

Győri Szakképzési Centrum Jedlik Ányos Gépipari és
Informatikai Technikum és Kollégium



Ready!

A büfé alkalmazás

Kékesi Ádám – Sulyok Dávid – Fekete Miklós

Győr 2023

Tartalom

Bevezetés	4
A probléma	4
Az ötlet	4
Publikáció	4
Alkalmazás használati útmutató	6
Kezdőlap	6
Regisztráció	6
Vásárlás menete	7
A program	9
Béta verzió	9
Csapatmunka	9
Feladatok elosztása	9
Projektszervezési, tervezési eszközök	11
Meg nem valósított funkciók	13
Github	14
Repository	14
Branchek, workflow	14
Feature branchek elnevezési szabályai	16
Commit üzenetek	17
Használt technológiák	17
Adatbázis	19
Adatbázis felépítése	20
Táblák	20
Backend	23
REST API	23
Autentikáció	25
Email küldés	25
Autorizáció	26
Adatfeldolgozás	28
Validáció	29
Frontend	30
Autentikáció	31
Regisztráció	32
Bejelentkezés	33
Email megerősítés és jelszó visszaállítás	34

Vásárlói oldalak.....	34
Büféválasztó oldal.....	34
Főoldal.....	35
Termékek egyedi oldala	37
Desktop felhasználói kinézet.....	38
Admin oldalak.....	38
Főoldal.....	38
Büfé oldal.....	39
Büfé szerkesztő oldal	40
Új büfé oldal	40
Kategóriák oldal.....	41
Termékek oldal	41
Tesztelés.....	42
Backend	42
E2E	44
Források:	45

Bevezetés

A probléma

Biztosan előfordult már önökkel, hogy egy büfében szerettek volna vásárolni de hosszú volt a sor. Iskolában ez nem csak kellemetlenséget okoz, hanem még a vásárlást is korlátozhatja, hiszen nem biztos, hogy a vásárlók sorra kerülnek a szünetben.

Ready! nevű alkalmazásunk erre ad megoldást, alkalmazásunkkal a vásárlók előre megrendelhetik a kívánt ételüket, ezzel megkönnyítik a dolgozók munkáját, gyorsítják a vásárlást.

Az ötlet

Az alkalmazás alapötlete az idei tanév előtt már fél évvel megszületett, a csapat egyik tagja részt vett egy országos start up és ötlet versenyen az Ideafesten.

A versenyre egy prezentációt kellett készíteni, amelyben többek között az alkalmazás felépítése és megvalósítási ütemterve is szerepelt. Az országos fordulóra el kellett készíteni az applikáció látványtervét a figma nevezetű design tervező webalkalmazásban.

A verseny során több konzultáción is részt vett csapatunk tagja, köztük a WOLT egyik munkatársaival is, amin keresztül sikerült az ételrendelő alkalmazás mögé látni, megtudni, hogyan is épül fel és mire lehet szükség a megvalósításához. Továbbá egy magyar start up projektben résztvevő UI és UX designerrel is sikerült felvenni a kapcsolatot, aki segített kialakítani az alap designt.

A regionális döntőn sikeres első helyet ért el a Ready! alkalmazás, az országos fordulóban pedig különdíjat is szerzett. Az ötlet, az arculat és a design már készen állt a megvalósításra, ezért választottuk ezt a szakdolgozatunknak.

Publikáció

Backend

<http://ec2-18-192-127-39.eu-central-1.compute.amazonaws.com/swagger>

Frontend

<https://master.d1hs3z07c3o1bl.amplifyapp.com/>


Github

<https://github.com/14A-D-Ready-team>

Alkalmazás használati útmutató

Kezdőlap

Ha a felhasználó nincs bejelentkezve akkor a bejelentkezés oldal jelenik meg, ha be van akkor pedig a büfé választó jelenik meg.



Ready!

Bejelentkezés

Email

Jelszó


[Elfelejtette jelszavát?](#)

Bejelentkezés

Nincs fiókja? [Regisztráció](#)

Vagy

[Bejelentkezés Google fiókkal](#)



Ready!

Büfé kiválasztása

Jedlik büfé

☆☆☆☆ 0.0

Büfé a Győri Szakképzési Centrum Jedlik Ányos
Gépipari és Informatikai iskolában. Széles választék,
gyors kiszolgálás.

Hétfő: 08:00 – 13:00 Kedd: 08:00 – 13:00 Szerda: 08:00
– 13:00 Csütörtök: 08:00 – 13:00 Péntek: 08:00 – 13:00
Szombat: Zárva Vasárnap: Zárva

Kiválasztás

[Vissza](#)

Regisztráció


Alkalmazásunk használatához regisztrálni kell. Regisztrációval négy fajta fiókot lehet létrehozni:

- Vásárló
- Büfé tulajdonos
- Büfé dolgozó
- Ready! admin

Büfé tulajdonos, dolgozó és Ready! admin fiókot az admin regisztrációs oldalon hozhatunk létre.

Vásárlói regisztrációnál van lehetőség Google fiókkal regisztrálni. Fiók létrehozás után a felhasználónak meg kell erősítenie az email címét, amit a regisztrálásnál megadott email címre küldött megerősítő emaillel tehet meg.

Vásárlás menete




Ready!

Admin Regisztráció

Felhasználó típusa: Büfé dolgozó ▾

Regisztráció

Van már fiókja? [Bejelentkezés](#)



Ready!

Regisztráció

Regisztráció

Van már fiókja? [Bejelentkezés](#)

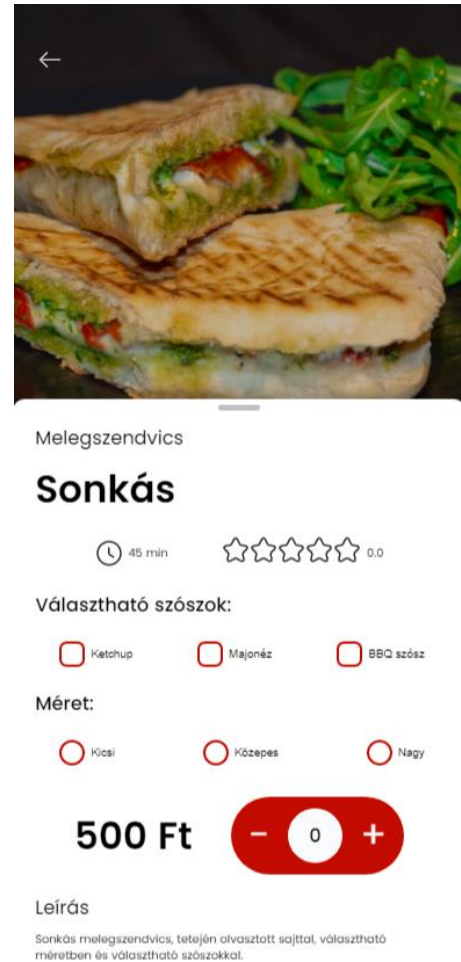
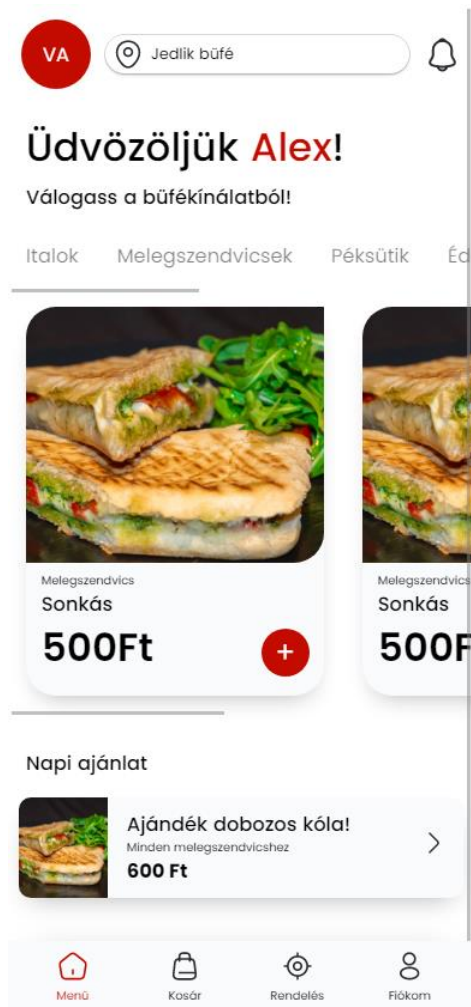
Vagy

Fiók aktiválás után az alkalmazás használatához be kell jelentkezni.

Bejelentkezés után a felhasználónak ki kell választania, hogy melyik büfében szeretne vásárolni, kiválasztás után az alkalmazás főoldalára jutunk. Itt a büfé kínálatát találjuk kategóriákra bontva. A kategóriák a termék fajtáját jelentik, például ital vagy pékáru. A főoldalon található még a büfé napi ajánlata is, ami akciós termékeket vagy menüket tartalmazhat.

Ha a felhasználó rákattint egy termékre akkor átkerül a termék oldalára ahol megtalálja a termék adatait, berakhatja azt a kosarába.

Tervezünk egy kosarat is, amiben majd a kosárba rakott termékeket lehet



átnézni és kifizetni.

A program

Béta verzió

2023.03.10. napjára terveztük ennek a változatnak a kiadását hiszen egyik csapattagunk számára ez volt a leadási határidő. Ez után folytattuk a munkát, implementáltuk a kimaradt funkciókat, kijavítottuk a felmerült hibákat.

Csapatmunka

A projektet kisebb részekre bontottunk amiket sorban oldottunk meg. Fejlesztés közben mindenkinek megvolt a feladata, ha elakadtunk akkor segítettünk egymásnak, így könnyen tudtunk haladni a saját munkánkkal.

Feladatok elosztása

Kékesi Ádám:

- frontend:
 - kategóriakezelő oldal az adminisztrációs felületen
 - termékekezelő oldal az adminisztrációs felületen
 - új termék létrehozó oldal az adminisztrációs felületen
 - admin menü, shell
 - útvonalak védeése, átirányítások
 - bejelentkezés sessionnel
 - autorizáció
 - bejelentkezés Google-lel
 - szerverüzenetek feldolgozása, hibakezelés
- backend:
 - CRUD műveletek a kategóriák kezelésére
 - kategóriák felépítése az adatbázisban

- CRUD műveletek a termékek kezelésére
- termékek kezeléséhez tartozó adatbázistáblák
- session management
- autentikáció Google-lel
- autorizáció
- egyéb:
 - ci/cd CircleCI-al
 - app hostolása AWS-en

Sulyok Dávid:

- frontend:
 - regisztrációs oldalak
 - bejelentkező oldal
 - email megerősítést küldő oldal
 - jelszó visszaállító oldalak
 - vásárlói főoldal
 - termékek egyedi oldala
 - büfé választó oldal
- egyéb:
 - figma design és látványterv
 - egyedi UI és UX design megvalósítása
 - megerősítő és jelszó visszaállító email kinézete

Fekete Miklós:

- frontend:
 - admin regisztrációs felület
 - admin főoldal
 - büfé adminisztrációs oldalak
- backend:
 - felhasználó autentikáció
 - regisztrációs, jelszó-émlekeztető emailek küldése
 - CRUD műveletek a büfék kezelésére
 - Entity-k létrehozása
- egyéb:
 - adatbázis megtervezése
 - ER-modell készítése

Projektszervezési, tervezési eszközök

1. Személyes megbeszélések:

Napi rendszerességgel tartottunk megbeszéléseket hiszen ugyan abba a nappali tagozatos osztályba járunk. Ezek során megbeszéltük ki, hogy halad a munkájával, segítettünk egymásnak ha valaki elakadt és megbeszéltük a terveinket hátralevő munkához.

2. Discord:

Online kommunikációnk fő eszköze. Létrehoztunk egy szerveret ahol tudunk egymással szóban és írásban is kommunikálni. Hétvégén itt tartottuk a megbeszéléseinket.

3. Facebook Messenger:

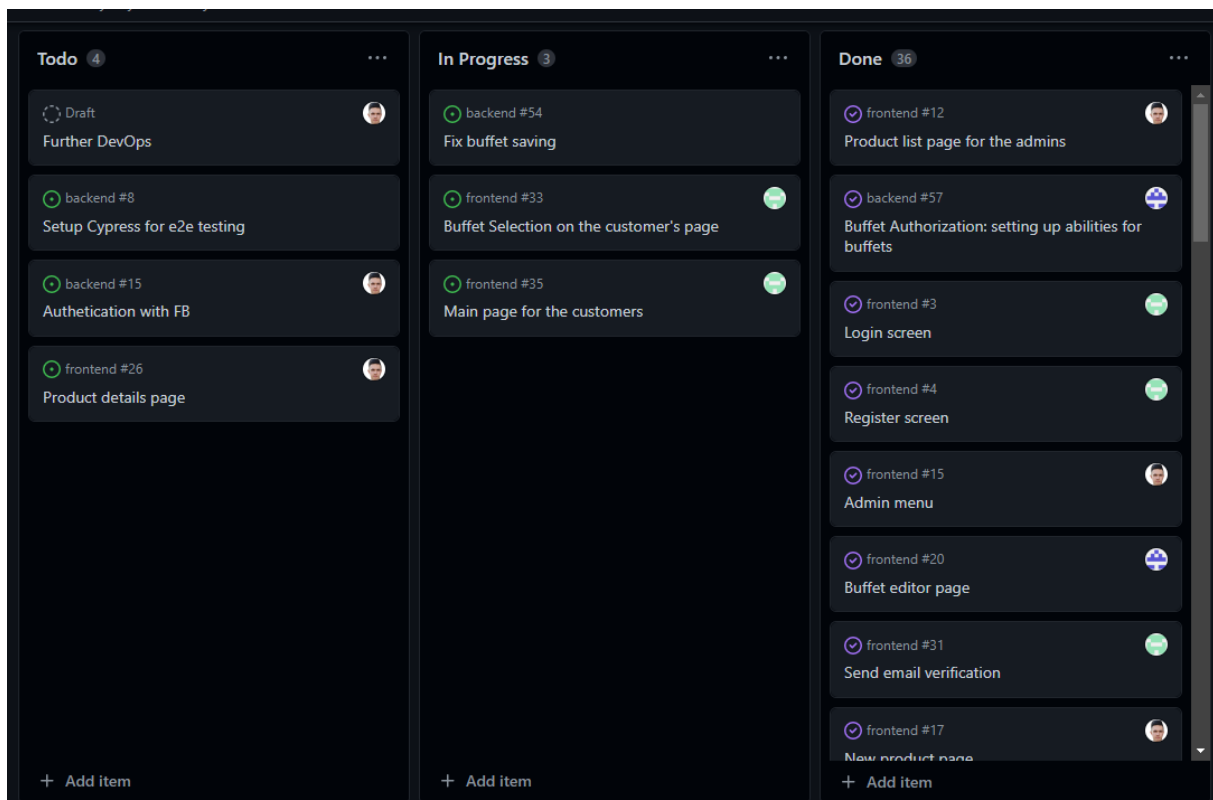
Másodlagos kommunikációs eszköz, általában gyors kérések és kérdések küldéséhez használtuk.

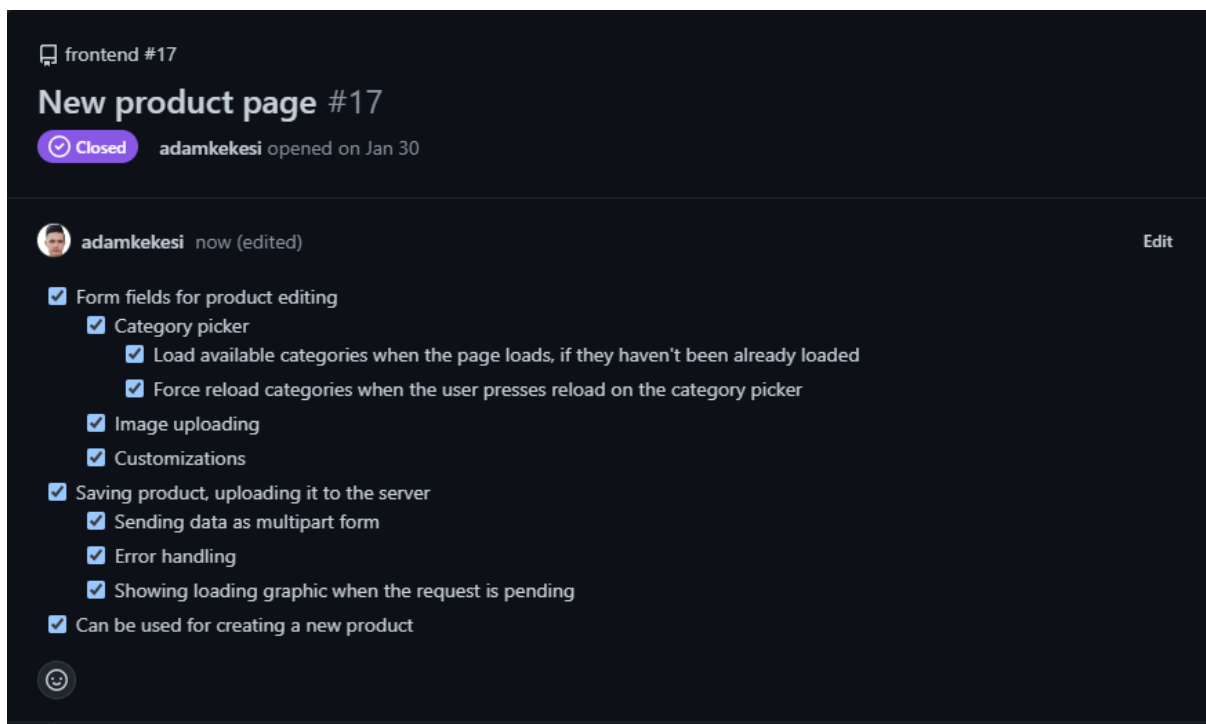
4. Github Kanban Board:

Ezt a projektmenedzselő eszközt használtuk a hátralévő munkák nyomon követésére. Feladatokat (issue) rögzítettünk benne, amelyeket felosztottunk egymás között. Az issue-kat a Github repositorykkal is össze lehetett kötni, azon belül Pull Requesteket és brancheket is hozzá lehetett rendelni.

Három részre osztottuk a táblánkat:

- Todo: jövőbeli feladataink
- In Progress: azok az issuek, amiken éppen dolgozunk
 - Maximum 2-3/személy, a Kanban irányelvek alapján
- Done: elkészült munkák





5. Github

Külön fejezet van neki

Meg nem valósított funkciók

Főoldalunkon még nem az elvárt módon jelennek meg a termékek. A végső változatban kategóriánként szeretnénk megjeleníteni a termékeket.

A termékek adatai még nem jelennek meg ha a felhasználó rákattint, így még vásárolni sem lehet.

Bár az admin felület már majdnem kész van, még hiányzik belőle, hogy a büfé munkás előben lássa a bejövő rendeléseket.

A jövőben tervezünk egy értékelés rendszert is bevezetni.

Github

Repository

Az alkalmazás forráskódját a Githubon tároltuk, a Git nevű verziókezelő rendszert használva. A Githubon a projekteinket úgynevezett repositorykban tárolhatjuk. Arra, hogy az alkalmazásunk forráskódját hogyan helyezzük el repositorykban, több módszer is létezik:

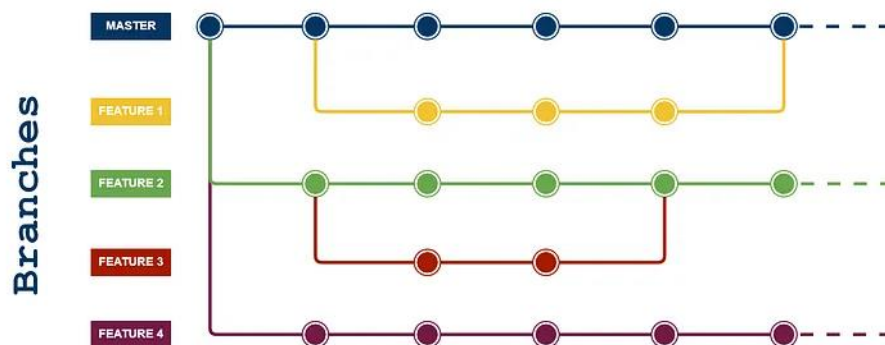
- Monorepo: az alkalmazás teljes forráskódja egy repositoryban helyezkedik el. Monorepo-kezelő szoftver (pl. Nx Workspace) használata nélkül nehéz kihasználni az előnyeit.
- Multirepo: az alkalmazás projektjei külön repositorykban találhatóak

Mi a multirepo megoldást választottuk, a frontendnek és a backendnek is külön repositoryt készítettünk. Mivel monorepo-kezelő szoftvert nem használtunk még, kitanulni egy ilyen technológiát időt vett volna el a fejlesztéstől. Ezért maradtunk a multireponál.

Branchek, workflow

Git workflownak a feature-branch workflowt választottuk. Ez a megoldás kétfajta branchet különböztet meg:

- master: Ez a fő branch, ebben működőképes kód található
- feature branchek: Az éppen készülő alkalmazásfunkcióknak külön brancheket indítottunk. Ezekben a branchekben félkész, hibás kód is található, amíg a fejlesztésük tart.



A feature brancheinket hozzákötöttük a Github Kanban-tábla issuejaihoz, amelyekben le volt dokumentálva az adott funkció leírása, követelményei.

Úgy gondoltuk, hogy ez a megoldás nem vesz el túl sok időt fejlesztéstől, viszont ad egy struktúrát a projektnek verziókezelési szempontból.

Ha végeztünk egy funkció lefejlesztésével, az adott feature-branch integrálását kértük a master branchbe egy Pull Requesttel. A Pull Requestet legalább egy csapattagnak át kellett néznie, mielőtt merge-eléssel beletettük volna az új funkciót a master branchbe.

Azért, hogy a master branch commit logját tisztán tartsuk, a PR-jainkat egy commitként olvasztottuk bele a masterbe. Így akárhány commit lehetett egy feature branchen, a master commit logjában csak a feature branch neve látszik. Ezt a mergelési módot Squash Merge-nek hívják.



A brancheket töröltük, miután integráltuk a masterba.

Feature branchek elnevezési szabályai

Már a fejlesztés elején felmerült, hogy szükség lenne a branchek elnevezéseit szabályokhoz kötni. Hosszas böngészés után találtunk olyan szabályrendszert, amely alapján mi is ki tudtuk alakítani a sajátunkat. Az AngularJs Commit Message Conventions és a NestJs open-source framework branch elnevezési szabályait vettük alapul.

Egy branch elnevezése így néz ki:

<típus>(<modul1>, <modul2>...)/<leírás>

6 branchtípust különböztettünk meg:

- ci: CircleCI beállításai
- feat: Új feature
- fix: Bugfixek
- refactor: Kód átalakítása, átnevezések, stb
- test: Tesztek írása

A branch neve azt is tartalmazza, hogy az alkalmazás melyik részére vannak kihatással a branch változtatásai (modul1, modul2). Példák a modulokra:

- admin: admin felület
- product: termékek

Végül pedig egy rövid, tömör leírást adunk a branchnek, ami utal arra, hogy a branchben mit változtatunk a kódon.

A Pull Request neve megegyezik a branch nevével, azzal a különbséggel, hogy “-” helyett szóközt használunk a leírásban, “/” helyett pedig “:”-ot.

Példa egy Pull Request nevére:

Feat(product): New product page

Ezen a branchen egy új funkciót fejlesztünk le, egy új termék létrehozására alkalmas oldalt. Ez a branch az alkalmazásnak azt a részét érinti, amely a termékekkel foglalkozik.

Commit üzenetek

A commit üzenetek elnevezésére különösebb szabályaink nem voltak. Erre kivétel a master branch, amelyre közvetlenül bugfixeket, kisebb változtatásokat töltöttünk csak fel. Ezekre a commitokra ugyan azok az elnevezési szabályok vonatkoztak, mint a branchekre.

Használt technológiák

Figma

Az alkalmazás látványterve Figmában készült. Később ez alapján alakítottuk ki az alkalmazás dizájnját.

Visual Studio Code

Csapatunk tagjai által jól ismert szerkesztőprogram. Az alkalmazás teljes forráskódja Visual Studio Code-ban lett elkészítve.

MySQL

Adatbáziskezelőnek a MySQL-t választottuk, hiszen ezzel a szoftverrel már volt tapasztalatunk.

XAMPP

Segítségével tudtunk lokálisan adatbázist kezelni, egyszerűsége miatt később könnyet át tudtunk váltani az Amazon Web Services szolgáltatásra.

AWS

Alkalmazásunkat az Amazon Web Services felhőszolgáltatás hostolja.

MikroORM

A MikroORM egy objektum-relációs leképező, amelyet az adatbázisunk és a backend közötti kommunikációra használtunk.

Typescript

Az alkalmazásunk frontendje és backendje is ezen a nyelven íródott. JavaScriptre fordul.

Nodejs

Backendünk futtatókörnyezete, amely képes JavaScript kódot futtatni.

NestJs

Backend keretrendszer, amelynek a felépítése hasonló az Angularhoz. Nodejs futtatókörnyezetet használ, és az Express frameworkre épít.

Ionic

Frontend keretrendszer, amely sok hasznos felhasználói felület elemet és funkciót tartalmaz, amely megkönnyíti a telefonos webalkalmazás fejlesztését.

SCSS

CSS-t kiegészítő stílusnyelv. Az alap CSS-t egészíti ki különböző hasznos funkciókkal, ezzel egyszerűbbé téve a kinézet testreszabását.

Angular

Komponens alapú, Typescript programozási nyelvre épülő frontend keretrendszer.

Adatbázis

Adatbázisunk elkészítése nehéz feladat lett volna. Projektünk mérete miatt sok táblát kellett létrehoznunk, ezért használtuk a MikroORM nevű objektum relációs leképezőt.

A MikroORM megkönnyítette a munkánkat, használatával kódban tudtuk létrehozni az adatbázist, így egyszerűen le tudtuk generálni az adatbázist.

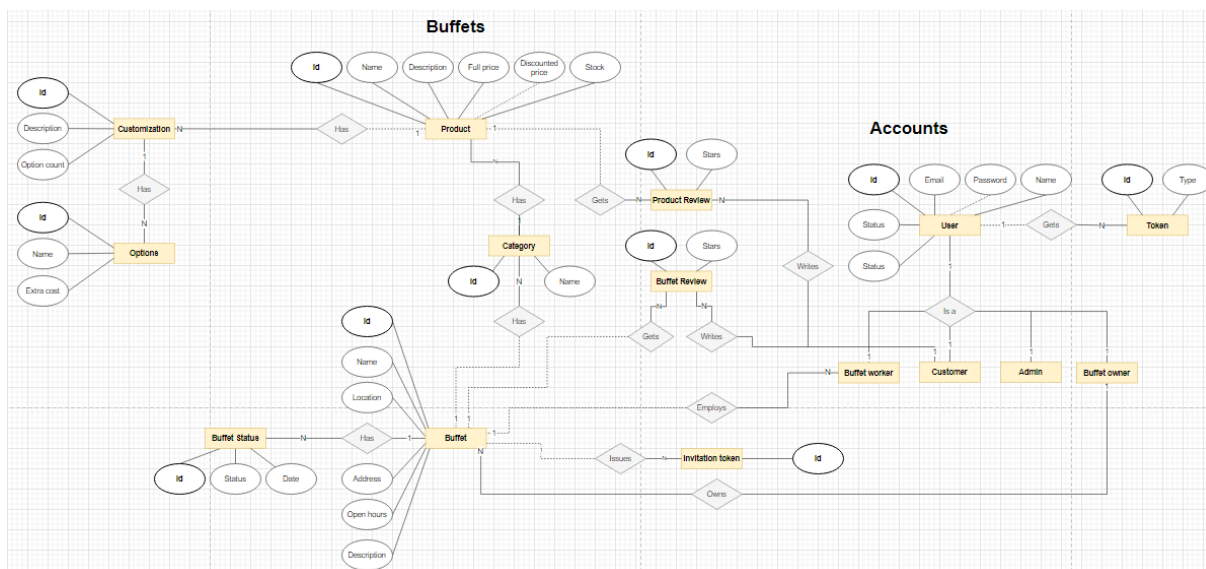
```
@Entity()
export class Category {
  @PrimaryKey({ autoincrement: true })
  @Expose({ toPlainOnly: true })
  public id: number;

  @Property({ length: 80 })
  @Expose()
  public name: string;

  @Exclude()
  @OneToMany(() => Product, product => product.category, {
    cascade: [Cascade.PERSIST],
  })
  public products = new Collection<Product>(this);

  @ManyToOne({
    cascade: [Cascade.PERSIST, Cascade.MERGE, Cascade.CANCEL_ORPHAN_REMOVAL],
  })
  public buffet?: IdentifiedReference<Buffet>;
}
```

Adatbázis felépítése



Táblák

user

- id - elsődleges azonosító
- type - felhasználó típusa
- name - felhasználó neve
- email - felhasználó email címe
- password - felhasználó jelszava argon2-vel titkosítva
- status - felhasználó aktivitásának státusza, fiók megerősítéssel lehet aktív

token

- id - elsődleges azonosító
- type - a token típusa
- user_id - a birtokló felhasználó azonosítója

buffet_worker

- user_id - elsődleges azonosító és egyben idegen kulcs is
- buffet_id - munkahelyének azonosítója

customer

- user_id - elsődleges azonosító és egyben idegen kulcs is

admin

- user_id - elsődleges azonosító és egyben idegen kulcs is

buffet_owner

- user_id - elsődleges azonosító és egyben idegen kulcs is

buffet

- id - elsődleges azonosító
- name - büfé neve
- coords - büfé koordinátái (később Google Maps-es büfé kereséshez)
- address - büfé címe
- hours - nyitvatartás
- description - leírás
- image - kép a büféről mediumblob-ban tárolva
- image_type - kép típusa
- buffet_owner_user_id - idegen kulcs, büfé tulajdonos azonosítója

buffet_invite_token

- id - elsődleges azonosító
- buffet_id - idegen kulcs, birtokló büfé azonosítója

buffet_status

- id - elsődleges azonosító
- status - büfé státusza, aktív vagy inaktív
- date - státusz változtatás ideje
- buffet_id - idegen kulcs, büfé azonosítója

buffet_review

- id - elsődleges azonosító
- stars - az értékelés, csillagban megadva, minimum 1.0 maximum 5.0, lehet tört
- buffet_id - idegen kulcs, az értékelést birtokló büfé azonosítója

product

- id - elsődleges azonosító
- name - a termék neve
- description - termék leírása
- full_price - a termék teljes ára
- discounted_price - a termék akciós ára
- image - kép a termékről mediumblob-ban tárolva
- image_type - kép típusa
- category_id - idegen kulcs, a termékhez tartozó kategória azonosítója

category

- id - elsődleges azonosító
- name - kategória neve
- buffet_id - idegen kulcs, a birtokló büfé azonosítója

customization

- id - elsődleges azonosító
- description - leírás a testreszabási lehetőségről
- option_count - lehetséges opciók száma
- product_id - idegen kulcs, a testreszabást birtokló termék azonosítója

option

- id - elsődleges azonosító
- name - az opció neve
- extra_cost - az opcióval járó extra költség
- customization_id - idegen kulcs, az opciót birtokló testreszabás azonosítója

Backend

Github url: <https://github.com/14A-D-Ready-team/backend>

Az alkalmazásunk által kezelt adatokhoz a hozzáférést a backend projektünk kezeli.

REST API

A backend projektünk HTTP-n kommunikál a klienssel, és a REST szabályait követi.

Az API-ról dokumentáció is elérhető a /swagger útvonalon.

Az erőforrásokat a /<erőforrás> útvonalon lehet elérni.

Végpontok egy átlagos CRUD controllernél:

- GET / : összes entitás lekérése, szűrési feltételek alapján, amelyeket query paraméterekben lehet megadni.
- GET /<id>: lekérés id alapján
- POST /: új létrehozása
- PATCH /: részleges frissítés
- DELETE /: törlés

Adatbáziskapcsolat

Ahogy fentebb már említettük, backend projektünkhöz a MikroORM-ot használtuk. Ez a szoftver Typescript osztályokból képes adatbázist leképezni, létrehozni. Az adatbázismezők, táblák beállításait Typescript decoratorokkal tudjuk megadni.

Adatbázislekéréseket is lehet készíteni a MikroORM-mal anélkül, hogy SQL kódot kellene írunk.

Például elsődleges kulcs alapján így kérhetünk le vele entitást:

```
public async findOne(id: number) {  
  return this.categoryRepository.findOne(id);  
}
```

Új rekordot így tölthetünk fel vele az adatbázisba:

```
const category = new Category({  
  ...rest,  
  buffet: Reference.create(buffet),  
});  
await this.categoryRepository.persistAndFlush(category);
```

Így frissíthetünk egy recordot:

```
categoryToUpdate = this.categoryRepository.assign(categoryToUpdate, {  
  ...rest,  
  buffet: newBuffet ? buffetId : categoryToUpdate.buffet?.id,  
});  
await this.categoryRepository.persistAndFlush(categoryToUpdate);
```

Törlést így valósíthatunk meg vele:

```
public async remove(id: number) {  
  const entity = await this.findOne(id);  
  if (entity) {  
    await this.categoryRepository.removeAndFlush(entity);  
  }  
}
```

Az adatbázishoz való kapcsolódáshoz a backendnek tudnia kell az adatbázis jelszavát is. Ilyen érzékeny adatokat nem írhatunk bele a forráskódba, ezért a kapcsolódási adatokat környezeti változókból szerzi meg a backend. Ezek az adatok helyi adatbázist használva így nézhetnek ki:


```
MIKRO_ORM_TYPE=mysql  
MIKRO_ORM_DB_NAME=ready-db-dev  
MIKRO_ORM_HOST=localhost  
MIKRO_ORM_PORT=3306  
MIKRO_ORM_USER=admin  
MIKRO_ORM_PASSWORD=
```

Autentikáció

Az első funkció amit elkészítettünk backenden az Entity-k után az autentikáció volt.

Autentikációnkkal lehet:

Regisztrálni az alkalmazásba, bejelentkezni. Ezeken kívül autentikációval erősítjük meg a felhasználó fiókját vagy változtatjuk meg jelszavát.

Email küldés

Regisztrációlál emailt kell küldenünk a felhasználónak, hogy az meg tudja erősíteni fiókját. Ezt a Google SMTP Server szolgáltatásával végezzük.

Ennek használatához létrehoztunk egy Google fiókot amihez applikáció jelszót igényeltünk.

Az alábbi kód egy példa az email küldésre.

```
public async sendWelcomeEmail(user: User, tokenId: string) {
  try {
    await this.mailerService.sendMail({
      to: user.email,
      from: "noreply.ready.team@gmail.com",
      subject: "Ready! üdvözlés",
      template: "welcome",
      context: {
        name: user.name,
        token: tokenId,
      },
    });
  } catch (error) {
    console.log(error);
  }
}
```

Meghívása:

```
await this.emailService.sendWelcomeEmail(createdUser, emailConfirmToken.id);
```

Ez a kód elküldi a felhasználónak a generált email megerősítő kódját a regisztrációnál megadott email címre. Az email kinézetét a 'template' paraméterben adjuk meg ami ebben az esetben az üdvözlő emailé lesz. 'context' paraméterben megadtuk az adatokat amiket az emailben meg kell jeleníteni: ezek a felhasználó neve és a kódja.

Autorizáció

A backend feladatai közé tartozik a jogosultságkezelés. A backend által szolgáltatott erőforrásokat védenünk kell a jogosulatlan hozzáféréstől. Nem engedhetjük meg, hogy például egy vásárló átírja egy büfé étlapját.

Az autorizáció megvalósítására számtalan módszer létezik:

- RBAC: Role-Based Access Control

Ez a módszer szerepköröket társít a felhasználókhöz. Az API műveleteknél pedig meg van határozva, hogy az adott műveletet milyen szerepkörű felhasználók végezhetik el.

- ABAC: Attribute-Based Access Control

Az RBAC továbbfejlesztett változata, itt már a szerepkörökön kívül lehet feltételeket is megadni az egyes erőforrások jogosultságainak beállításánál.

Nekünk komplex jogosultságkezelésre volt szükségünk, ezért az ABAC-ot használtuk. Ennek a megvalósításában a casl nevű npm csomag segített.

Az ABAC 4 részből áll:

- Alany: az a felhasználó, aki végrehajtja a műveletet
- Művelet: meghatározza, hogy mit akar a felhasználó csinálni az erőforrással
- Erőforrás
- Környezet: egyéb körülmények

A casl npm csomag segítségével programkóddal tudtuk megadni az autorizációs szabályokat. Például a termékeknél ez így néz ki:

```

public async createForUser(user?: User) {
  const builder = new AbilityBuilder<ProductAbility>(createMongoAbility);
  const { can } = builder;

  can(Action.Read, Product);

  if (!user) {
    return builder.build();
  }

  const ownCategoryIds = await this.getOwnCategoryIds(user);

  can(Action.Create, [Product, CreateProductDto], {
    categoryId: { $in: ownCategoryIds },
  });

  can(Action.Update, [Product, UpdateProductDto], {
    categoryId: { $in: ownCategoryIds },
  });

  can(Action.Delete, Product, {
    categoryId: { $in: ownCategoryIds },
  });

  return builder.build();
}

```

Adatfeldolgozás

Backendünknek a beérkező adatot memóriában tárolt objektumokká kell átalakítania JSON vagy multipart/form-data formátumból. Ezt idegen szóval deserializatióknak hívják.

A visszaküldött adathoz pedig a memóriában tárolt objektumot alakítjuk át JSON-né. Ezt serializationnak hívják.

Mi ennek a 2 problémának a megoldására a class-transformer és a multer npm csomagokat használtuk.

A class-transformer oda-vissza konvertál TS objektumokat osztálypéldányokká.

A multer multipart-formdata-t dolgoz fel a formban küldött fájlokkal együtt.

Validáció

Fontos, hogy a kientől beérkező adatokat ellenőrizzük. Nem engedhetjük például, hogy a termék ára negatív legyen, vagy a nevének hossza 6000 karakter. Ezt a problémát validációnak nevezzük, és a class-validator npm csomaggal oldjuk meg.

A kientől beérkező adatokat egy dto (data transfer object) osztállyal modellezzük, amelyben felsoroljuk a bekért mezőket, amelyekre validációs szabályokat határozzunk meg.

Kékesi Ádám, yesterday | 1 author (Kékesi Ádám)

```
export class CreateCategoryDto {  
  @Expose()  
  @ApiProperty()  
  @IsString()  
  @MinLength(1)  
  @MaxLength(80)  
  public name: string;  
  
  @Expose()  
  @ApiProperty()  
  @IsNumber({ allowInfinity: false, allowNaN: false, maxDecimalPlaces: 0 })  
  public buffetId: number;  
}
```

Így néz ki például a kategória létrehozására szolgáló dto. A name mező hosszát 1 és 80 karakter közé korlátozzuk, a buffetId mezőnél meg elvárjuk, hogy egész szám legyen.

Ha ezeket a szabályokat megsértjük, akkor a backend hibát küld vissza:

```
"errorCode": "InvalidDataException",
"invalidProperties": {
  "name": {
    "errorMessages": {
      "maxLength": "name must be shorter than or equal to 80 characters",
      "minLength": "name must be longer than or equal to 1 characters",
      "isString": "name must be a string"
    },
    "children": {}
  },
  "buffetId": {
    "errorMessages": {
      "isNumber": "buffetId must be a number conforming to the specified constraints"
    },
    "children": {}
  }
}
```

Frontend

Github url: <https://github.com/14A-D-Ready-team/frontend>

Frontend-hez a TypeScript programozási nyelvre épülő Angular keretrendszert választottuk, mivel az idei évben a tanórák során sokat foglalkoztunk vele és

úgy állapítottuk meg, hogy a mi alkalmazásunkhoz megfelelő funkciókkal rendelkezik:

- Komponens alapú keretrendszer, amely segítségével skálázható webalkalmazások készíthetők.
- Egy jól integrált könyvtár gyűjtemény tartozik hozzá, amelyben sok hasznos funkciót tartalmaz, beleértve az útválasztást (routing), űrlapkezelést (forms management), kliens-szerver kommunikációt és még sok más.
- Tartalmaz fejlesztői csomagokat, amelyek segítik a kód fejlesztését, összeállítását, tesztelését és frissítését.

Az Angular mellett az Ionic keretrendszert is használtuk, amely egyszerűbbé tette mind a nagyképernyős, mind a mobil nézet elkészítését. Rengeteg előre elkészített modern felhasználói felületelemet tartalmaz, amelyek kinézete könnyedén testreszabhatóak. Az Ionic mellett a Capacitort is használtunk, amely segítségével az alkalmazás könnyedén kiadható mind Android, mind IOS operációs rendszerekre.

Autentikáció

Az első lépés frontenden az autentikációs oldalak elkészítése volt. Ide tartozik a bejelentkezés, a regisztráció, a felhasználó megerősítése és a jelszó visszaállítása.

Az autentikációs oldalaknál az űrlapokhoz a reactive forms-t használtuk, amely segítségével dinamikus űrlapok hozhatók létre. A reactive formok lehetővé teszik az űrlap elemek, mint például az input mezők, a checkbox-ok és a radio button-ok dinamikus kezelését és validálását.

A reactive formok előnyei:

- Egyszerűbb hibakezelés: A reactive formok lehetővé teszik a hibák egyszerűbb kezelését, és lehetővé teszik az űrlap elemek állapotának nyomon követését.
- Egyszerű validálás: A reactive formok lehetővé teszik a dinamikus validálást, ami azt jelenti, hogy az űrlap validálása a felhasználói interakciókra történik.

A reactive forms használatához az Angular Forms modulra van szükség. A Forms modul a ReactiveFormsModule függőséggel importálható. Ezután az űrlap elemeket a FormBuilder segítségével lehet inicializálni, majd az FormGroup használatával lehet az állapotokat és a validációt kezelni.

```
public signupForm: FormGroup<SignupForm>;

@Select((state: { signup: SignupStateModel }) => state.signup.status)
public signupStatus!: Observable<SignupStatus>;

constructor(private store: Store) {
  this.signupForm = new ClassValidatorFormGroup<SignupForm>(SignupDto, {
    name: new ClassValidatorFormControl<string>(""),
    email: new ClassValidatorFormControl<string>(""),
    password: new ClassValidatorFormControl<string>(""),
  });
}

public signup() {
  this.store.dispatch(new Signup());
}
```

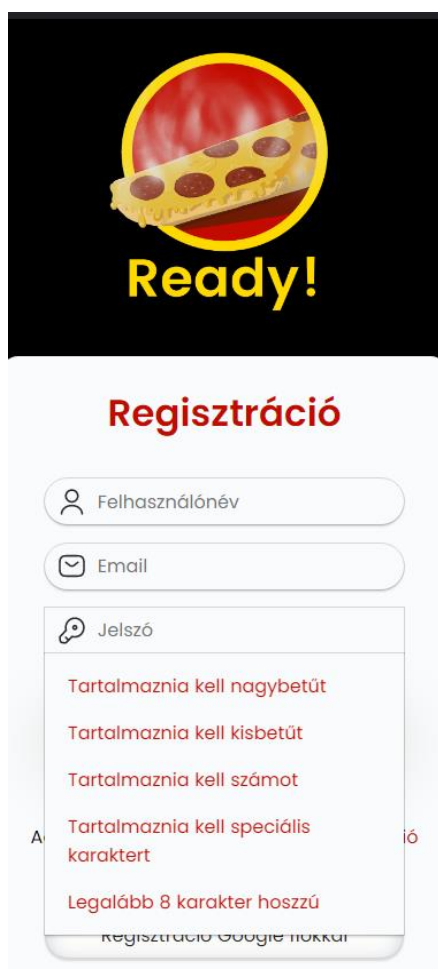
Regisztráció

Kétféle regisztrációs oldal is készült, egyik a vásárlóknak, akiknek gyorsabb Google regisztráció is elérhető, illetve egy külön oldal az adminoknak, büfé tulajdonosoknak és büfé dolgozóknak.

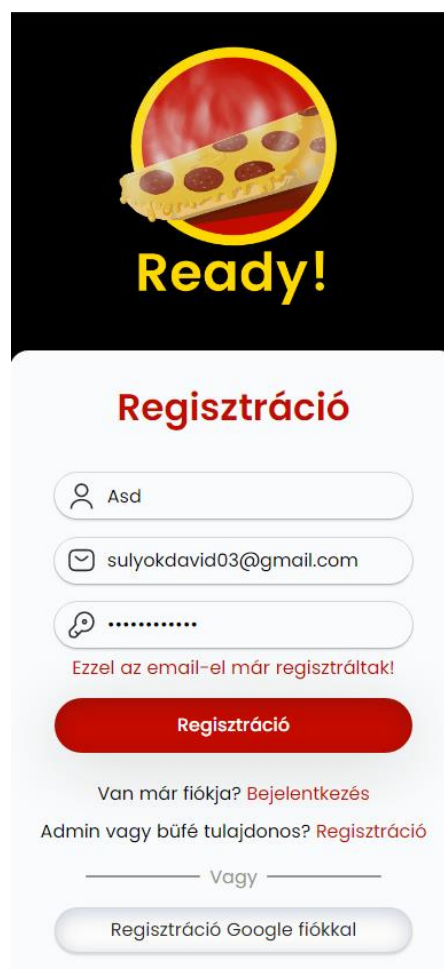
A regisztrációs oldalak a már fentebb is említett reactive forms segítségével készültek. A regisztrációs oldalon 3 input mező található, amelyek az admin regisztrációs fülön kiegészülnek egy felhasználó típusa választó mezővel. A felhasználó az adatai megadása után a regisztráció gombra kattintva tud

regisztrálni az alkalmazásba. A felhasználónak meg kell adnia egy nevet, egy email címet és egy jelszót és esetlegesen a felhasználó típusát.

Az űrlap addig nem engedi továbbküldeni a felhasználó adatait a backend felé, ameddig minden mezőt helyesen ki nem töltött. Amennyiben a felhasználó hibásan adta meg valamelyik adatát, az input mezők ezt jelzik validáció segítségével. Egyrészt piros színezéssel, másrészt a hiba pontos részleteivel. A szerveroldali hibákat is jelzi a felület, amelyhez külön komponenst készítettünk.



The screenshot shows the 'Regisztráció' (Registration) form. At the top is a logo with a pizza slice and the word 'Ready!'. Below the title, there are three input fields: 'Felhasználónév' (Username), 'Email', and 'Jelszó' (Password). The 'Jelszó' field is highlighted with a red border and contains several red error messages: 'Tartalmaznia kell nagybetűt' (Must contain uppercase letters), 'Tartalmaznia kell kisbetűt' (Must contain lowercase letters), 'Tartalmaznia kell számot' (Must contain a number), 'Tartalmaznia kell speciális karaktert' (Must contain a special character), and 'Legalább 8 karakter hosszú' (At least 8 characters long). At the bottom, there is a link for 'Regisztráció Google fiókkal' (Register with Google account).



The screenshot shows the 'Regisztráció' (Registration) form after a successful registration. The 'Email' field now contains 'sulyokdavid03@gmail.com' and a red message below it says 'Ezzel az email-el már regisztráltak!' (Already registered with this email!). The 'Jelszó' field is now empty. The 'Regisztráció' button is highlighted in red. Below the button, there are links for 'Van már fiókja? Bejelentkezés' (Already have an account? Login) and 'Admin vagy büfé tulajdonos? Regisztráció' (Admin or cafe owner? Registration). At the bottom, there is a link for 'Regisztráció Google fiókkal' (Register with Google account).

Sikeres regisztráció után a felhasználó email üzenetet kap fiókjának megerősítéséhez. Ennek hiányában a felhasználó nem tud bejelentkezni és inaktív felhasználó marad, tehát nem fér hozzá az alkalmazáshoz.

Bejelentkezés

Bejelentkezés oldalból egy készült, amelyen keresztül az összes felhasználó típus be tud jelentkezni.

Az ezen az oldalon található űrlapon egy email cím és jelszó párossal lehet bejelentkezni, illetve lehetőség van Google bejelentkezésre is. A szerveroldali hibákért itt is az ennek létrehozott külön komponens felel.

Az oldalon található két link, melyek elvezetnek a jelszó vagy email megerősítést kérő oldalra.

Email megerősítés és jelszó visszaállítás

Ezen oldalak űrlapján egy input mező található a felhasználó email címének, illetve egy küldés gomb. A felhasználó itt email címének megadásával tud jelszó emlékeztető, illetve hitelesítő email-t kérni.

Vásárlói oldalak

Büféválasztó oldal

Bejelentkezés után, az alkalmazás erre az oldalra vezeti a felhasználót. Itt található egy “Válasszon büfét” gomb, amelyre kattintva megjelenik az elérhető büfék listája. Ebből a listából kiválaszthatjuk melyik büféből szeretnénk majd vásárolni.

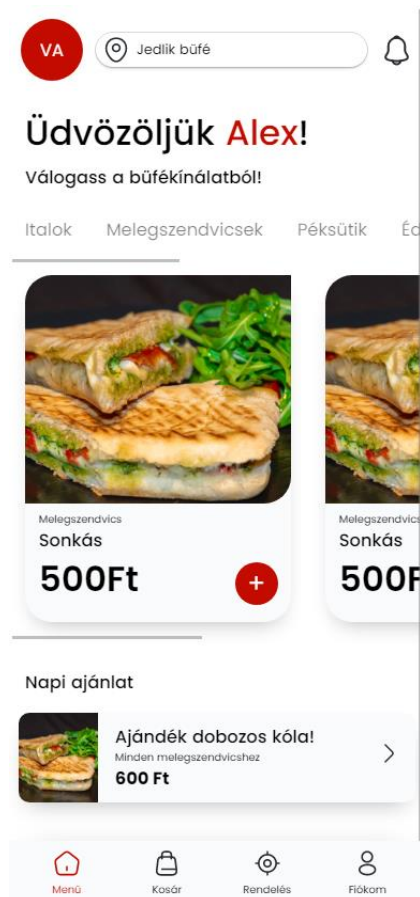
A kiválasztás után különböző adatok jelennek meg az adott büféről:

- Büfé neve
- Büfé címe
- Leírása

A kiválasztás gombra kattintva az alkalmazás átvezet az adott büfé főoldalára.

Főoldal

Amennyiben van kiválasztott büfé, a felhasználó itt tekintheti meg annak kínálatát.



Az oldal tetején egy gomb formájában van lehetőség megtekinteni az aktuálisan kiválasztott büfét. Erre a gombra kattintva visszavezet az alkalmazás minket a büfé választó oldalra.

A termékeket kategóriákra bontva lehet megtekinteni. A kategóriák kiválasztására radio gombok vannak létrehozva, amelyekre kattintva megjelennek az adott kategóriába tartozó termékek a gombok alatt.

Minden termékhez egy kártya tartozik, amelyen megjelenik annak képe, neve, kategóriája, árazása, valamint egy gomb, amellyel megtekinthető az adott termék egyedi oldala és testreszabhatósága.

A termékek alatt a napi ajánlatok kapnak helyet, amelyek értesítenek az adott napi leárazásokról.

Termékek egyedi oldala

Ezen az oldalon a kiválasztott termék részletes leírása és személyreszabhatósági opciói jelennek meg, valamint itt lesz majd lehetőségünk a terméket hozzáadni a kosárhoz.

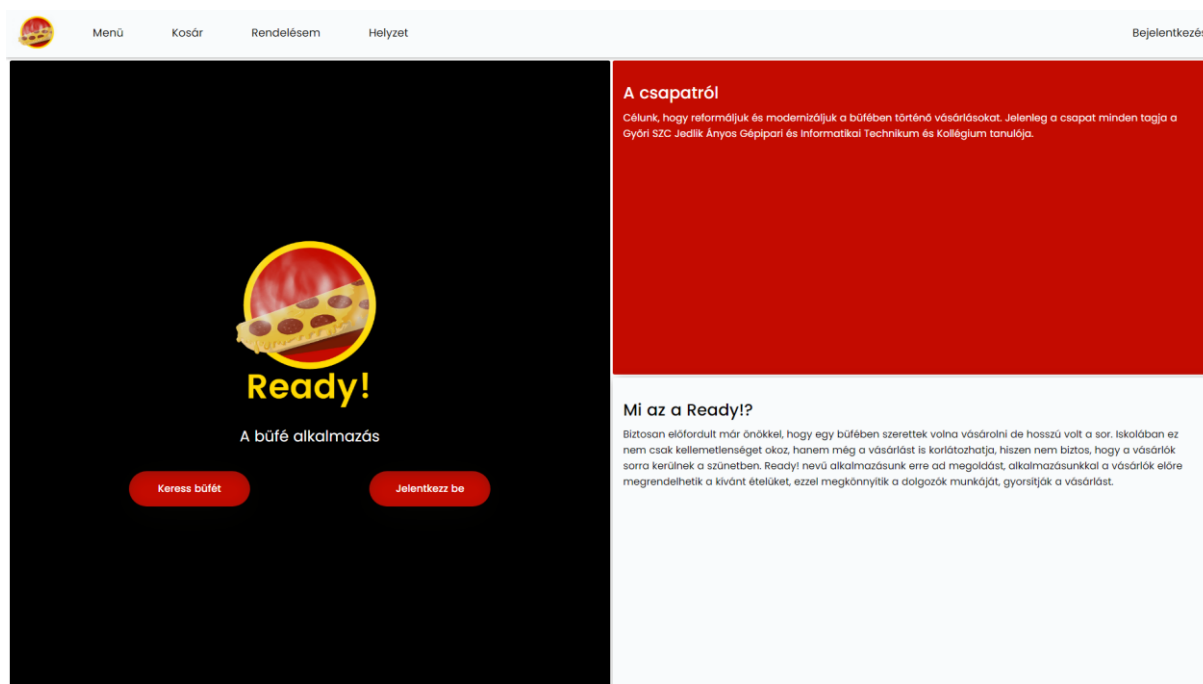


A személyre szabásra két opció lehetséges, a több illetve az egy választós. Termékenként változik miket lehet rajtuk változtatni vagy hozzáadni.

Desktop felhasználói kinézet

A felhasználóknak lehetőségük van egy egyszerű böngészőből is elérni és használni az alkalmazást. Ehhez egy külön kinézetet készítettünk, amely a nagyobb képernyőkhöz igazodik.

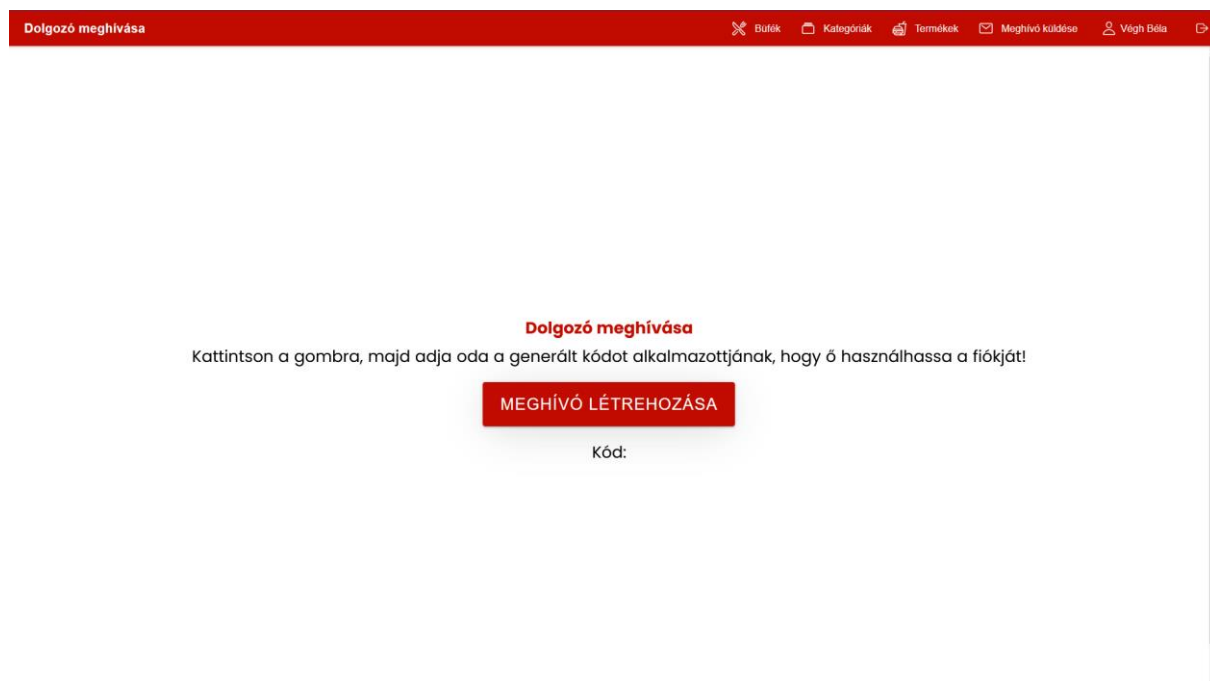
Egyik fő különbség, hogy az alkalmazás ezen formájában bejelentkezés előtt van egy üdvözlő oldal, amely különböző információkkal szolgál a Ready!-ről.



Admin oldalak

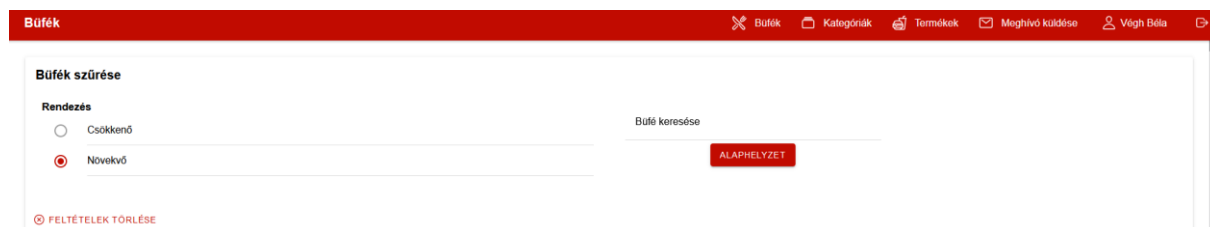
Főoldal

Eléréséhez büfé dolgozó, büfé tulajdonos vagy admin fiókkal be kell jelentkezni. Az admin felület főoldalán lehet meghívni a büfébe a dolgozókat. Ehhez ki kell választani egy büfét a Büfék oldalról.

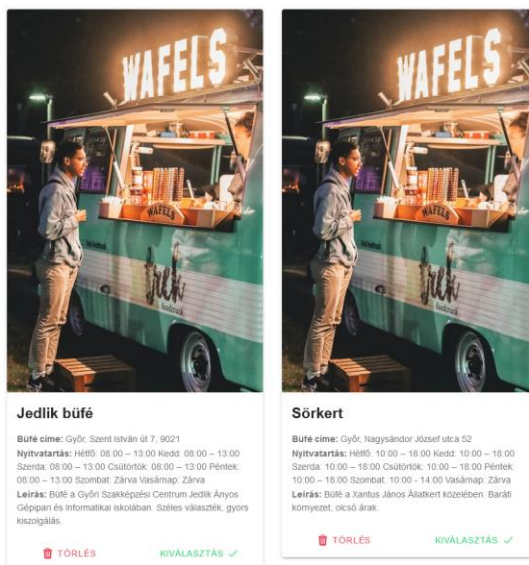


Büfé oldal

Itt találhatóak a büfék listába rendezve. Büféket lehet A-Z-ig vagy Z-A-ig rendezni, valamint lehet rájuk keresni.



Erről az oldalról érhető el a büfé szerkesztő és a büfé létrehozó oldal is.



Büfé szerkesztő oldal

Itt lehet szerkeszteni már létező büfét. Ha nincs változás akkor nem lehet menteni.

Büfé szerkesztése

Büfé képe

Név

Jedlik büfé

Büfé címe

Győr, Szent István út 7, 9021

Koordináták

47.68245999072101, 17.63033089810631

Nyitvatartás

Hétfő: 08:00 – 13:00
Kedd: 08:00 – 13:00
Szerda: 08:00 – 13:00
Csütörtök: 08:00 – 13:00
Péntek: 08:00 – 13:00
Szombat: Zárva
Vasárnap: Zárva

Leírás

Büfé a Győri Szakképzési Centrum Jedlik Ányos Gépipari és Informatikai iskolában. Széles választék, gyors kiszolgálás.

MÉGSE MENTÉS

Új büfé oldal

Ezen az oldalon lehet felvenni új büfét. A büféhez kötelező képet is feltölteni amit mediumblobba tárolunk backenden.

Új büfé létrehozása

Büfé képe

Leírás

0 / 800

MÉGSE

MENTÉS

Név

0 / 100

Büfé címe

0 / 100

Koordináták

0 / 100

Nyitvatartás

0 / 200

Kategóriák oldal

Megjeleníti a kategóriákat, lehet újat létrehozni, törölni.

Kategóriák		Büfék Kategóriák Termékek Meghívó küldése Fekete Miklós	
Italok			
Melegszendvicsek			
Péksütek			
Édességek			
Gyorsételek			
Fincsi ☺			
Forró italok			
Nassolnivalók			

+

Termékek oldal

A Büfék oldalhoz hasonlóan működik csak termékeket jelenít meg. Ezen az oldalon lehet szűrni a termék kategóriájára, teljes árára, akciós árára és arra, hogy hány darab elérhető a termékből.

Termékek szűrése

Kategória
Nincs kiválasztva

Tejles ár
—

Csökkentett ár
—

Raktáron
—

ÉRTÉK TARTOMÁNY

ALAPHELYZET


ÉRTÉK TARTOMÁNY

ALAPHELYZET

ÉRTÉK TARTOMÁNY

ALAPHELYZET

🔍 FELTÉTELEK TÖRLÉSE



Kinley


Aliquam fugit ipsum quidem. Iure recusandae expedita nisi eius. Nam perspiciatis consectetur id.

Tejles ár: HUF1,048.00
Csökkentett ár: HUF1,226.00
Raktáron: 74

Opciók:

●●● Méret

TÖRLÉS



Sonkás melegszendvics

Dignissimos temporibus exercitationem delectus placeat veritatis provident esse quidem. Dolorem molestias possimus tempore ipsum molestias cumque omnis. Repudiandae ab libero quibusdam perspiciatis magni nesciunt soluta et. Ratione amet minima assumenda fugiat ad. Et assumenda reprehenderit nostrum eorum necessitatibus consectetur dignissimos iusto repudiandae.


Tejles ár: HUF3,366.00
Csökkentett ár: HUF792.00
Raktáron: 82

Opciók:

●●● Szósz

●●● Sajt

TÖRLÉS



Gombás melegszendvics

Aut modi voluptatem quidem atque aliquam nobis. Labore suscipit aliquam cum voluptatibus. Sequi neque alias in. Pariatur dignissimos tempora et.


Tejles ár: HUF3,699.00
Csökkentett ár: HUF2,296.00
Raktáron: 4

Opciók:

●●● Szósz

●●● Sajt

TÖRLÉS



Sonkás-kukoricás melegszendvics

Dolorem dolor architecto eum amet quod facilis. Quae incidunt repudiandae dolorem aut. Totam voluptate quam nesciunt facere commodi eius. Quisquam dolor iusto laborum odit temporibus nemo odio ut qui.

Tejles ár: HUF3,293.00
Csökkentett ár: HUF4,303.00
Raktáron: 60

Opciók:

●●● Szósz

●●● Sajt

TÖRLÉS

Tesztelés

Backend

Backendben unit teszteltünk a jest npm csomag segítségével. Például az exceptionFactory segédmetódus lefedettsége:

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Line #s
src/shared/validation/utils	100	95.45	100	100	
exception-factory.util.ts	100	95.45	100	100	38

Részlet a unit tesztből:

```

describe("validation/exceptionFactory", () => {
  Run | Debug
  it("should be able to build an exception from a single ValidationError without children", () => {
    const errors: ValidationError[] = [
      {
        property: "testprop1",
        constraints: { c1: "Errormessage 1", c2: "Errormessage 2" },
      },
    ],
  ];

  const result = exceptionFactory(errors);

  expect(result).toBeInstanceOf(InvalidDataException);
  expect(result.invalidProperties).toBeInstanceOf(Map);
  expect(result.invalidProperties.size).toBe(1);
  expect(result.invalidProperties.get("testprop1")).toBeInstanceOf(
    InvalidProperty,
  );
  expect(
    result.invalidProperties.get("testprop1").errorMessages,
  ).toBeInstanceOf(Map);
  expect(result.invalidProperties.get("testprop1").errorMessages.size).toBe(
    2,
  );
  expect(
    result.invalidProperties.get("testprop1").errorMessages.get("c1"),
  ).toBe("Errormessage 1");
  expect(
    result.invalidProperties.get("testprop1").errorMessages.get("c2"),
  ).toBe("Errormessage 2");

  expect(result.invalidProperties.get("testprop1").children).toBeInstanceOf(
    Map,
  );
  expect(result.invalidProperties.get("testprop1").children.size).toBe(0);
});

Run | Debug
it("should be able to build an exception from a single ValidationError without any constraints", () => {
  const errors: ValidationError[] = [{ property: "testprop1" }];

```

E2E

End-to-end tesztelésre a Cypress technológiát használtuk, amellyel a UI böngészését lehet automatizálni. A Cypress képernyőképet és videót is készít a tesztelésről, amelyeket a mellékletek között találhatók. Például itt a bejelentkező oldal e2e tesztje:

```
describe("Login page", () => {
  it("Logs user in", () => {
    cy.viewport("iphone-xr");
    cy.visit("http://localhost:4200/login");
    cy.get("#email").type("kekesi.adam@students.jedlik.eu");
    cy.get("#pswd").type("Supa$3cr3t!!!");
    cy.screenshot("form fields filled");
    cy.get(".ready-btn").click();
    cy.screenshot("clicked login");
    cy.url().should("contain", "buffet-select");
    cy.wait(3000);
    cy.screenshot("login successful");
  });

  it("Tries wrong password", () => {
    cy.viewport("iphone-xr");
    cy.visit("http://localhost:4200/login");
    cy.get("#email").type("kekesi.adam@students.jedlik.eu");
    cy.get("#pswd").type("wrong");
    cy.get(".ready-btn").click();
    cy.url().should("contain", "login");
    cy.get(".validation-text").should(
      "have.html",
      "Hibás email és jelszó páros!",
    );
    cy.wait(3000);
    cy.screenshot("error message");
  });
});
```

Az első teszt helyes jelszóval próbál bejelentkezni, és leellenőrzi, hogy a bejelentkezés sikeres-e, és az alkalmazás átnavigál-e a büféválasztó oldalra.

A második teszt hibás jelszóval próbálkozik, és leellenőrzi, hogy a bejelentkezés sikertelen volt-e, és hibaüzenet megjelent-e.

Források:

Skillshare agile/scrum course

<https://kinsta.com/blog/monorepo-vs-multi-repo/>

<https://gist.github.com/stephenparish/9941e89d80e2bc58a153>

<https://medium.com/javarevisited/5-different-git-workflows-50f75d8783a7>

https://hu.wikipedia.org/wiki/Objektum-rel%C3%A1ci%C3%B3s_lek%C3%A9p%C3%A9s

<https://expressjs.com/>

<https://nestjs.com/>

<https://nodejs.org/en/>

<https://www.typescriptlang.org/docs/handbook/decorators.html>

<https://mikro-orm.io/>

<https://learn.microsoft.com/hu-hu/azure/architecture/best-practices/api-design>

<https://frontegg.com/guides/abac>

<https://www.okta.com/blog/2020/09/attribute-based-access-control-abac/>

<https://blog.logrocket.com/understanding-typescript-object-serialization/>

<https://github.com/typestack/class-transformer>

<https://casl.js.org/v6/en/>

<https://www.npmjs.com/package/multer>

<https://angular.io/guide/what-is-angular>

<https://ionicframework.com/>

<https://ideafest.hu/>

<https://wolt.com/hu/hun>

<https://www.youtube.com/@JoshuaMorony>

<https://sass-lang.com/>

<https://www.figma.com/>

<https://www.youtube.com/watch?v=7SDpTOLeqHE>

<https://restfulapi.net/>

<https://javascript.plainenglish.io/how-to-work-with-input-fields-in-cypress-b1c7c22a1156>

<https://www.cypress.io/>