagyon sok adat tartozik, így nem lehet az összes fontos információt róla a táblázat egy sorában megjeleníteni, így egy „Go to recipe” opció is elérhető, mely a recept oldalra navigál.

2.11.2 Az alkalmazás belső adatainak kezelése

Az alkalmazás belső adatai alatt a mértékegységek, diéták, allergének, és recept kategóriák értendők, mivel ezek minden oldalon, minden felhasználónál megegyeznek. Az adminnak viszont hozzáférése van ezek megváltoztatására. Mindegyik típushoz tartozik egy külön oldal, ahol az adatok listázva vannak ábécé sorrendben, és lehet közöttük keresni is. Egy ilyen oldal az alábbi 2.22-es ábrán látható, de a másik három is így néz ki:

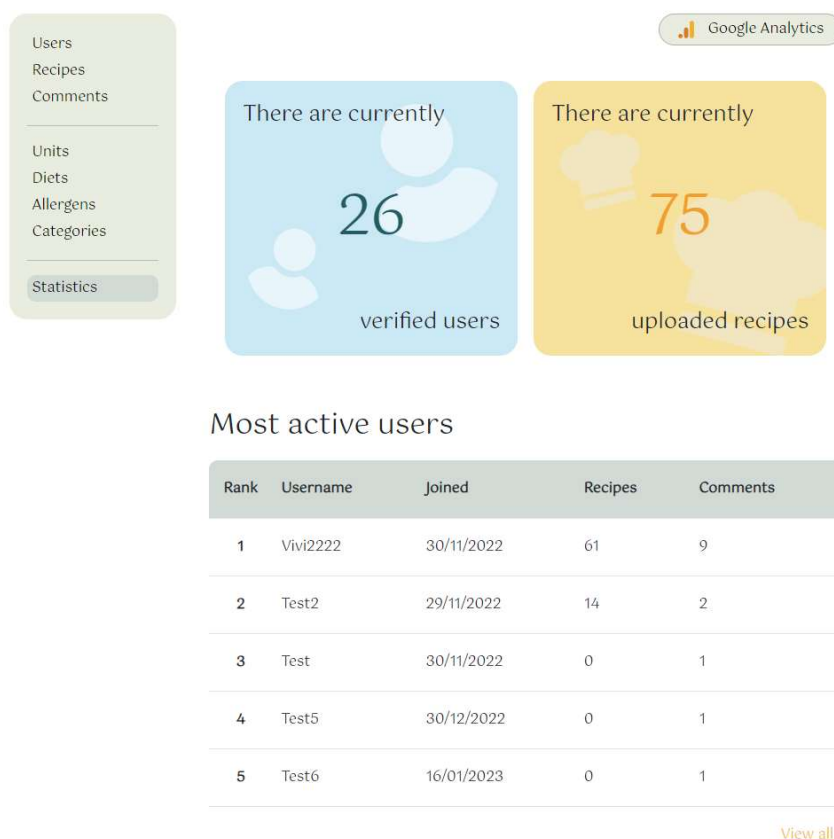


2.22 ábra: Admin oldal, recept kategóriák

Az „Add <típus>” gombbal új elemet lehet létrehozni az adott típushoz (például a fenti képen új kategóriát), az „Edit” gombbal pedig egy elem nevét lehet szerkeszteni. Két ugyanolyan nevű elem nem szerepelhet a listában. Egy adatot törölni is lehet, ebben az esetben az adott mértékegység, diéta, allergén, vagy kategória el fog tűnni a receptek leírásából és a felhasználói preferenciákból. Erről az adatbázisban beállított „onDelete: Cascade” fog gondoskodni, mely az ezen adatokra hivatkozó külső kulcsokra van beállítva.

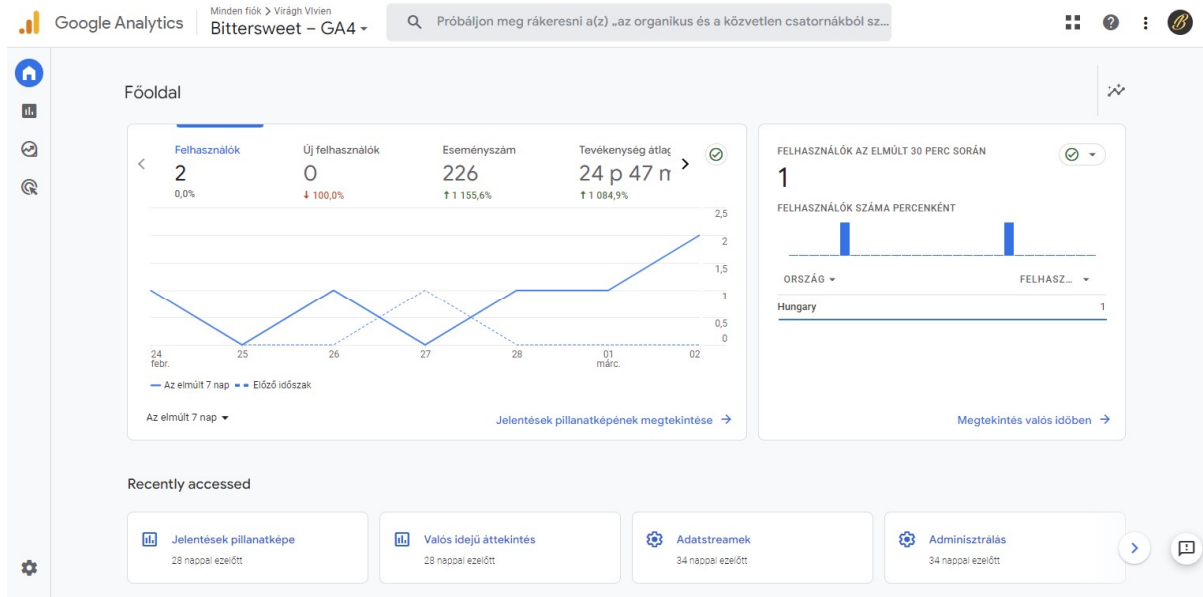
2.11.3 Statisztikák

A statisztikák oldalon az alkalmazással kapcsolatos különböző statisztikák tekinthetők meg. Látható az összes hitelesített felhasználó száma, és az oldalon elérhető összes recept száma is. Táblázatosan láthatjuk az első 5 legaktívabb felhasználót, melyek között a sorrend a beküldött receptjeik és az írt kommentjeik száma alapján dől el. Megtekinthetjük az 5 legnépszerűbb recept kategóriát, itt a rangsor a receptekben való szereplések száma szerint alakul ki. Mindezen ranglistáknak a teljes verzióját is meg lehet tekinteni a „View all”-ra kattintva, ahol már minden felhasználó és kategória helyét láthatjuk. Az oldal kialakítása az alábbi 2.23-as képen látható:



2.23 ábra: Admin oldal, statisztikák

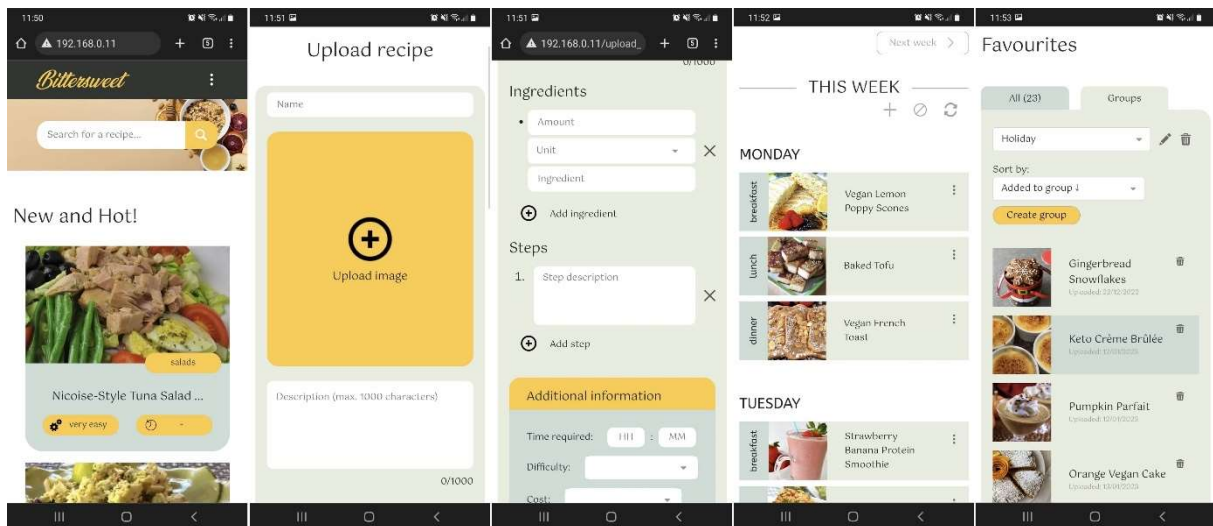
További, részletesebb statisztikák megtekintésére is van lehetőség. A weboldal számára létre lett hozva egy Google Analytics [36] oldal, amelyet a „Google Analytics” gombra kattintva és az alkalmazás email címével bejelentkezve gyorsan el lehet érni. Ennek megvalósításához csupán egy, a szolgáltatás által biztosított kódot kellett bemásolni az index.html <head> elemébe. Az oldalon nyomon lehet követni többek között az aktuális felhasználók számát, azok aktivitását, hogy melyik országban hány felhasználó van, mely oldalon mely funkciók vannak a legtöbbet használva, és még rengeteg további információ is elérhető. A főoldal a 2.24-es képen látható:



2.24 ábra: Google Analytics főoldal

2.12 Reszponzivitás

Az alkalmazás minden egyes oldala rezponzívan van kialakítva, nem csak monitoron, de táblagépeken és telefonokon is kényelmesen használható, minden funkció ugyanúgy elérhető. Az alábbi 2.25-ös képen néhány oldal található telefonos nézetben:



2.25 ábra: Reszponzív oldalak telefonon megtekintve

Ahogy látható, az oldalak elrendezése a képernyő mérete szerint változik, és érintőképernyős eszközökön a máshol egér rámutatással megjelenő funkciók alpból meg vannak jelenítve. Mindez a Bootstrap 5 Grid oszlopaival és rezponzív sablon elemeivel, illetve további media query-k segítségével lett megvalósítva. Utóbbira az alábbi 2.26-os ábrán láthatunk példákat:



2.26 ábra: Media query-k

A bal oldalon egy az oldal szélességét figyelő media query-t láthatunk, mely a kedvencek oldalra érvényes. Ha a képernyő szélessége kisebb vagy egyenlő lesz, mint 575px, akkor életbe lépnek a benne szereplő CSS szabályok. Itt például megfigyelhető, hogy bizonyos container elemek rendezése sor helyett oszlopra vált, emiatt máshova lesznek igazítva a benne lévő elemek, és azokra új margók kerülnek. A jobb oldalon lévő query pedig érintőképernyős eszközökön lesz érvényes, a példában beállítjuk, hogy mindig meg legyen jelenítve egy olyan ikon, amit eddig csak hover segítségével lehetett elérni.

3. Összefoglaló

Összeségében elmondható, hogy sikeresen létrehoztam egy működőképes, könnyen használható, letisztult felületű, és reszponzív receptmegosztó oldalt, mely rendelkezik a feladatkiírásban megadott minden funkcióval. Bár jelen állapotában is teljes értékű alkalmazást alkot, még számos lehetőség áll fenn a jövőbeli kibővítésre és továbbfejlesztésre. Ilyen lehet például egy nagy hozzávaló és mértékegység adatbázis hozzáadása a jelenlegihez, mely mértékegység átváltási információkat is tartalmaz a különböző termékekhez, illetve a heti menü ajánló algoritmust is tovább lehet fejleszteni, akár mesterséges intelligenciával is.

Irodalomjegyzék

- [1] Wikipédia, JavaScript [Utolsó hozzáférés: 2023.05.05.]: <https://hu.wikipedia.org/wiki/JavaScript>
- [2] SZTE Szkriptnyelvek kurzus előadásának fíliái (Dr. Dombi József Dániel, Dr. Antal Gábor, 2022)
- [3] Wikipédia, MySQL [Utolsó hozzáférés: 2023.05.05.]: <https://en.wikipedia.org/wiki/MySQL>
- [4] SZTE Adatbázisok kurzus előadásának anyaga (Dr. Balázs Péter Attila, Dr. Németh Gábor, 2021)
- [5] Prisma dokumentáció [Utolsó hozzáférés: 2023.05.05.]: <https://www.prisma.io/docs/concepts/overview/what-is-prisma>
- [6] Wikipédia, Docker (software) [Utolsó hozzáférés: 2023.05.05.]: [https://en.wikipedia.org/wiki/Docker_\(software\)](https://en.wikipedia.org/wiki/Docker_(software))
- [7] „Docker alapozó” (Fazekas László, Medium.com, 2021) [Utolsó hozzáférés: 2023.05.05.]: <https://thebojda.medium.com/docker-alapoz%C3%B3-b8efb6aa68e9>
- [8] Az npm hivatalos weboldala [Utolsó hozzáférés: 2023.05.05.]: <https://www.npmjs.com/>
- [9] Wikipédia, Node.js [Utolsó hozzáférés: 2023.05.05.]: <https://en.wikipedia.org/wiki/Node.js>
- [10] Wikipédia, Express.js [Utolsó hozzáférés: 2023.05.05.]: <https://en.wikipedia.org/wiki/Express.js>
- [11] „What Is Express JS In Node JS?” (Anubhav Sharma, Simplilearn, 2023) [Utolsó hozzáférés: 2023.05.05.]: <https://www.simplilearn.com/tutorials/nodejs-tutorial/what-is-express-js>
- [12] Express.js dokumentáció, Routing [Utolsó hozzáférés: 2023.05.05.]: <https://expressjs.com/en/guide/routing.html>
- [13] „Controller-Service-Repository” (Tom Collings, Medium.com, 2021) [Utolsó hozzáférés: 2023.05.05.]: <https://tom-collings.medium.com/controller-service-repository-16e29a4684e5>
- [14] Web tervezés kurzus előadás fíliái (Holló Csaba, 2021)
- [15] Web tervezés kurzus gyakorlati jegyzet (Cservenák Bence, 2021) [Utolsó hozzáférés: 2023.05.05.]: <https://okt.inf.szte.hu/webtervezes/gyakorlat/fejezet1/>
- [16] Wikipédia, HTML [Utolsó hozzáférés: 2023.05.05.]: <https://hu.wikipedia.org/wiki/HTML>
- [17] Sass dokumentáció [Utolsó hozzáférés: 2023.05.05.]: <https://sass-lang.com/documentation/>
- [18] Wikipédia, Sass (stylesheet language) [Utolsó hozzáférés: 2023.05.05.]: [https://en.wikipedia.org/wiki/Sass_\(stylesheet_language\)](https://en.wikipedia.org/wiki/Sass_(stylesheet_language))
- [19] Wikipédia, Model-view-viewmodel [Utolsó hozzáférés: 2023.05.05.]: <https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93viewmodel>
- [20] „Android UI architecture migration to MVVM” (Sebastien Eggenspieler, Dashlane.com, 2022) [Utolsó hozzáférés: 2023.05.05.]: <https://blog.dashlane.com/android-ui-architecture-mvvm/>
- [21] Vue.js dokumentáció, Introduction [Utolsó hozzáférés: 2023.05.05.]: <https://vuejs.org/guide/introduction.html>
- [22] Wikipédia, Vue.js [Utolsó hozzáférés: 2023.05.05.]: <https://en.wikipedia.org/wiki/Vue.js>
- [23] Vue Router dokumentáció [Utolsó hozzáférés: 2023.05.05.]: <https://router.vuejs.org/introduction.html>
- [24] Vite dokumentáció, „Why Vite” [Utolsó hozzáférés: 2023.05.05.]: <https://vitejs.dev/guide/why.html#why-vite>
- [25] Wikipédia, Vite (software) [Utolsó hozzáférés: 2023.05.05.]: [https://en.wikipedia.org/wiki/Vite_\(software\)](https://en.wikipedia.org/wiki/Vite_(software))
- [26] npm weboldala, axios [Utolsó hozzáférés: 2023.05.05.]: <https://www.npmjs.com/package/axios>
- [27] W3Schools Bootstrap 5 Tutorial, Grid System [Utolsó hozzáférés: 2023.05.05.]: https://www.w3schools.com/bootstrap5/bootstrap_grid_system.php
- [28] Wikipédia, Bootstrap (front-end framework) [Utolsó hozzáférés: 2023.05.05.]: [https://en.wikipedia.org/wiki/Bootstrap_\(front-end_framework\)](https://en.wikipedia.org/wiki/Bootstrap_(front-end_framework))
- [29] Pinia dokumentáció, Introduction [Utolsó hozzáférés: 2023.05.05.]: <https://pinia.vuejs.org/introduction.html>
- [30] Flaticon weboldala [Utolsó hozzáférés: 2023.05.05.]: <https://www.flaticon.com/>
- [31] Freepik weboldala [Utolsó hozzáférés: 2023.05.05.]: <https://www.freepik.com/>
- [32] Google Fonts weboldala [Utolsó hozzáférés: 2023.05.05.]: <https://fonts.google.com/>
- [33] Figma weboldala [Utolsó hozzáférés: 2023.05.05.]: <https://www.figma.com/>
- [34] Allrecipes weboldala [Utolsó hozzáférés: 2023.05.05.]: <https://www.allrecipes.com/>
- [35] „What Is the Maximum Email Address Length?” (Lifewire.com, Heinz Tschabitscher, 2020) [Utolsó hozzáférés: 2023.05.05.]: <https://www.lifewire.com/is-email-address-length-limited-1171110>
- [36] Google Analytics weboldala [Utolsó hozzáférés: 2023.05.05.]: <https://analytics.google.com/>

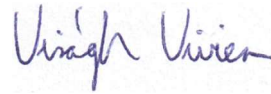
Nyilatkozat

Alulírott Virágh Vivien programtervező informatikus BSc szakos hallgató, kijelentem, hogy a dolgozatomat a Szegedi Tudományegyetem, Informatikai Intézet Műszaki Informatika Tanszékén készítettem, programtervező informatikus BSc diploma megszerzése érdekében.

Kijelentem, hogy a dolgozatot más szakon korábban nem védtem meg, saját munkám eredménye, és csak a hivatkozott forrásokat (szakirodalom, eszközök, stb.) használtam fel.

Tudomásul veszem, hogy szakdolgozatomat / diplomamunkámat a TVSZ 4. sz. mellékletében leírtak szerint kezelik.

Dátum: 2023. 05. 05.



aláírás

Köszönetnyilvánítás

Köszönöm témavezetőmnek, Dr. Mingesz Róbertnek a sok segítséget, támogatást, és javaslatokat, melyet a szakdolgozatom készítése közben nyújtott nekem. Köszönettel tartozom még továbbá a családomnak és barátaimnak, akik az alkalmazás tesztelésével, technikai javaslatokkal, és további ötletekkel segítették a dolgozat megvalósulását. Végezetül pedig köszönöm Dékány Márknak és Kiss Biankának a folyószöveg lektorálását.

Mellékletek

A.

Router:

```
router.post('/register', userController.register);
```

Controller:

```
module.exports.register = async (req, res) => {  
  try {  
    res.json( await userService.register(req.body));  
  } catch (exception) {  
    if (exception instanceof HttpException){  
      sendHttpException(res, exception);  
      return;  
    }  
    sendServerErrorResponse(res, exception.message);  
  }  
}
```

Service:

```
module.exports.register = async (userData) => {  
  const errors = [];  
  
  if(!userData.username?.trim() || !userData.email?.trim() ||  
    !userData.password?.trim()){  
    errors.push("Please fill in all fields");  
  }  
  
  if(userData.username?.trim().length > 100) {  
    errors.push("Username can't be longer than 100 characters");  
  }  
  
  if(userData.email?.trim().length > 256) {  
    errors.push("Email can't be longer than 256 characters");  
  }  
  
  if(userData.email?.trim() &&  
    !userData.email?.toLowerCase().match(EMAIL_REGEX)){  
    errors.push("Invalid email");  
  }  
  
  if(userData.password?.trim() !== "" && userData.password?.trim().length < 6){  
    errors.push("Password must be at least 6 characters long")  
  }  
  
  if(errors.length > 0){  
    throw new BadRequest(errors);  
  }  
}
```

```

try {
  let user = await userRepository.createUser(userData);

  jwt.sign(
    {
      user: user.id,
    },
    EMAIL_SECRET,
    {
      expiresIn: 15 * 60, //15 minutes
    },
    (err, emailToken) => {
      const url = `${process.env.CLIENT_REQUEST_URL}/verification/${emailToken}`;

      transporter.sendMail({
        to: user.email,
        subject: 'Bittersweet - Email Verification',
        html: `
          <div style="...">
        `
      });
    },
  );

  return user.id;
} catch (exception) {
  throw exception
}
}

```

Repository:

```

module.exports.createUser = async (userData) => {
  const salt = await bcrypt.genSalt(10);
  userData.password = await bcrypt.hash(userData.password, salt);

  try {
    return await prisma.User.create({
      data: {
        ...userData
      }
    });
  } catch (exception) {
    console.log(exception);
    switch (exception.meta.target){
      case "User_username_key": throw new BadRequest(["The username is already taken."]);
      case "User_email_key": throw new BadRequest(["The email is already in use."]);
    }
    throw new InternalServerError("Something went wrong during signup.");
  } finally {
    await prisma.$disconnect();
  }
}
}

```