



Széchenyi István Katolikus Technikum és Gimnázium

Szoftverfejlesztő és -tesztelő projektfeladat

„Kalória Kalkulátor”

Készítették:

Antal Dominik Zoltán

Gál Gergő

Mészáros Róbert

Ózd, 2025

Tartalomjegyzék

Bevezetés.....	3
Felhasználói dokumentáció	4
Rendszerkövetelmények.....	4
A webalkalmazás hardverkövetelménye	4
Az alkalmazás futtatásához szükséges szoftverek	4
Webalkalmazás indítása	4
Webalkalmazás használata	5
Fejlesztői dokumentáció.....	12
Alkalmazott fejlesztői és csoportmunka eszközök.....	12
Adatbázis	12
Frontend	12
Backend	12
Adatbázis	13
Modell leírása	13
Táblák, kapcsolatok.....	14
E-K diagram	16
Frontend	17
Backend	32
Tesztelés	36
Frontend tesztelése	36
Backend tesztelése.....	40
Továbbfejlesztési lehetőségek	45
Irodalomjegyzék	46

Bevezetés

A táplálkozási problémák igen gyakoriak a világon, viszont alkalmazásokat, amik erre segítséget nyújtanának, keveset találtunk, és mindegyik régi. Ezért tehát a projektünk a kalória és tápanyag bevitel számontartásához és a táplálkozási célok eléréshez készült.

Az oldal használatához először regisztrálni kell, hogy mindenki egyénileg tudja a saját táplálkozását követni, alakíttatni.

A weboldal felületén ki lehet választani az aznap bevitt ételeket, aminek tápanyagértékei (kalória, fehérje, zsír, szénhidrát) hozzáadódnak az aznapi már bevitt ételekhez.

Ha nincs az adatbázisban az adott étel, akkor azt fel lehet vinni az adatbázisba a nevének és tápértékeinek megadása után. Ez megkönnyíti a későbbiek folyamán a felhasználók étkezésének eltárolását, mert folyamatosan bővílhet az adatbázis sok különböző étellel, így több esély van arra, hogy a felhasználó megtalálja, amit eltárolna, nem kell feltöltenie.

Amikor az oldalon rögzítünk magunknak egy étkezést, eltárolódik az aktuális dátum és idő, hogy később vissza lehessen nézni, hogy mikor mit ettünk.

Összességében tehát az alkalmazás célja, hogy segítsen a táplálkozásban és a kalóriabevitelben azok számára, akik egészségesebben szeretnék a mindennapok folyamán étkezni.

A GitHub repository linkje: <https://github.com/ntldmnk/calculatorproject>

Felhasználói dokumentáció

Rendszerkövetelmények

A webalkalmazás hardverkövetelménye

	Ajánlott	Minimális
Processzor (CPU)	Intel(R)_Core(TM)_i3-9100F_CPU_@_3.60GHz	Intel Core i3-6100T Dual-Core 3.2GHz
Operációs rendszer (OP)	Windows 10 pro	Windows 10
RAM	8 GB Ram	6 GB RAM
Tárhely	6GB	6GB
Internet	Széles sávú	Széles sávú
GPU	NVIDIA GeForce GTX 1050	Intel(R) UHD Graphics 620

Az alkalmazás futtatásához szükséges szoftverek

XAMPP, NodeJS, Angular CLI, Composer, és egy böngésző.

Webalkalmazás indítása

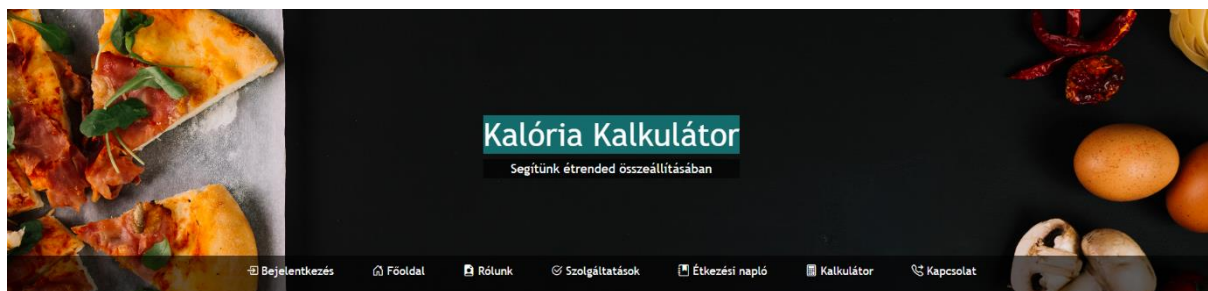
A backend üzembe helyezése a Composer és a XAMPP telepítésével kezdődik. Első lépésként a XAMPP Control Panelt meg kell nyitnunk, és ott elindítani a MySql és Apache modulokat, ez létrehozza a kapcsolatot a lokális adatbázissal. Ezután egy parancssorban elnavigálunk a „backend” mappába, majd a következő parancsot lefuttatjuk: „php artisan migrate --seed”. Ez létrehozza az adatbázis, és feltölti a regisztrációkat tartalmazó adattáblát. Következő lépésként meg kell nyitnunk a XAMPP Control Panel phpMyAdmin felületét, a MySql-lel egy vonalban lévő „Admin” gombra kattintva. Ezen a felületen a bal oldali oszlopban kikeressük az „etelek” adatbázist. Ha megtaláltuk és kiválasztottuk, a képernyő tetején lévő SQL gombra kattintva megjelenítünk egy beviteli mezőt a képernyő tetején. Abba a beviteli mezőbe beillesztjük az „adatbázis” mappában található „adatbázis-feltoltes.sql” fájl tartalmát, majd a beviteli mező alatt található „indítás” gombra kattintunk, melynek hatására az adatbázisunk fel fog tölteni adatokkal. A parancssorban a backend mappában tartózkodva le kell futtatnunk a következő parancsot: „php artisan serve”. A backendünk ennek hatására kapcsolódik, és már eléri a korábban feltöltött adatbázist.

A frontend üzembe helyezéséhez a NodeJs telepítésére lesz szükség. Miután ez megtörtént, parancssorban le kell futtatnunk a következő parancsot: „npm install -g @angular/cli”. Ez telepíteni fogja az „Angular CLI-t”, amely lehetővé teszi az alkalmazás futtatását. Parancssorban a „frontend” mappába navigálva futtatnunk kell a következő parancsot: „npm install”. Miután ez lefutott, és feltelepült a „node_modules” mappa, futtatható a következő

parancs: „ng serve -o”. Ennek hatására az alkalmazás el kell induljon az alapértelmezett böngészőnkben.

Webalkalmazás használata

A webalkalmazás összes oldalán található egy navigációs sáv, ahol található a bejelentkezés/regisztráció gomb, ahol egy felhasználói fiókot regisztrálhatunk magunknak, hogy a webalkalmazást használhassuk. Ezen kívül még található ebben a sávban több gomb, ami különböző oldalakra juttat el bennünket.



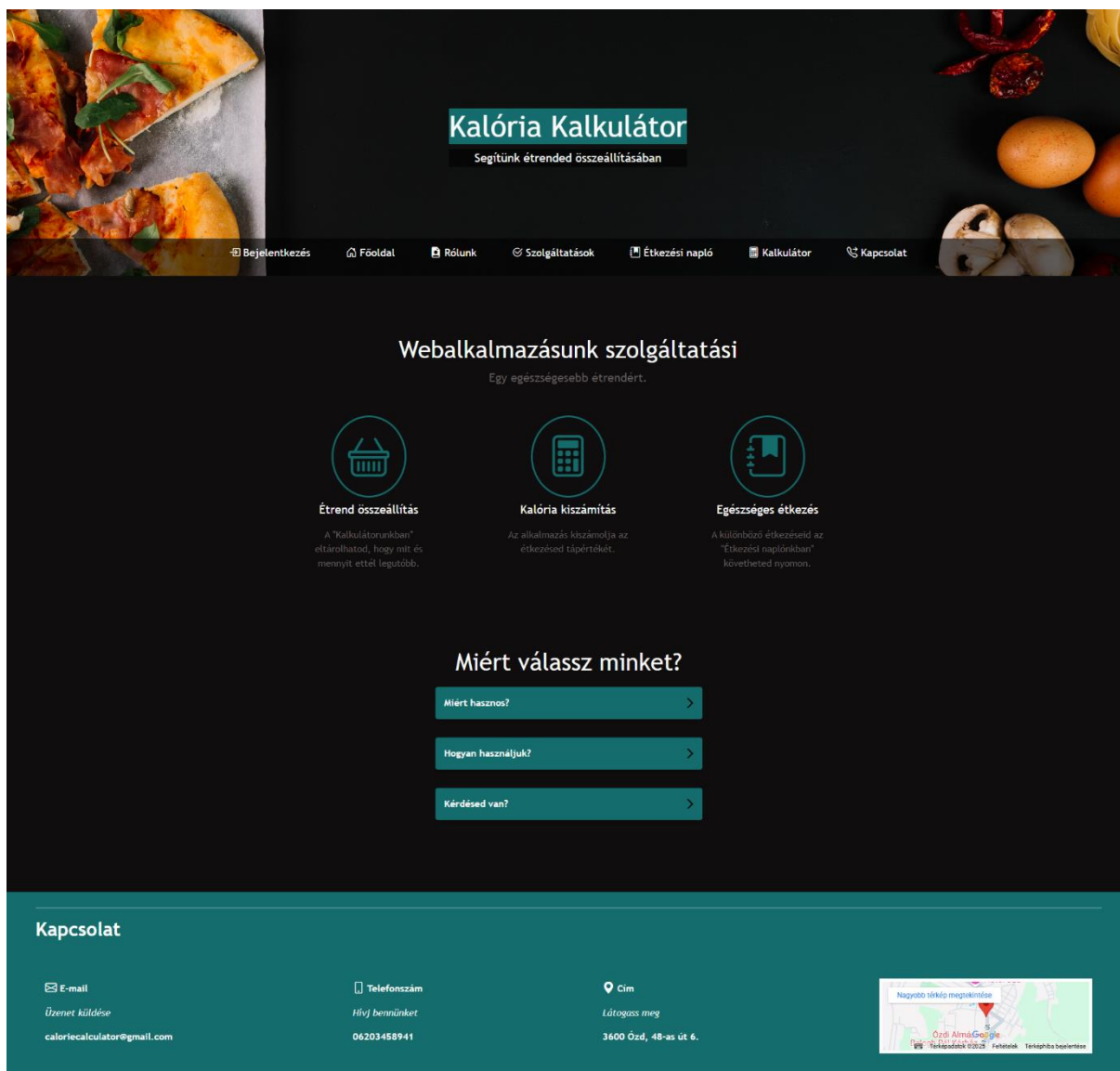
1. kép: navigációs sáv

A navigációs sávhoz hasonlóan található minden oldalon egy lábléc, ahol elérhetőségek találhatók.



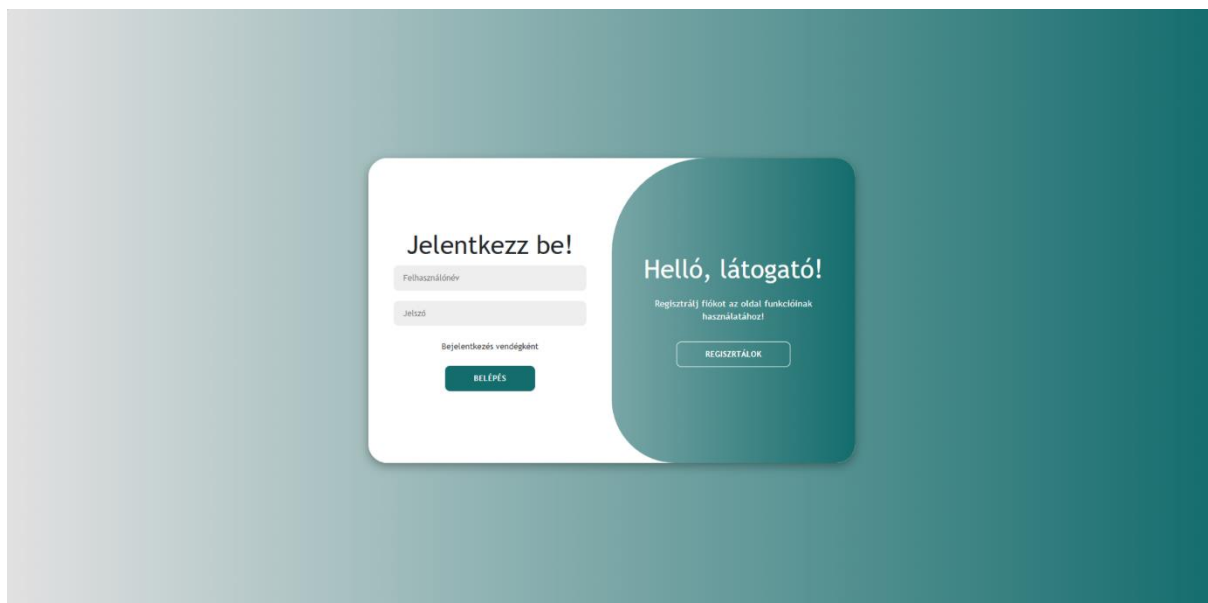
2. kép: lábléc

A főoldalon általános információk olvashatók a webalkalmazásról.

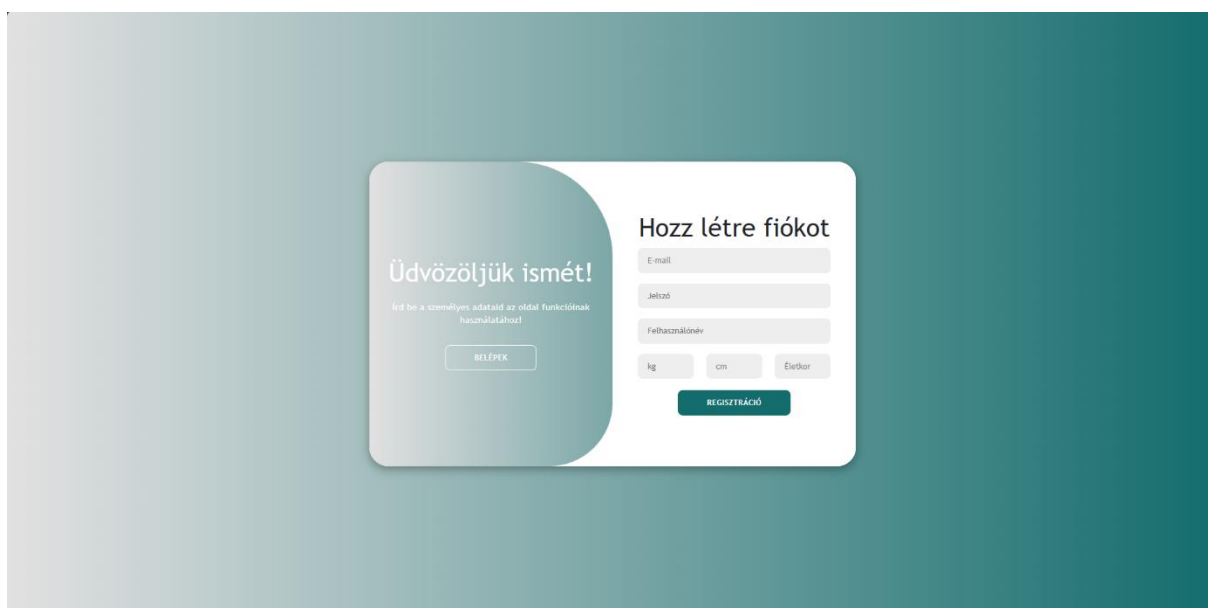


3. kép: főoldal

A navigációs sávon a „Bejelentkezés” gombra kattintva bejelentkezhettek meglévő fiókukba, vagy regisztrálhattok egy új fiókot.



4. kép: bejelentkezési menü



5. kép: regisztrációs menü

A „rólunk” gombra kattintva a webalkalmazás fejlesztőinek bemutatkozóí oldalára juthatunk el.

Kalória Kalkulátor
Segítünk étrended összeállításában

Bejelentkezés Foodal Rólunk Szolgáltatások Étkezési napló Kalkulátor Kapcsolat

Az alkalmazás készítői:

Antal Dominik
Frontend tervező / fejlesztő

“ Az informatika az általános iskola utolsó évében kezdett el foglalkoztatni és ezért választottam továbbtanulásnál a Szoftverfejlesztő- és tesztelő szakot. Inkább a frontend és a design részében találtam meg magamat, ahol a kreativitásomnak is teret tudok adni és ezt érzem magamhoz igazán közelnek. Amellett, hogy sokszor kihívást jelent egy feladatnak a megoldása, ez sikerélményt is okoz és ezért szeretek igazán programozni, hiszen ez folyamatosan motivál és előrébb visz. A csapalmunkában jónak tartom magamat, könnyen alkalmazkodom társaimhoz, szívesen segítek, illetve szívesen tanulok új dolgokat is. Ezen a vonalon szeretném a tanulmányaimat folytatni az egyetemen, programtervező informatikus karon.

“

Mészáros Róbert
Adatbázis tervező / fejlesztő

“ Egész gyerekkoromban érdekelték, egyenesen vonzottak a számítógépek, a működésük, felépítésük kívül-belül, hardveresen, szoftveresen. Ezért 2020-ban Szoftverfejlesztő és tesztelő szakra jelentkeztem a SZIKSZI-be, hogy közelebbről megismerhessem, hogyan is jöhetnek létre különböző szoftverek, és megtanulhassam, hogy hogyan lehetnék én is programozó. A tanulmányaim során több programozási nyelvet is elsajátítottam, viszont amiben a legbiztosabbnak érzem tudásom, az az adatbázis kezelés, ezért ebben a webalkalmazás fejlesztő projektben elvállaltam az adatbázis megtervezését és fejlesztését.

“

Gál Gergő
Backend tervező / fejlesztő

“ Az informatika általános iskolás koromban felkeltette az érdeklődésemet, így a középiskolai tanulmányaimat a szoftverfejlesztő és tesztelő szakon folytatom. A programozáson belül leginkább a frontend és a backend területei fogtak meg, hiszen egyszerre adnak lehetőséget a logikai gondolkodásra és a problémamegoldásra, amit szeretek a programozásban. Szeretnék továbbtanulni a mérnök-informatikus szakon az egyetemen, ahol mélyebb tudást szerezhetek, és tovább építhetem azt az alapot, amit a középiskolában megszereztem. Hiszem, hogy az informatika folyamatos fejlődése újabb és újabb kihívásokat tartogat, és ezek megoldása az, ami igazán érdekel.

“

Kapcsolat

E-mail
Üzenet küldése
caloriecalculator@gmail.com

Telefonszám
Hívj bennünket
06203458941

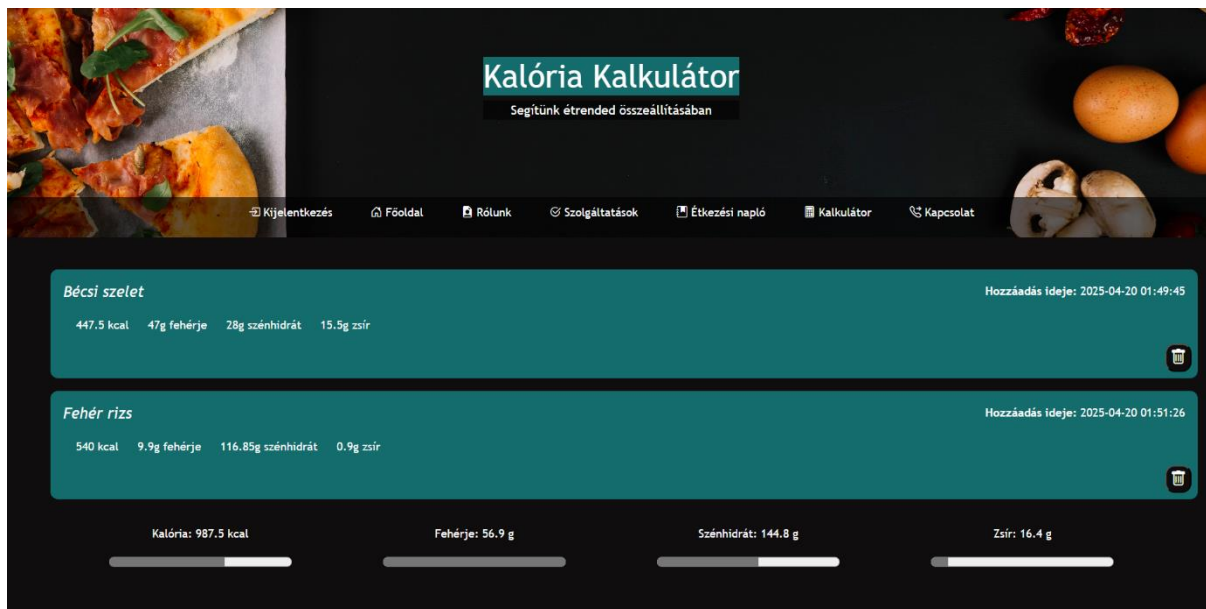
Cím
Látogass meg
3600 Ózd, 48as út 6.

Nagobb térkép megtekintése
Órdi Alma Google
Térkép készítője: Google
Földrajzi adatok forrása: OpenStreetMap

6. kép: a fejlesztők bemutatkozóí oldala

A „szolgáltatások” gomb a főoldalra visz, azon belül legörget egyenesen a webalkalmazás szolgáltatásaihoz.

Az „Étkezési napló” gomb a naplózási felületre visz, ahol a kalkulátorban rögzített étkezések lesznek megtekinthetők, illetve, hogy az aznapi rögzített étkezések mennyire töltik ki a napi ajánlott tápanyag bevitelt.



7. kép: étkezési naplók oldala

A „kalkulátor” gomb a kalkulátorhoz visz, ahol az adatbázisban található ételek és köretek közül fogjuk tudni kiválasztani, hogy miből mennyit ettünk aznap.

X

Adjon hozzá egy ételt:

Nem találja az ételét? [Adja hozzá.](#)

Keresés...

Főételek

Lecsó

Fehérje: 0.9

Szénhidrát: 5.8

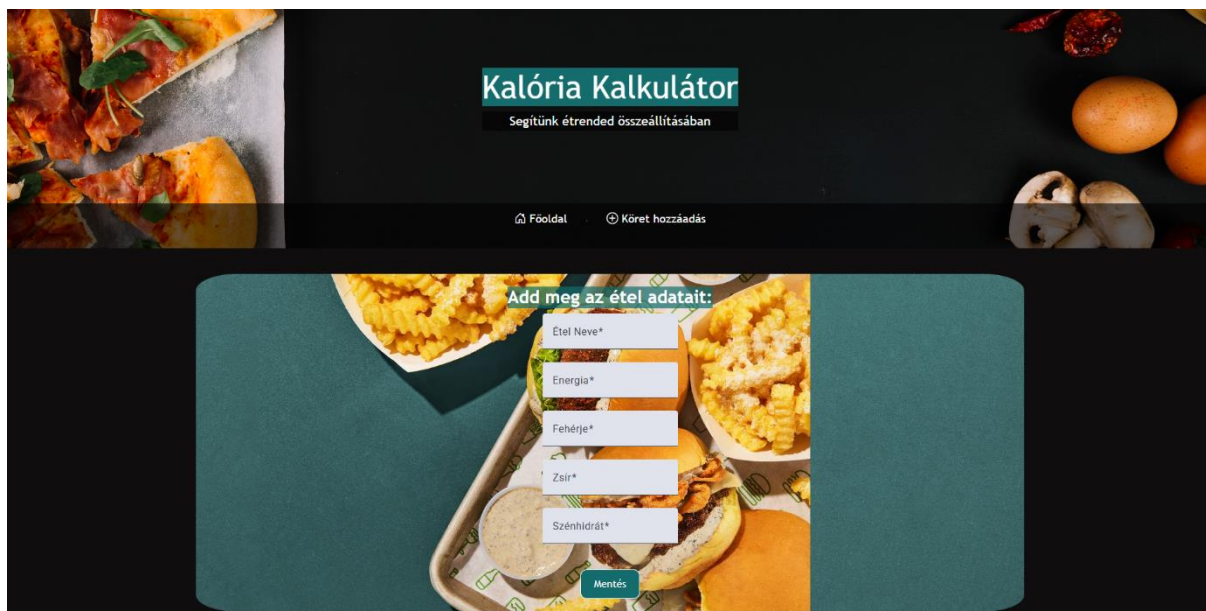
Zsír: 0.8

kcal: 31/100g

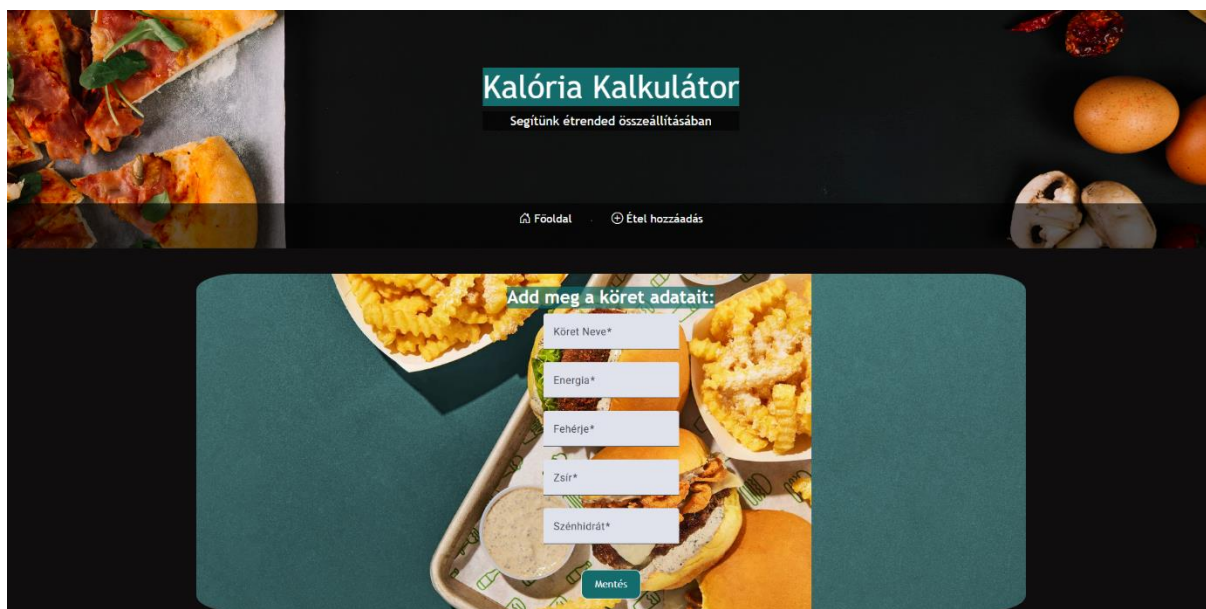
Mennyiség g

8. kép: kalkulátor

Ha nem találjuk a rögzíteni kívánt ételt, akkor a „Nem találja az ételét? Adja hozzá.” szövegre kattintva megnyithatóak az étel, illetve köret feltöltő felületek. A két felület között a navigációs sávon lévő „Étel hozzáadás”, illetve „Köret hozzáadás” gombokkal lehetséges. Amikor a feltöltésekkel készen vagyunk, a „Főoldal” gombra kattintva visszaléphetünk a főoldalra.

The screenshot shows the 'Kalória Kalkulátor' (Calorie Calculator) website. The header features the title 'Kalória Kalkulátor' and the subtitle 'Segítünk étrended összeállításában'. Below the header is a navigation bar with 'Főoldal' (Home) and 'Étel hozzáadás' (Add food). The main content area displays a form titled 'Add meg az étel adatait:' (Add food data). The form includes input fields for 'Étel Neve*' (Food Name*), 'Energia*' (Energy*), 'Fehérje*' (Protein*), 'Zsír*' (Fat*), and 'Szénhidrát*' (Carbohydrates*). A 'Mentés' (Save) button is located at the bottom of the form. The background of the form area shows a collage of various food items like corn, bread, and vegetables.

9. kép: étel feltöltési menü

The screenshot shows the 'Kalória Kalkulátor' (Calorie Calculator) website. The header features the title 'Kalória Kalkulátor' and the subtitle 'Segítünk étrended összeállításában'. Below the header is a navigation bar with 'Főoldal' (Home) and 'Étel hozzáadás' (Add food). The main content area displays a form titled 'Add meg a köret adatait:' (Add side dish data). The form includes input fields for 'Köret Neve*' (Side dish Name*), 'Energia*' (Energy*), 'Fehérje*' (Protein*), 'Zsír*' (Fat*), and 'Szénhidrát*' (Carbohydrates*). A 'Mentés' (Save) button is located at the bottom of the form. The background of the form area shows a collage of various food items like corn, bread, and vegetables.

10. kép: köret feltöltési menü

A „kapcsolat” gomb megnyomásakor az oldal legörget az lábléchez.

Fejlesztői dokumentáció

Alkalmazott fejlesztői és csoportmunka eszközök

A fejlesztéshez használt eszközök a Visual Studio Code Laravel és Angular keretrendszer, illetve XAMPP.

A tervezés során használt wireframe eszköz a Figma, illetve a csoportmunka tervező szoftver a Trello.

Adatbázis

Az adatbázist fejlesztette Mészáros Róbert, a XAMPP phpMyAdmin felületével SQL nyelven. Az SQL nyelvű kód a Visual Studio Code szoftverben került megírásra, majd a phpMyAdmin felületen lett lefuttatva.

Frontend

A frontendet fejlesztette Antal Dominik Zoltán a Visual Studio Code Angular keretrendszerével.

Backend

A backendet fejlesztette Gál Gergő a Visual Studio Code Laravel keretrendszerével.

Adatbázis

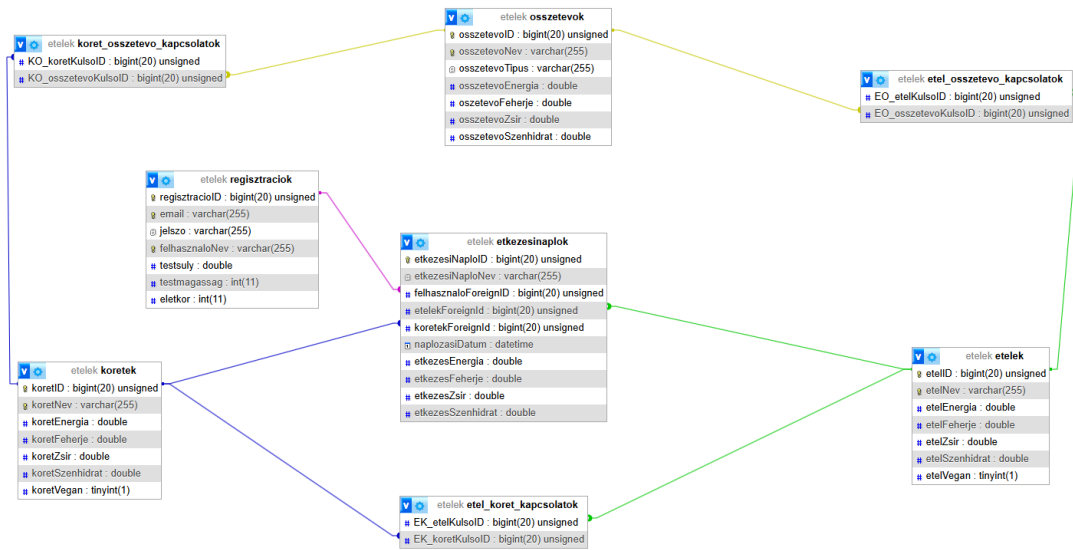
Modell leírása

Adattáblák	Mezők	Leírás
osszetevok	osszetevoID int osszetevoNev string osszetevoTipus string osszetevoEnergia int osszetevoFeherje float osszetevoZsir float osszetevoSzenhidrat float	Összetevő azonosítója Összetevő neve Összetevő típusa (pl. hús) Összetevő kalóriaértéke Összetevő fehérjetartalma Összetevő zsírtartalma Összetevő szénhidráttartalma
koretok	koretID int koretNev string koretEnergia int koretFeherje float koretZsir float koretSzenhidrat float koretVegan boolean	Köret azonosítója Köret neve Köret kalóriaértéke Köret fehérjetartalma Köret zsírtartalma Köret szénhidráttartalma Vegán köret (igen/nem)
etelek	etelID int etelNev string etelEnergia int etelFeherje float etelZsir float etelSzenhidrat float etelVegan boolean	Étel azonosítója Étel neve Étel kalóriaértéke Étel fehérjetartalma Étel zsírtartalma Étel szénhidráttartalma Vegán étel (igen/nem)
regisztraciok	regisztracioID int nev string email string jelszo string felhasznaloNev string testsuly float testmagassag int eletkor int	Regisztráció azonosítója Felhasználó neve Felhasználó e-mail címe Felhasználó jelszava Bejelentkezési felhasználónév Felhasználó testsúlya Felhasználó testmagassága Felhasználó életkor
etkezesinaplo	etkezesiNaploID int etkezesiNaploNev string felhasznaloForeignID int etelForeignID int koretForeignID int, naplozasiDatum date etkezesEnergia double	Naplózás azonosítója Naplózás neve Regisztráció külső azonosítója Étel külső azonosítója Köret külső azonosítója Naplózás dátuma Étkezés kalóriaértéke

	etkezesFeherje double etkezesZsir double etkezesSzenhidrat double	Étkezés fehérjetartalma Étkezés zsírtartalma Étkezés szénhidráttartalma
etel_osszetevo_kapcsolatok	EO_etelKulsoID int, EO_osszetevoKulsoID int	Étel külső azonosítója Összetevő külső azonosítója A tábla az ételeket és az összetevőket kapcsolja össze, hogy több összetevő megadható legyen egy adott ételhez.
koret_osszetevo_kapcsolatok	KO_koretKulsoID int, KO_osszetevoKulsoID int	Köret külső azonosítója Összetevő külső azonosítója A tábla a köreteket és az összetevőket kapcsolja össze, hogy több összetevő megadható legyen egy adott körethez.
etel_koret_kapcsolatok	EK_etelKulsoID int, EK_koretKulsoID int	Étel külső azonosítója Köret külső azonosítója Az ételeket és a köreteket kapcsolja össze, hogy lehessen a felhasználó által kedvükre cserélni az ételek és a köretek kombinációit.

Táblák, kapcsolatok

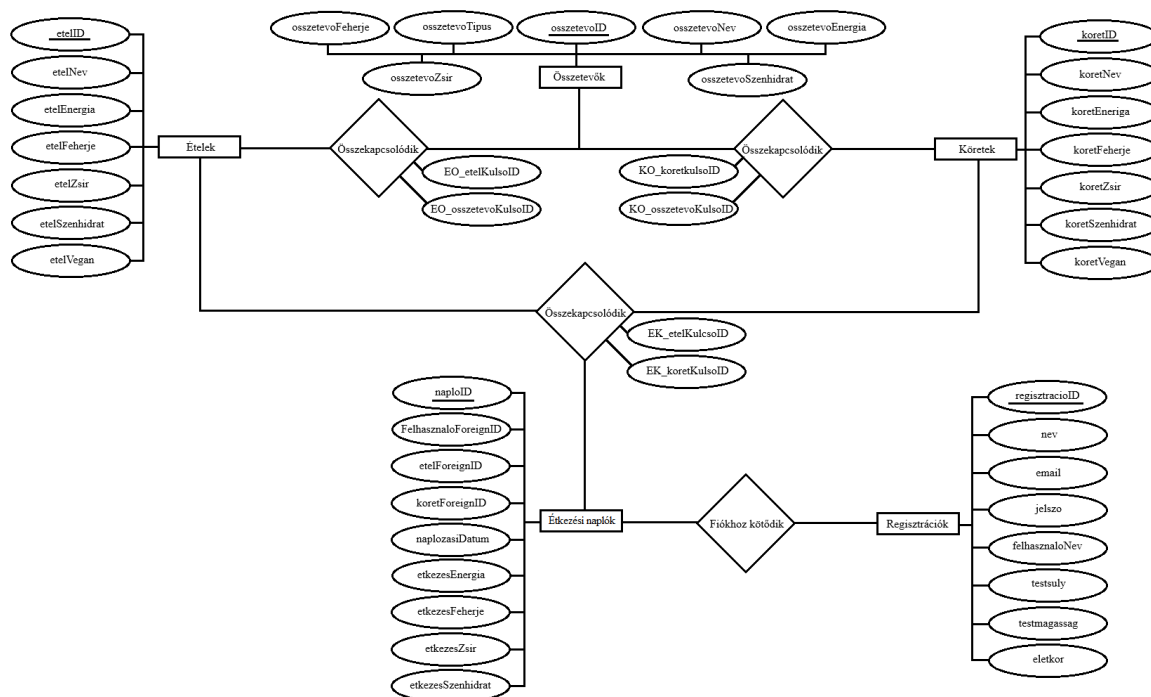
Az adatbázis javarészt az étrendek naplózása köré épült. Ehhez szükséges volt összekötni a naplók tábláját („etkezesiNaplok”) az ételekével („etelek”) és köretekével („koretek”), illetve hogy minden felhasználó a saját naplóit lássa, a felhasználókéval („regisztraciok”) is. Az adatbázis elő lett készítve arra, hogy az összetevőket is tudja kezelni a különböző ételekben, viszont ennek a fejlesztését idő szűkében el kellett engedjünk.



11. kép: adattáblák és kapcsolatai

E-K diagram

Az E-K diagramon az adatbázis adattábláinak logikai kapcsolatai láthatók.



12. kép: az adatbázis E-K diagramja

Frontend

A frontend három modellt (model), egy szolgáltatást (service), és nyolc komponenst (component) használ.

A három modell a „naplozas.ts”, „koret.ts” és „etel.ts”, mindegyiknek az adatbázisban történő adatkezelés megkönnyítése a feladata. Ezekben a modellekben tárolódnak el a lekért, illetve feltölteni kívánt adatok.

```
export interface Naplozas{
  etkezesiNaploID:number;
  felhasznaloForeignID : number;
  etelekForeignId : number;
  koretForeignId : number;
  naplozasiDatum : any;
  etkezesEnergia : number;
  etkezesFeherje : number;
  etkezesZsir : number;
  etkezesSzenhidrat : number;
}
```

13. kép: naplózás modell

```
export interface Koret{
  koretID: number;
  koretNev: string;
  koretEnergia: number;
  koretFeherje: number;
  koretZsir: number;
  koretSzenhidrat: number;
  koretVegan: boolean;
}
```

14. kép: köret modell

```
export interface Etel{
  etelID: number;
  etelNev: string;
  etelEnergia: number;
  etelFeherje: number;
  etelZsir: number;
  etelSzenhidrat: number;
  etelVegan: boolean;
}
```

15. kép: étel modell

Az „api.service.ts” szolgáltatás az, ami kommunikál a backend-el és küldi el a különböző adatokat feldolgozásra, illetve az adatbázisba való feltöltésre.

A `loggedInUser` változó egy signal, ami azt követi, hogy be van-e jelentkezve valaki az oldalra, vagy sem. Alapértelmezetten az értéke hamis.

```
loggedInUser = signal<boolean>(false);
```

16. kép: "loggedInUser" változó

A constructor-ban meg van adva paraméterben a `HttpClient`, hogy az alkalmazás tudjon kommunikálni a backend-el. A constructor törzsében létrehozunk egy „user” változót, ami eltárolja az éppen bejelentkezett felhasználói fiók adatait, és beállítja a `loggedInUser` értékét igazra.

```
constructor(private http: HttpClient) {  
  const user = localStorage.getItem('user');  
  if (user) {  
    this.loggedInUser.set(true);  
  }  
}
```

17. kép: konstruktor

A „login” és „register” függvények bonyolítják le a felhasználó bejelentkezését, illetve regisztrációját. A „register” elküldi a felhasználó paraméterben kapott adatait a backend-be, ahol az feltöltődik az adatbázisba. A „login” elküldi a paraméterben megadott bejelentkezési adatokat, amit a backend feldolgoz, és ha helyesek a megadott adatok, akkor visszaküldi a felhasználó adatait, emellett beállítja a `loggedInUser` értékét igazra.

```
login(user: any): Observable<any> {  
  return this.http.post<any>(`${this.apiUrl}/login`, user).pipe(  
    tap(response => {  
      if (response.success) {  
        this.loggedInUser.set(true);  
      }  
    })  
  );  
}  
  
register(user: any): Observable<any> {  
  return this.http.post<any>(`${this.apiUrl}/regisztraciok`, user);  
}
```

18. kép: a regisztrációért és bejelentkezésért felelős függvények

Az „etelek” és „koretek” függvények lekérlik az ételeket, illetve a köreteket az adatbázisból.

```
etelek(): Observable<Etel[]> {
  return this.http.get<Etel[]>(`${this.apiUrl}/etelek`);
}

koretek(): Observable<Koret[]> {
  return this.http.get<Koret[]>(`${this.apiUrl}/koretek`);
}
```

19. kép: az ételek és köretek lekérésének függvényei

A „naplozasok” és „naplozasoklekeres” függvények a naplózásokat kéri le az adatbázisból, a „naplozasok” csak egyet, paraméterben megadott azonosító (id) alapján, a „naplozasoklekeres” az összes naplózást lekéri.

```
naplozasok(id:number): Observable<Naplozas[]> {
  return this.http.get<Naplozas[]>(`${this.apiUrl}/etkezesiNaplok/${id}`);
}

naplozasoklekeres(id:number): Observable<Naplozas[]> {
  return this.http.get<Naplozas[]>(`${this.apiUrl}/etkezesiNaplok`);
}
```

20. kép: naplót lekérő függvények

Az étkezési naplók frissítésének érdekében van egy „Naplozas” (modellbéli interface) típusú signal, amit visszaad egy függvény.

```
naplok = signal<Naplozas[]>([]);
```

21. kép: „naplok” változó

```
getSharedSignal() {
  return this.naplok;
}
```

22. kép: „getSharedSignal” függvény

Ezt a frissítést a „frissitNaplozasokat” függvény hajtja végre.

```
frissitNaplozasokat(id: number) {
  this.naplozasok(id).subscribe(data => {
    this.naplok.set(data);
  });
}
```

23. kép: „frissitNaplozasokat” metódus

Az új étkezési naplók feltöltéséről a „naploPush” függvény gondoskodik. Paraméterben megkapja az új napló adatait, amit elküld backend-be.

```
naploPush(naplo: any): Observable<any> {
  return this.http.post<any>(`${this.apiUrl}/etkezesiNaplok`, naplo).pipe(
    tap((ujNaplo) => {
      const aktualis = this.naplok();
      const ujLista = aktualis.concat(ujNaplo);
      this.naplok.set(ujLista);
    })
  );
}
```

24. kép: "naploPush" metódus

A naplózások törlését a „naploDelete” metódus végzi. Paraméterben megkapja az adott naplózás azonosítóját (id), majd ezt a törlési kérést elküldi backend-be.

```
naploDelete(id: number): Observable<any> {
  return this.http.delete<any>(`${this.apiUrl}/etkezesiNaplok/${id}`).pipe(
    tap(() => {
      const aktualis = this.naplok();
      const ujLista = aktualis.filter(naplo => naplo.etkezesiNaploID !== id);
      this.naplok.set(ujLista);
    })
  );
}
```

25. kép: "naploDelete" függvény

A felhasználó által feltölteni kívánt ételeket, illetve köreteket az „etelPost” és „koretPost” metódusok kezelik. Paraméterben megkapják az étel, vagy köret adatait, amiket elküld backend-be.

```
etelPost(etel: any): Observable<any> {
  return this.http.post<any>(`${this.apiUrl}/etelek`, etel);
}

koretPost(koret: any): Observable<any> {
  return this.http.post<any>(`${this.apiUrl}/koretek`, koret);
}
```

26. kép: "etelPost" és "koretPost" függvények

A nyolc komponens a „login”, „home”, „about”, „calculator”, „etelpush”, „koretpush”, „log”, és a „notfound”. Minden komponens konstruktorában (constructor) paraméterül van adva a szolgáltatás, hogy képesek legyenek adatot kezelni, illetve legtöbbjükben a Router, hogy a komponensek közötti navigálás is megtörténhessen.

A „login” komponens feladata a felhasználók adatainak kezelése, a regisztráció és bejelentkezés lebonyolítása, ehhez használ egy ideiglenes objektumot „user” néven. Ez az objektum a login.component.html-ből töltődik fel adatokkal „ngModel” segítségével. Példaként az „email” mező felöltése látható.

```
user = {  
  email: '',  
  jelszo: '',  
  felhasznaloNev: '',  
  testsuly: '',  
  testmagassag: '',  
  életkor: ''  
};
```

27. kép: ideiglenes "user" objektum

```
<input  
  type="email"  
  placeholder="E-mail"  
  [(ngModel)]="user.email"  
  name="email"  
  email  
  required  
  #email="ngModel"  
>
```

28. kép: ngModellel történő értékadás az "user" objektumnak

A „registration” metódus hajtja végre a felhasználó adatainak a feltöltését.

```
registration() {  
  this.service.register(this.user).subscribe(  
    response => {  
      console.log('Regisztráció sikeres:', response);  
      alert('Sikeres regisztráció!');  
      this.router.navigate(['login']);  
    },  
    error => {  
      console.error('Regisztráció hiba:', error);  
      alert('Hiba történt a regisztráció során!');  
    }  
  );  
}
```

29. kép: "registration" metódus

A „login” metódus a felhasználó bejelentkezését hajtja végre.

```
login() {  
  this.service.login(this.user).subscribe(  
    response => {  
      console.log('Bejelentkezés sikeres:', response);  
      localStorage.setItem('user', JSON.stringify(response.user));  
      this.service.loggedInUser.set(true);  
      this.router.navigate(['home']);  
    },  
    error => {  
      console.error('Bejelentkezés hiba:', error);  
      alert('Hibás felhasználónév vagy jelszó!');  
    }  
  );  
}
```

30. kép: "login" metódus

A „logout” metódus a felhasználó kijelentkezését hajtja végre.

```
logout() {  
  this.user = {  
    email: '',  
    jelszo: '',  
    felhasznaloNev: '',  
    testsuly: '',  
    testmagassag: '',  
    életkor: ''  
  };  
  localStorage.removeItem('user');  
  this.loggedInUser.set(false);  
  this.router.navigate(['/login']);  
}
```

31. kép: "logout" metódus

A „home” komponens a webalkalmazás főoldala, az alkalmazás bemutatása a célja. Viszont a felhasználó bejelentkezése alapján elérhetővé teszi a „calculator” és „log” komponenseket.

Először lekéri a felhasználó bejelentkezésének állapotát.

```
get isLoggedIn() {  
  return this.service.loggedInUser();  
}
```

32. kép: bejelentkezési állapot lekérés

És ennek megfelelően a helyi tárhelyen (local storage) eltárolja az adatait, vagy kitörli őket onnan.

```
login() {  
  this.loggedIn.set(true);  
  localStorage.setItem('user', JSON.stringify('user'));  
}  
  
logout() {  
  this.loggedIn.set(false);  
  localStorage.removeItem('user');  
}
```

33. kép: "login" és "logout" metódusok

Ha a felhasználó be van jelentkezve, akkor elérheti a kalkulátort, ha nincs, akkor viszont a „login” komponensre navigálja a függvény.

```
openModal(event: Event) {
  document.body.style.overflow = 'hidden';
  event.preventDefault();
  if (this.isLoggedIn) {
    const dialogRef = this.dialog.open(CalculatorComponent, {
      width: '900px',
      height: '700px'
    });

    dialogRef.afterClosed().subscribe(() => {

      const user = localStorage.getItem('user');
      if (user) {
        const userjson = JSON.parse(user);
        const userid = userjson.regisztracioID;
        this.service.frissitNaplozatokat(userid);
      }
    });
  } else {
    alert("Bejelentkezés szükséges a kalkulátorhoz!");
    this.router.navigate(['/login']);
  }
}
```

34. kép: a kalkulátort megnyitó "openModal" metódus

Az „about” komponens szinte megegyezik a „home” komponenssel, csak az alkalmazás szolgáltatásai helyett a fejlesztő csapatot mutatja be a komponens. A háttérben ugyan azok a funkciók érhetőek el, ugyan azon a módon, mint a „home” komponens esetében.

A „calculator” komponensben történik meg a különböző étkezések rögzítése.

Ehhez az adatbázisból lekérjük az ételeket és köreteket a „getEtelek” és „getKoretek” metódusokkal, és eltároljuk „Etel” és „Koret” (modellbéli interface) típusú tömbökbe.

```
etel: Etel[] = [];  
koret: Koret[] = [];
```

35. kép: az ételeket és köreteket tároló tömbök

```
getEtelek() {  
  this.service.etelek().subscribe(data => {  
    this.etel = data;  
  });  
}  
  
getKoretek() {  
  this.service.koretek().subscribe(data => {  
    this.koret = data;  
  });  
}
```

36. kép: az „etel” és „koret” tömböket feltöltő „getEtelek” és „getKoretek” metódusok

Létrehozunk egy „Naplozas” (modellbéli interface) típusú ideiglenes objektumot, amit aztán majd a „probaEgy”, és „probaKetto” metódusokkal feltöltünk a felhasználó által kiválasztott étel, vagy köret adataival, illetve a további fontos adatokkal, hogy feltölthető lehessen az adatbázisba.

```
naplo : Naplozas = {  
  etkezesiNaploID:0,  
  felhasznaloForeignID : 0,  
  etelekForeignId : 0,  
  koretekForeignId : 0,  
  naplozasDatum : '',  
  etkezesEnergia : 0,  
  etkezesFeherje : 0,  
  etkezesZsir : 0,  
  etkezesSzenhidrat : 0  
};
```

37. kép: „naplo” nevű ideiglenes objektum

```

user = localStorage.getItem('user');
probaEgy(etelID: number) {

    const mennyiseg = this.etelMennyiseg[etelID] || 100;

    if (this.user) {
        const userjson = JSON.parse(this.user);
        this.naplo.felhasznaloForeignID = userjson.regisztracioID;
    }

    this.naplo.etelekForeignId = this.etel[etelID - 1].etelID;
    this.naplo.koretetekForeignId = this.koret[0].koretID;
    this.naplo.naplozasDatum = this.datePipe.transform(new Date, 'YYYY-MM-dd hh:mm:ss');
    this.naplo.etkezesEnergia = (this.etel[etelID - 1].etelEnergia/100)*mennyiseg;
    this.naplo.etkezesFeherje = (this.etel[etelID - 1].etelFeherje/100)*mennyiseg;
    this.naplo.etkezesZsir = (this.etel[etelID - 1].etelZsir/100)*mennyiseg;
    this.naplo.etkezesSzenhidrat = (this.etel[etelID - 1].etelSzenhidrat/100)*mennyiseg;
    this.service.naploPush(this.naplo).subscribe();
}

```

38. kép: a "naplo" feltöltését végző "probaEgy" metódus

```

probaKetto(koretID: number) {

    const mennyiseg = this.koretMennyiseg[koretID] || 100;

    if (this.user) {
        const userjson = JSON.parse(this.user);
        this.naplo.felhasznaloForeignID = userjson.regisztracioID;
    }

    this.naplo.etelekForeignId = this.etel[0].etelID;
    this.naplo.koretetekForeignId = this.koret[koretID-1].koretID;
    this.naplo.naplozasDatum = this.datePipe.transform(new Date, 'YYYY-MM-dd hh:mm:ss');
    this.naplo.etkezesEnergia = (this.koret[koretID - 1].koretEnergia/100)*mennyiseg;
    this.naplo.etkezesFeherje = (this.koret[koretID - 1].koretFeherje/100)*mennyiseg;
    this.naplo.etkezesZsir = (this.koret[koretID - 1].koretZsir/100)*mennyiseg;
    this.naplo.etkezesSzenhidrat = (this.koret[koretID - 1].koretSzenhidrat/100)*mennyiseg;

    this.service.naploPush(this.naplo).subscribe();
}

```

39. kép: a "naplo" feltöltését végző "probaKetto" metódus

A „filteredEtel” és „filteredKoret” metódusok az ételek és köretek név szerinti keresésében segítenek. Ehhez van egy segéd változó, a „searchText”, ami a felhasználó által beírt szöveget tárolja.

```
searchText: string = '';
```

40. kép: "searchText" változó

```
filteredEtel() {  
  return this.etel.filter(e => e.etelNev !== 'Üres' && e.etelNev.toLowerCase().includes(this.searchText.toLowerCase()));  
}  
  
filteredKoret() {  
  return this.koret.filter(e => e.koretNev !== 'Üres' && e.koretNev.toLowerCase().includes(this.searchText.toLowerCase()));  
}
```

41. kép: "filteredEtel" és "filteredKoret" függvények

A „restoreScroll” metódus visszagörget a komponens legtetejére automatikusan, így ha korábban használtuk, nem úgy jelenik meg, ahogy hagytuk.

```
private restoreScroll() {  
  setTimeout(() => {  
    [document.documentElement, document.body].forEach(el => {  
      el.style.overflow = 'auto';  
      el.classList.remove('cdk-global-scrollblock');  
    });  
  }, 0);  
}
```

42. kép: "restoreScroll" metódus

Az „etelPush” és „koretPush” komponensek feladata szimplán új ételek, illetve köretek hozzáadása az adatbázishoz. Mindkét komponens ugyanúgy működik, bemutatni az „etelPush” komponenssel fogjuk.

A komponensben van egy ideiglenes objektum, amit a felhasználó által megadott adatokkal feltöltünk, majd az az „onSubmit” metódussal elküldünk a backend-be.

```
etel = {  
  etelNev: '',  
  etelEnergia: null,  
  etelFeherje: null,  
  etelZsir: null,  
  etelSzenhidrat: null  
};
```

43. kép: "etel" ideiglenes objektum

```

onSubmit() {
  this.service.etelPost(this.etel).subscribe(
    response => {
      console.log('Sikeresen feltöltve!', response);
      alert('Sikeresen feltöltve!');
    },
    error => {
      console.error('Hiba történt a feltöltés során!', error);
      alert('Hiba történt a feltöltés során!');
    }
  );
  console.log(this.etel)
}

```

44. kép: "onSubmit" metódus

A „log” komponensben lehet a naplózásokat megtekinteni, és törölni, illetve látható a napi bevitt étrendünk tápértéke a napi ajánlotthoz képest.

Itt is lekérjük az előző komponensekhez hasonlóan az ételeket és köreteket tömbökbe, illetve a naplózásokat jelként (signal) megkapjuk a szolgáltatásból.

```

naplok!: WritableSignal<Naplozas[]>;

```

45. kép: "naplok" tömb

```

this.naplok = this.service.getSharedSignal();

```

46. kép: "naplok" értékadása

Ezt a tömböt folyamatosan frissíti az oldal, ha változás történik benne, illetve szűri felhasználók szerint, hogy a bejelentkezett felhasználó csak a saját naplózásait lássa.

```

const user = localStorage.getItem('user');
if (user) {
  const userjson = JSON.parse(user);
  const userid = userjson.regisztracioID;
  this.service.frissitNaplozatokat(userid);
}

```

47. kép: felhasználók szerinti szűrés

A „getKaloria”, „getFeherje”, „getSzenhidrat” és „getZsir” függvények a napi ajánlott tápanyagmennyiséget számolják ki a felhasználó megadott adatai alapján, amik a naplók alatt megjelenő folyamatjelző sáv (progress bar) maximális értékei lesznek.

```

getKaloria() {
  const user = localStorage.getItem('user');
  if (user)
  {
    const userjson = JSON.parse(user);
    const osszesKaloria = userjson.testsuly * 24;
    return osszesKaloria;
  }
  else {
    return 2000;
  }
}

```

48. kép: "getKaloria" függvény

```

getFeherje() {
  const user = localStorage.getItem('user');
  if (user)
  {
    const userjson = JSON.parse(user);
    const osszesFeherje = userjson.testsuly * 0.8;
    return osszesFeherje;
  }
  else {
    return 150;
  }
}

```

49. kép: "getFeherje" függvény

```

getSzenhidrat() {
  const user = localStorage.getItem('user');
  if (user)
  {
    const userjson = JSON.parse(user);
    const osszesSzenhidrat = userjson.testsuly * 4;
    return osszesSzenhidrat;
  }
  else {
    return 300;
  }
}

```

50. kép: "getSzenhidrat" függvény

```

getZsir() {
  const kaloria = this.getKaloria();
  const user = localStorage.getItem('user');
  if (user)
  {
    const osszesZsir = kaloria / 9.3;
    return osszesZsir;
  }
  else {
    return 100;
  }
}

```

51. kép: "getZsir" függvény

Az aktuális bevitt tápértékeket a „totalKaloria”, „totalFeherje”, „totalSzenhidrat” és „totalZsir” változók tárolják és számolják.

```

totalKaloria = computed(() => {
  return this.naplok().reduce((sum, naplo) => sum + naplo.etkezesEnergia, 0);
});

totalFeherje = computed(() => {
  return this.naplok().reduce((sum, naplo) => sum + naplo.etkezesFeherje, 0);
});

totalSzenhidrat = computed(() => {
  return this.naplok().reduce((sum, naplo) => sum + naplo.etkezesSzenhidrat, 0);
});

totalZsir = computed(() => {
  return this.naplok().reduce((sum, naplo) => sum + naplo.etkezesZsir, 0);
});

```

52. kép: Az aktuális tápértéket számontartó változók.

Az étkezési naplók nevének kiírása az „etelNevKiiratas” és „koretNevKiiratas” függvényekkel történik.

```
etelNevKiiratas(beID : number) {  
  var nev = "";  
  for (let i = 0; i < this.etel.length; i++)  
  {  
    if (beID === this.etel[i].etelID)  
    {  
      nev = this.etel[i].etelNev;  
    }  
  }  
  if (nev === "Üres")  
  {  
    nev = "";  
  }  
  return nev;  
}
```

53. kép: "etelNevKiiratas" függvény

```
koretNevKiiratas(beID : number) {  
  var nev = "";  
  for (let i = 0; i < this.koret.length; i++)  
  {  
    if (beID === this.koret[i].koretID)  
    {  
      nev = this.koret[i].koretNev;  
    }  
  }  
  if (nev === "Üres")  
  {  
    nev = "";  
  }  
  return nev;  
}
```

54. kép: "koretNevKiiratas" függvény

A „notfound” komponens feladata mindössze, hogy jelezze a felhasználó számára, ha a komponens, amit megpróbált elérni nem létezik, ez egy egyszerű címsorban történik meg.

```
<h1>404 A keresett komponens nem található!</h1>
```

55. kép: a "notfound" komponens címsora

Ez a komponens csak akkor érhető el, ha manuálisan átírjuk az oldal URL-jét egy olyanra, ami nincs használatban.

Backend

A backendben öt controller található: „EtelController”, „KoretController”, „OsszesitesController”, „OsszetevoController”, és „RegisztracioController”.

Mind az öt controller ugyan azzal a hat alap metódussal van ellátva, csak mindegyik a saját mezőit ellenőrzi, kezeli. A bemutatáshoz az EtelController metódusait hozzuk fel példának.

Az „index” metódus egy adatlekérést hajt végre az adatbázisból, ami az összes étel összes adatát visszaadja.

```
public function index()
{
    return Etel::all();
}
```

56. kép: "index" metódus

A „store” metódus paraméterben megadott adatot tölt fel az adatbázisba.

A „Validator” segítségével ellenőrizzük, hogy minden kötelező adat megtalálható-e a paraméterben található kérésben. Amennyiben valamelyik hiányzik, egy hibaüzenetet kapunk eredményül, ellenkező esetben viszont feltölti az adatokat az adatbázisba, és visszaadja a feltöltött rekord azonosítóját.

```
public function store(Request $request)
{
    $validator = Validator::make($request->all(),
    [
        'etelNev'=>'required',
        'etelEnergia'=>'required',
        'etelFeherje'=>'required',
        'etelZsir'=>'required',
        'etelSzenhidrat'=>'required',
        'etelVegan'=>'required'
    ]);

    if($validator->fails())
    {
        return response()->json(['message' => 'Kötelező adat hiányzik'], 400);
    }

    $etel = Etel::create($request->all());
    return response()->json(['id' => $etel->etelID],202);
}
```

57. kép: "store" metódus

A „getById” metódus is egy adatlekérést hajt végre, viszont ez az adott adat azonosító mezőjét ellenőrzi, és csak azt az egy rekordot adja vissza, aminek az azonosítójával a paraméterben keresett azonosító megegyezik. Ha nincs egyező azonosítójú rekord, hibaüzenetet ad vissza.

```
public function getById(int $id)
{
    $etel = Etel::find($id);
    if(is_null($etel))
    {
        return response()->json(['Hiba:'=>'Étel nem található!'],404);
    }

    return response()->json($etel,201);
}
```

58. kép: "getById" metódus

A „destroy” metódus rekordot töröl az adatbázisból. Paraméterben megkap egy azonosítót, és ha talál egy vele azonos azonosítójú rekordot, törli azt az adatbázisból. Ha nem talál, hibaüzenetet ad vissza.

```
public function destroy(int $id)
{
    $etel = Etel::find($id);

    if (!$etel) {
        return response()->json(['message' => 'Az étel nem létezik!'], 404);
    }

    $etel->delete();
    return response()->json(['message'=> 'Sikeres törlés!'], 204);
}
```

59. kép: "destroy" metódus

Az „update” metódus megváltoztatja az adatait egy már meglévő rekordnak. Paraméterben megkapja az új adatokat, illetve a keresett azonosítót. Először megkeresi az azonosítót a rekordok között, ha nem találja meg, akkor hibaüzenetet ad vissza. Ha viszont megtalálja, a „Validatorral” ellenőrzi, hogy minden kötelező adat megvan-e az új adathalmazban. Ha nincs üresen hagyott kötelező mező, frissíti a rekordot az új adatokkal. Ha nincs meg minden kötelező adat, hibaüzenetet ad vissza.

```
public function update(Request $request, $id)
{
    $etel = Etel::find($id);
    if(is_null($etel))
        return response()->json(['Üzenet:'=>'Nem létezik az étel!'],404);

    $validator = Validator::make($request->all(),
    [
        'etelNev'=>'required',
        'etelEnergia'=>'required',
        'etelFeherje'=>'required',
        'etelZsir'=>'required',
        'etelSzenhidrat'=>'required',
        'etelVegan'=>'required'
    ]
    );
    if($validator->fails()){
        return response()->json(['message' => 'kötelező adatok hiányoznak'],402);
    }

    $etel -> update($request->all());
    return response()->json(['Következő id-jű ételek adatai módosultak' => $etel->etelID],201);
}
```

60. kép: "update" metódus

A „searchbyname” metódus név alapján keresi meg az adott rekordot. Paraméterben egy szöveget kap, és ha az adott rekord név mezőjével egyezik a szöveg, visszaadja azt a rekordot. Ha a kettő nem egyezik, hibaüzenetet ad vissza.

```
public function searchbyname(string $etelNev){
    $etel=Etel::where('etelNev',$etelNev)->first();

    if(is_null($etel)){
        return response()->json(['message'=>'Etel nem található'],404);
    }
    return response()->json($etel,200);
}
```

61. kép: "searchbyname" metódus

A fent említett hat metóduson kívül a „RegisztracioController” el lett látva egy hetedik, „login” metódussal, ami a webalkalmazásra való bejelentkezést teszi lehetővé. Paraméterben megkapja a bejelentkezni kívánó felhasználó felhasználónevét és jelszavát, majd a felhasználó alapján megkeresi az adott felhasználó adatait. Ellenőrzi, hogy a jelszavak egyeznek-e az adatbázisban szereplő jelszóval, és ha igen, akkor visszaadja a felhasználó adatait. Ha nem, akkor hibaüzenetet ad vissza.

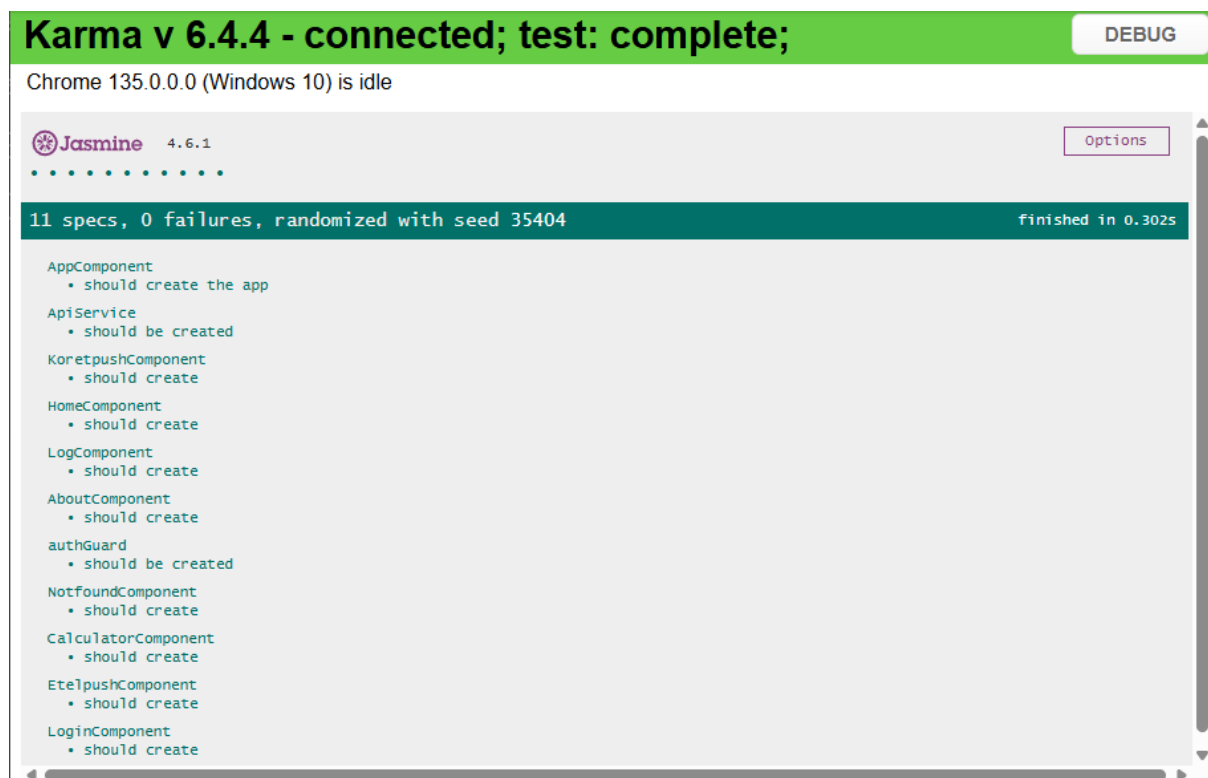
```
public function login(Request $request)
{
    $adatok = $request->only('felhasznaloNev', 'jelszo');
    $user = Regisztracio::where('felhasznaloNev', $adatok['felhasznaloNev'])->first();
    if ($user && $user->jelszo === $adatok['jelszo']) {
        return response()->json(['user' => $user], 200);
    }
    return response()->json(['error' => 'Rossz adatok'], 401);
}
```

62. kép: "login" metódus

Tesztelés

Frontend tesztelése

A frontend tesztelése az Angularba épített „ng test” paranccsal elérhető Karma, Jasmine tesztfelülettel, illetve manuálisan, böngészőben történt.



63. kép: frontend Jasmine teszt

Ha regisztráció közben valamelyik adatbeviteli mezőbe belekattintunk, viszont nem adunk meg adatot és kikattintunk belőle, vagy hibás adatot adunk meg, az alkalmazás jelzi a felhasználó számára. Ha hibás vagy hiányzó adatokkal akarunk regisztrálni, az oldal közli velünk felugró ablakban.



The image shows a registration form titled "Hozz létre fiókot" (Create account). On the left, a large teal bubble contains the text "Üdvözlünk ismét!" (Welcome back!) and "Írd be a személyes adataid az oldal funkcióinak használatához!" (Enter your personal data to use the site's features!), with a "BELÉPEK" (Log in) button. The right side contains input fields for "Email", "Jelszó" (Password), and "Felhasználónév" (Username), each with a red error message. Below these are three boxes for "Magasság" (Height) with values -5, 15, and 8, each with a red error message. A "REGISZTRÁCIÓ" (Register) button is at the bottom right.

Hozz létre fiókot

helytelenEmailcim

E-mail megadása kötelező!

...

A jelszónak legalább 6 karakter hosszúnak kell lennie!

Felhasználónév

Felhasználónév megadása kötelező!

-5 15 8

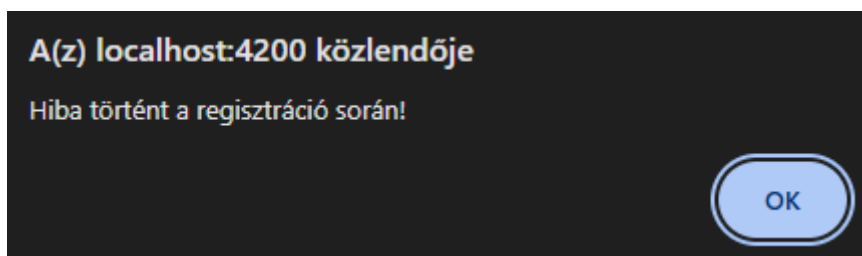
A testűlynak pozitív számnak kell lennie!

A magasságnak 50 cm-nél nagyobb-nak kell lennie!

Az életkornak legalább 10-nek kell lennie!

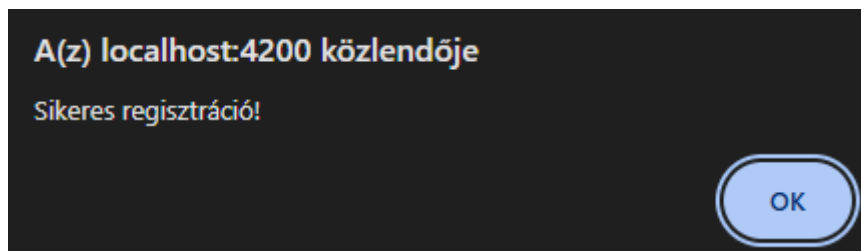
REGISZTRÁCIÓ

64. kép: regisztrációs hibák



65. kép: regisztrációs hibaüzenet

Ha helyes adatokat adtunk meg, akkor a regisztráció sikeresen megtörténik, és ezt az oldal közli is velünk felugró ablakban.



66. kép: sikeres regisztrációs üzenet

A regisztrációhoz hasonlóan a bejelentkezésnél is hibát jelez az oldal, ha hiányoznak az adatok.

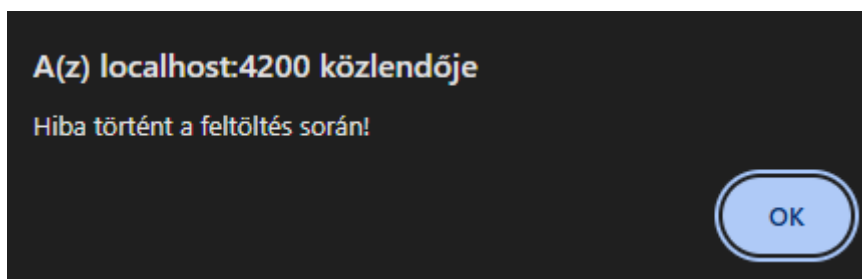
A login page titled "Jelentkezz be!". It features two input fields: "Felhasználónév" (Username) and "Jelszó" (Password), both with red error messages below them. The error messages are "Felhasználónév megadása kötelező!" (Username is required!) and "A jelszónak legalább 6 karakter hosszúnak kell lennie!" (The password must be at least 6 characters long!). Below the fields is a "Bejelentkezés vendégként" (Log in as guest) link and a "BELÉPÉS" (Log in) button. To the right, there is a large teal section with the text "Helló, látogató!" (Hello, visitor!), "Regisztrálj fiókot az oldal funkcióinak használatához!" (Register an account to use the site's features!), and a "REGISZTRÁLOK" (Register) button.

67. kép: bejelentkezési hibák

A kalkulátorban, ha a felhasználó új ételt vagy köretet akar hozzáadni az adatbázishoz, de valamelyik adatmezőt nem tölti ki, a rendszer jelzi, hogy kötelező megadni az adatokat. Ha kötelező adat megadása nélkül szeretné feltölteni az ételt vagy a köretet, az oldal közli vele. A bemutatásra az ételek feltöltésimenüjét hoztuk példának.

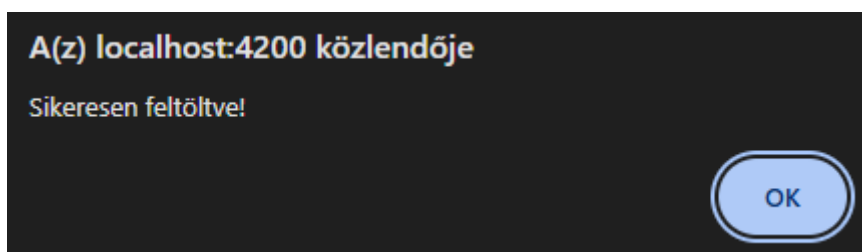
A screenshot of a web application interface for adding food data. The background shows a tray with food items like fries and a burger. Overlaid on this is a form titled "Add meg az étel adatait:". The form contains six input fields, each with a red asterisk indicating it is required. The fields are labeled: "Étel Neve*", "Energia*", "Fehérje*", "Zsír*", and "Szénhidrát*", each followed by the text "Kötelező mező". At the bottom of the form is a green button labeled "Mentés".

68. kép: étel feltöltési hibás mezők



69. kép: étel feltöltési hibaüzenet

Ha a felhasználó megadta az összes adatot, sikeresen hozzáadhatja ezt az adatbázishoz, amit az oldal jelez felugró ablakban.



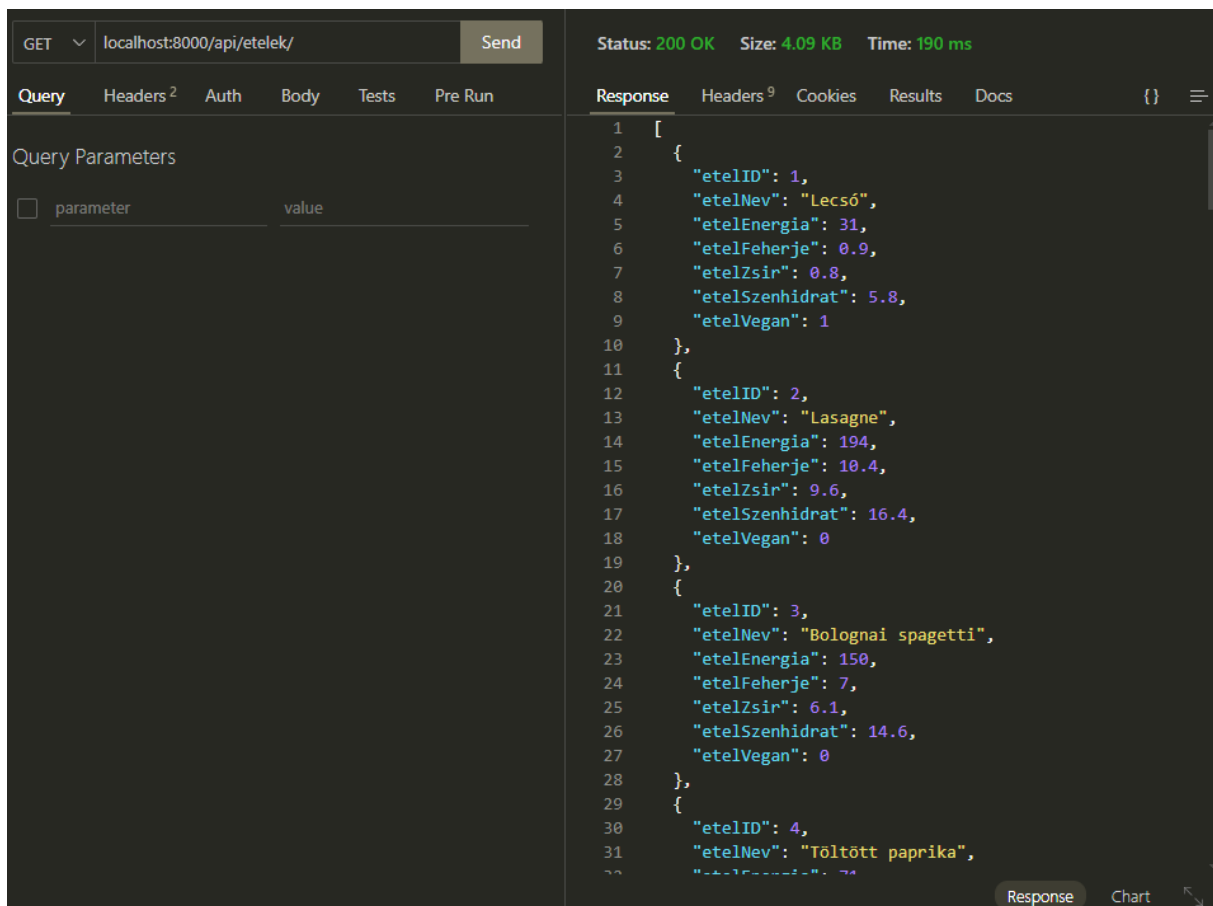
70. kép: sikeres ételfeltöltési üzenet

Backend tesztelése

A backend tesztelése a Thunder Client, ingyenes API Request kezelőjével történt.

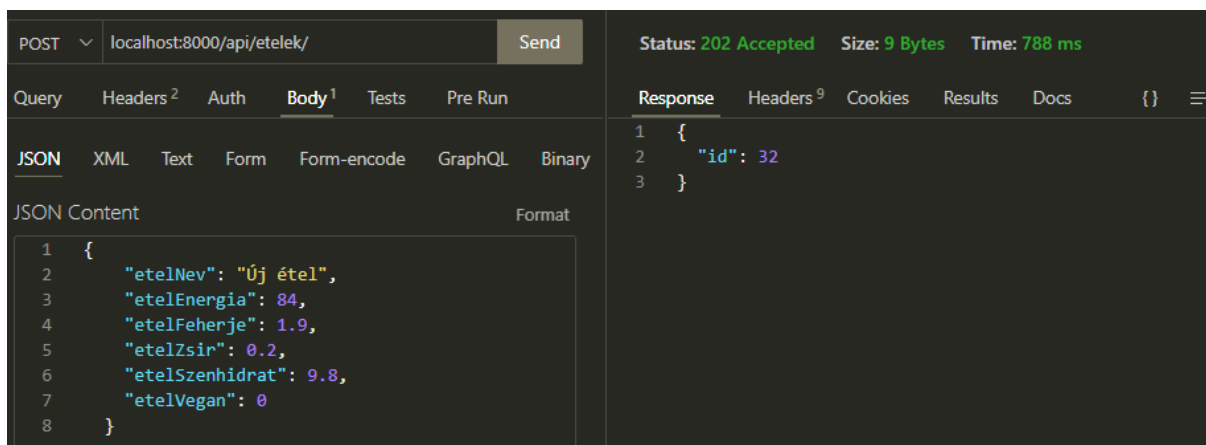
Mind az öt „controller” metódusainak tesztelése ugyanúgy zajlik, csak a saját beállított API végpontját kell megadni URL-ként. A bemutatáshoz az „EtelController” metódusait, és azoknak beállított API végpontjait hozzuk példának.

Az „index” metódus válasza:



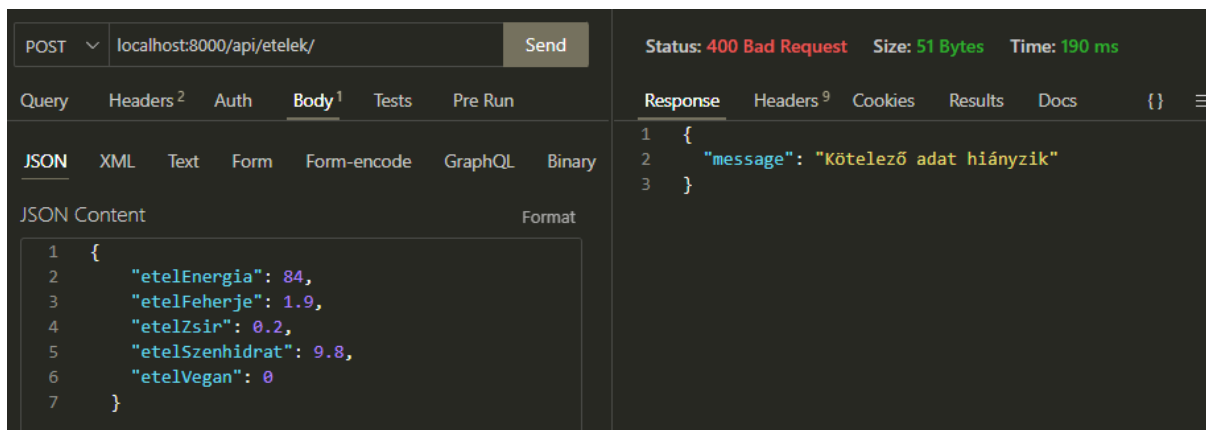
71. kép: "index" válasz

A „store” metódus válasza sikeres feltöltés esetén:



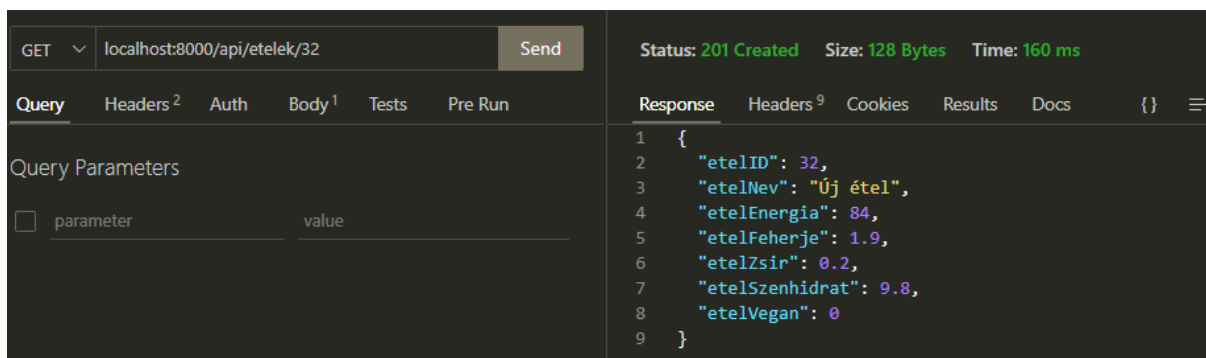
72. kép: sikeres "store" válasz

A „store” metódus válasza sikertelen feltöltés esetén:



73. kép: sikertelen "store" válasz

A „getById” metódus válasza sikeres lekérdezés esetén:



74. kép: sikeres "getById" válasz

A „getById” metódus válasza sikertelen lekérdezés esetén:

GETlocalhost:8000/api/etelek/33Send

Query

Headers²

Auth

Body¹

Tests

Pre Run

Query Parameters

Status: 404 Not FoundSize: 46 BytesTime: 143 ms

Response

Headers⁹

Cookies

Results

Docs

{}

≡

1 {

2 "Hiba:": "Étel nem található!"

3 }

75. kép: sikertelen "getById" válasz

A „destroy” metódus válasza sikeres törlés esetén:

DELETE	localhost:8000/api/etelek/32	Send	Status: 204 No Content	Size: 0 Bytes	Time: 162 ms
Query	Headers ²	Auth	Body ¹	Tests	Pre Run
Response	Headers ⁸	Cookies	Results	Docs	{}
<pre>1</pre>					

76. kép: sikeres "destroy" válasz

A „destory” metódus válasza sikertelen törlés esetén:

<div>DELETE localhost:8000/api/etelek/33 Send</div>					<div>Status: 404 Not Found Size: 44 Bytes Time: 178 ms</div>				
<div>Query Headers² Auth Body¹ Tests Pre Run</div>					<div>Response Headers⁹ Cookies Results Docs {} ≡</div>				
<div>Query Parameters</div>					<div><pre>1 { 2 "message": "Az étel nem létezik!" 3 }</pre></div>				

77. kép: sikertelen "destroy" válasz

Az „update” metódus válasza sikeres frissítés esetén:

PUTlocalhost:8000/api/etelek/33Send

Query

Headers²

Auth

Body¹

Tests

Pre Run

JSON

XML

Text

Form

Form-encode

GraphQL

Binary

JSON ContentFormat

1 {

2 "etelNev": "Legújabb étel",

3 "etelEnergia": 84,

4 "etelFeherje": 1.9,

5 "etelZsir": 0.2,

6 "etelSzenhidrat": 9.8,

7 "etelVegan": 0

8 }

Status: 201 CreatedSize: 72 BytesTime: 198 ms

Response

Headers⁹

Cookies

Results

Docs

{}

≡

1 {

2 "Következő id-jű ételek adatai módosultak": 33

3 }

78. kép: sikeres "update" válasz

Az „update” metódus válasza, ha valamelyik fontos adat hiányzik:

PUT localhost:8000/api/etelek/33 Send

Status: 402 Payment Required Size: 55 Bytes Time: 173 ms

Query Headers² Auth Body¹ Tests Pre Run

JSON XML Text Form Form-encode GraphQL Binary

JSON Content Format

```
1 {
2   "etelEnergia": 84,
3   "etelFehérje": 1.9,
4   "etelZsír": 0.2,
5   "etelSzenhidrat": 9.8,
6   "etelVegan": 0
7 }
```

Response Headers⁹ Cookies Results Docs {} ≡

```
1 {
2   "message": "kötelező adatok hiányoznak"
3 }
```

79. kép: sikertelen "update" válasz (hiányzó adatok)

Az „update” metódus válasza, ha rossz azonosítót kap, mint frissítendő rekordot:

PUT localhost:8000/api/etelek/34 Send

Status: 404 Not Found Size: 49 Bytes Time: 140 ms

Query Headers² Auth Body¹ Tests Pre Run

JSON XML Text Form Form-encode GraphQL Binary

JSON Content Format

```
1 {
2   "etelEnergia": 84,
3   "etelFehérje": 1.9,
4   "etelZsír": 0.2,
5   "etelSzenhidrat": 9.8,
6   "etelVegan": 0
7 }
```

Response Headers⁹ Cookies Results Docs {} ≡

```
1 {
2   "Üzenet": "Nem létezik az étel!"
3 }
```

80. kép: sikertelen "update" válasz (azonosító hiba)

A „searchbyname” metódus válasza sikeres adatkérés esetén:

GET localhost:8000/api/etelkajanev/Lasagne Send

Status: 200 OK Size: 120 Bytes Time: 131 ms

Query Headers² Auth Body Tests Pre Run

Query Parameters

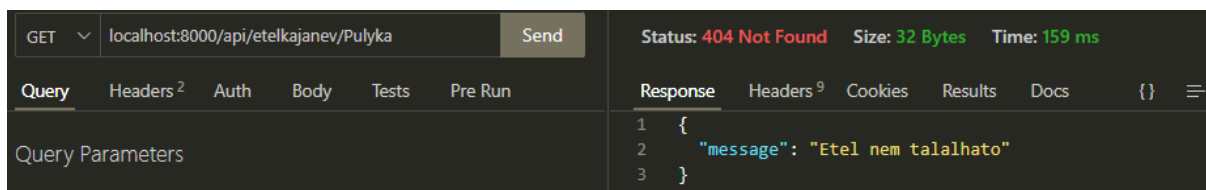
☐ parameter value

Response Headers⁹ Cookies Results Docs {} ≡

```
1 {
2   "etelID": 2,
3   "etelNev": "Lasagne",
4   "etelEnergia": 194,
5   "etelFehérje": 10.4,
6   "etelZsír": 9.6,
7   "etelSzenhidrat": 16.4,
8   "etelVegan": 0
9 }
```

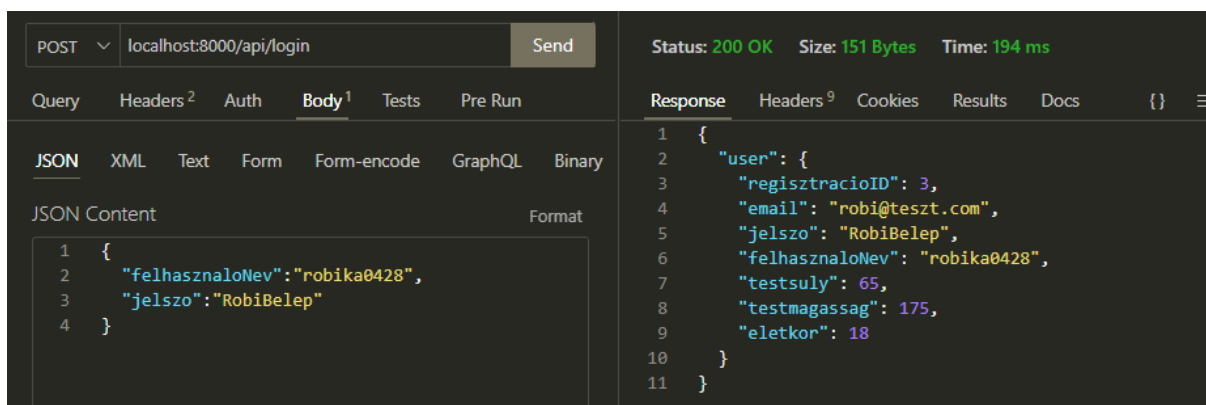
81. kép: sikeres "searchbyname" válasz

A „searchbyname” metódus válasza sikertelen adatkérés esetén:



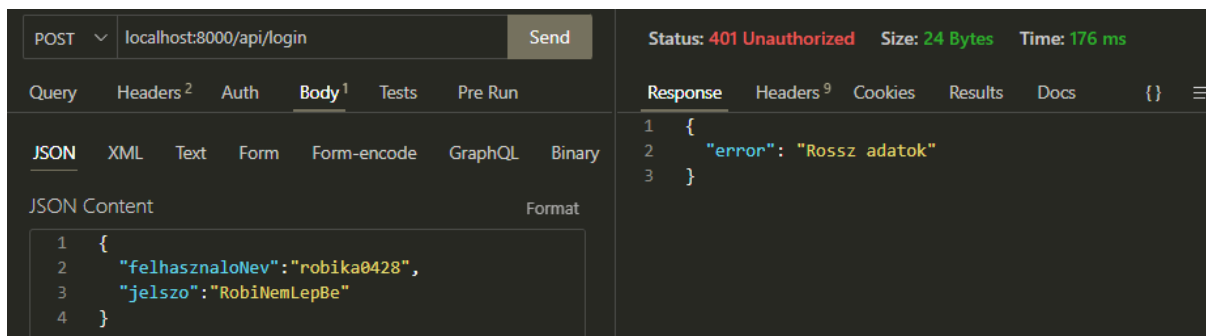
82. kép: sikertelen "searchbyname" válasz

A „login” metódus kizárólag a „RegisztracioController” része. A válasza sikeres bejelentkezés esetén:



83. kép: sikeres "login" válasz

A „login” metódus válasza sikertelen bejelentkezés esetén:



84. kép: sikertelen "login" válasz

Továbbfejlesztési lehetőségek

A weboldal ajánlhatna a felhasználóknak ételeket a naplóknál, hogy a felhasználónak egyszerűbb legyen egy egészséges étrendet választania.

Összetevők szerint lehetne összeállítani az ételeket, illetve köreteket, így a felhasználó, ha allergiás valamire, vagy nem szeret valamit, ami az adott ételben lenne, azt kivehetné, és a program ahhoz mérten számolná a tápértékeket.

Kétlépcsős azonosítást (2FA) implementálni a felhasználók e-mail címével, hogy biztonságosabb lehessen az alkalmazás, és ne történhessen adatlopás.

Irodalomjegyzék

<https://stackoverflow.com/questions/40003575/angular-2-error-no-provider-for-http-in-karma-jasmine-test>