

TestreSzabva

Étkezéstervező- és segítő alkalmazás

Készítette:

Szima Ákos, Suták Dávid Tamás

Tartalomjegyzék:

1. [Bevezetés](#)
2. [Fejlesztői dokumentáció](#)
 - 2.1 [Adatbázis felépítése](#)
 - 2.2 [Tipikus algoritmusok](#)
 - 2.3 [Tesztelési dokumentáció](#)
 - 2.4 [További fejlesztési lehetőségek](#)
3. [Felhasználói dokumentáció](#)
 - 3.1 [Telepítés és indítás](#)
 - 3.2 [A program részletes használata](#)
 - 3.3 [Hibajelzések magyarázata](#)
 - 3.4 [Információkérés és támogatás](#)
4. [Köszönetnyilvánítás](#)

1.) Bevezetés

A TestreSzabva projekt célja, hogy megkönnyítse az egészséges és fenntartható életmód elérését azok számára, akik szeretnék rendszeresen és tudatosan követni étrendjüket. A szoftverünk olyan innovatív megoldást kínál, amely heti bontásban segíti elő a felhasználók étrendjének tervezését, lehetőséget adva a teljes személyre szabhatóságra és a rugalmas beállításokra.

A témaválasztásunk mögött személyes tapasztalataink és a piacon elérhető megoldások hiányosságai állnak. Megfigyeltük, hogy a jelenlegi étrendkövető programok többsége csupán napi szinten támogatja a felhasználót, ami hosszú távon nem teszi lehetővé a kitartó, egészséges életmód kialakítását. Éppen ezért döntöttünk úgy, hogy egy olyan rendszert fejlesztünk, mely nemcsak az azonnali kalória- és étkezési adatokat veszi figyelembe, hanem átfogó, heti tervezést és folyamatos önellenőrzést is biztosít.

A szoftverünk többek között a következő funkciókkal rendelkezik:

- **Heti tervezés:** Lehetővé teszi az étrend heti szintű lebontását, így a felhasználó előre tudja ütemezni étkezéseit és étlapját.
- **Személyre szabott beállítások:** A felhasználók egyéni igényeihez és céljaihoz igazodó étrendi ajánlások kerülnek kidolgozásra.
- **Részletes statisztikák és önellenőrzés:** Az alkalmazás integrált önellenőrző listával dolgozik, melynek segítségével a dokumentáció és a felhasználói felület minősége folyamatosan mérhető, százalékos értékeléssel és körülbelüli osztályzattal.

A célközönségünk elsősorban olyan egyénekből áll, akik elkötelezettek az egészséges életmód és testsúly menedzsmentje iránt. Legyen szó kezdőkről, akik most ismerkednek az egészséges táplálkozás alapjaival, vagy tapasztalt sportolókról, akik részletes és hosszútávú támogatást keresnek, a TestreSzabva mindenki számára kínál hasznos eszközöket.

Ezzel a megközelítéssel szeretnénk nem csupán egy egyszerű étrendkövető alkalmazást bemutatni, hanem egy komplex, felhasználóbarát rendszert, amely támogatja a tudatos életmód kialakítását, és hozzájárul a hosszú távú egészségmegőrzéshez.

2.) Fejlesztői Dokumentáció

Motiváció és egyediség

A TestreSzabva projekt azért született, mert a meglévő érendkövető alkalmazások többsége kizárólag napi szinten segíti a felhasználót, így nem biztosít hosszútávon fenntartható, személyre szabott megoldást. A projektünk a heti bontású tervezés és a folyamatos önellenőrzés révén nem csupán egyszeri, hanem tartós, életmódváltást támogató alkalmazás, amely a felhasználói igényekhez rugalmasan igazodik.

A fejlesztés során egy modern, nagy teljesítményű fejlesztői gépet használtunk, mely többmagos processzorral, nagy kapacitású memóriával és SSD háttértárral rendelkezik. A választott szoftveres eszközkészlet

Backend: ASP.NET Core Web Api (9.0)

Modern, teljesítményorientált backend megoldás, amely támogatja a RESTful API-k fejlesztését és a magas biztonsági szintet.

◆ **Backend bővítmények(packages):**

- Microsoft.AspNetCore.Authentication.JwtBearer (9.0.3)
- Microsoft.AspNetCore.Identity.EntityFrameworkCore (9.0.3)
- Microsoft.AspNetCore.OpenApi (9.0.3)
- Microsoft.EntityFrameworkCore (9.0.3)
- Microsoft.EntityFrameworkCore.Design (9.0.3)
- Microsoft.EntityFrameworkCore.Sqlite (9.0.3)
- Microsoft.EntityFrameworkCore.Tools (9.0.3)
- Swashbuckle.AspNetCore (7.3.1)

Admin Panel: WPF Application

Az asztali alkalmazásban a WPF keretrendszer lehetővé teszi a gazdag, interaktív felhasználói felület kialakítását.

Unit Test: xUnit Test Project

Az automatizált egységtesztelés révén biztosítottuk a kód megbízhatóságát és a funkcionális megfelelést.

Frontend: React + TypeScript

A frontend fejlesztésben a React dinamikus komponens-alapú architektúrája és a TypeScript típusbiztonsága biztosítja a fejlesztés gyorsaságát és hibamentességét.

◆ Modulok:

- Node
- React
- React-Router-Dom
- React-Lucide
- React-Chartjs-2

Ezek a technológiák lehetővé teszik a moduláris felépítést, az egyszerű karbantarthatóságot és a könnyű integrációt a jövőbeli bővítésekkel.

Adatszerkezet és adatmodell:

Adatbázis felépítése

A rendszer adatbázisának kialakítása során törekedtünk a normalizált,

jól strukturált adatmodell megvalósítására, amely garantálja a hatékony lekérdezést és a konzisztens adatok kezelését. Az adatbázis fő entitásai a következők:

□ **Etel, Kategoria, EtelKategoria:**

Az **Etel** és **Kategoria** táblák az alapinformációkat tartalmazzák, míg az **EtelKategoria** tábla valósítja meg a több-több kapcsolatot, lehetővé téve egy étel több kategóriához rendelését.

□ **Felhasznalo:**

Ebben a táblában kerülnek rögzítésre a felhasználók adatai, beleértve a személyes paramétereket (súly, magasság, életkor, stb.) és a személyre szabott étrendi célokat.

□ **HetiEtrend és MealFood:**

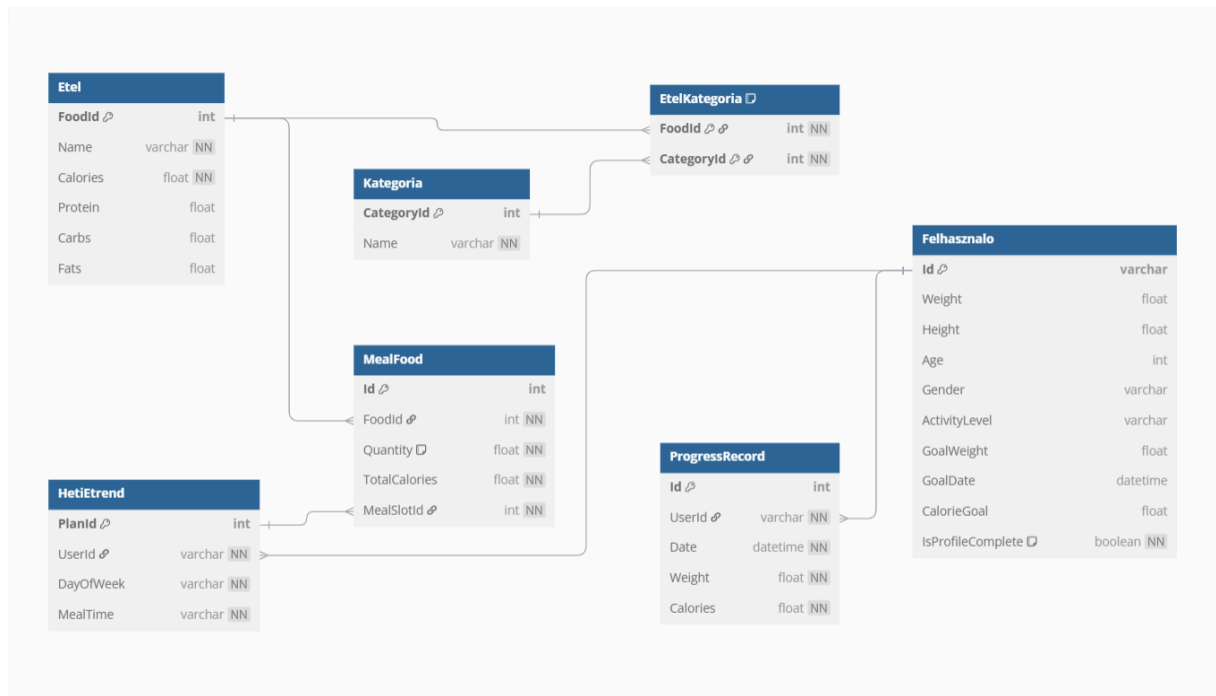
A **HetiEtrend** tábla a heti étkezési tervet reprezentálja, míg a **MealFood** a konkrét étkezésekhez tartozó tételsorokat rögzíti, beleértve az ételek mennyiségét és kalóriaértékét.

□ **ProgressRecord:**

Ez a tábla felelős a felhasználói aktivitás és az egészségügyi mutatók folyamatos nyomon követéséért.

Adatszerkezet dokumentációja

A táblák közötti kapcsolatokat egy diagrammal illusztráljuk, amely világosan bemutatja a relációkat (pl. az EtelKategoria idegen kulcsai az Etel és Kategoria táblákra mutatnak). Az adatmodell részletes leírása tartalmazza a táblák attribútumainak meghatározását, a kulcsok típusát, valamint az egyes mezők kötelezőségét, így a rendszer átlátható és könnyen karbantartható marad.



Adatbázis kódja:

```

Table Etel {
  FoodId int [pk, increment]
  Name varchar [not null]
  Calories float [not null]
  Protein float
  Carbs float
  Fats float
}
  
```

```

Table Kategoria {
  CategoryId int [pk, increment]
  Name varchar [not null]
}
  
```

```

Table EtelKategoria {
  FoodId int [not null]
  CategoryId int [not null]
  indexes {
    (FoodId, CategoryId) [pk]
  }
}
  
```



```
}  
}
```

```
Table Felhasznalo {  
  Id varchar [pk]  
  Weight float  
  Height float  
  Age int  
  Gender varchar  
  ActivityLevel varchar  
  GoalWeight float  
  GoalDate datetime  
  CalorieGoal float  
  IsProfileComplete boolean [not null, default: false]  
}
```

```
Table HetiEtrend {  
  PlanId int [pk, increment]  
  UserId varchar [not null]  
  DayOfWeek varchar [not null]  
  MealTime varchar [not null]  
}
```

```
Table MealFood {  
  Id int [pk, increment]  
  FoodId int [not null]  
  Quantity float [not null, note: "1-500"]  
  TotalCalories float [not null]  
  MealSlotId int [not null]  
}
```

```
Table ProgressRecord {  
  Id int [pk, increment]
```

```
UserId varchar [not null]  
Date datetime [not null]  
Weight float [not null]  
Calories float [not null]  
}
```

Ref: EtelKategoria.FoodId > Etel.FoodId

Ref: EtelKategoria.CategoryId > Kategoria.CategoryId

Ref: HetiEtrend.UserId > Felhasznalo.Id

Ref: MealFood.FoodId > Etel.FoodId

Ref: MealFood.MealSlotId > HetiEtrend.PlanId

Ref: ProgressRecord.UserId > Felhasznalo.Id

//Adatfájlok később, demo adatok stb

Tipikus algoritmusok, fontos működési szerkezetek

A rendszer implementációjában több olyan algoritmus szerepel, amelyek nemcsak a funkcionális követelmények teljesítését szolgálják, hanem a kód minőségét és karbantarthatóságát is elősegítik.

1. CSV Importálási Logika (C#)

❖ Mit csinál?

Ez az algoritmus felelős egy CSV fájl beolvasásáért, annak tartalmának soronkénti feldolgozásáért, a fejléc kihagyásáért, valamint az adatok validálásáért és konvertálásáért egy megfelelő adattranzfer objektummá (DTO).

Kiemelt megoldások a kódban:

□ Guard Clause a fájl ellenőrzésére:

- csharp
- Másolás
- `if (string.IsNullOrEmpty(selectedFilePath))`
- `return;`
- Az első ellenőrzés gyorsan visszatér, ha nincs kiválasztott fájl – ez csökkenti a felesleges beágyazott elágazásokat.

□ Aszinkron beolvasás és validáció:

- csharp
- Másolás
- `var lines = await File.ReadAllLinesAsync(selectedFilePath);`
- `if (lines.Length < 2) { /* hibaüzenet */ return; }`

Az aszinkron működés nem blokkolja a UI-t, és a fájl minimum két sorát (fejléc + legalább egy adat sor) ellenőrzi.

□ **Egyszerű, érthető iteráció és hibakezelés:**

A for ciklus az első sor (fejléc) kihagyásával dolgozik, minden sor esetén ellenőrzi, hogy nem üres, majd a Split metódussal osztja fel a sor tartalmát. Az egyes sorok esetében a belső try-catch blokk gondoskodik arról, hogy egyetlen rosszul formázott sor ne akadályozza meg az egész importálási folyamatot.

❖ **Clean Code jellemzők:**

- **Rövid, célratörő függvények:** Az eseménykezelők egy-egy feladatot látnak el (pl. fájl kiválasztása, importálás).
- **Érthető változónév:** Pl. selectedFilePath, successCount, errorCount – ezek azonnal utalnak a szerepükre.
- **Egyszerű hibakezelés:** A try-catch blokkok világosan elválasztják a hibás állapotokat, így a kód robusztus és jól karbantartható.

2. Kategória Azonosítók Feldolgozása (C#)

❖ **Mit csinál?**

Ez az algoritmus a CSV egy adott oszlopában lévő kategória azonosítókat dolgozza fel. Az azonosítókat pontosvesszővel elválasztva várja, majd minden egyes értéket `int.TryParse` segítségével ellenőriz és átalakít, mielőtt hozzáadná őket a `DTO CategoryIds` listájához.

Kiemelt megoldások a kódban:

□ **Felosztás és validálás:**

csharp

Másolás

```
if (columns.Length > 5 &&  
!string.IsNullOrEmpty(columns[5]))  
{  
    var catIds = columns[5].Split(';');  
    foreach (var id in catIds)  
    {  
        if (int.TryParse(id.Trim(), out int catId))  
        {  
            dto.CategoryIds.Add(catId);  
        }  
    }  
}
```

A kód először ellenőrzi, hogy létezik-e a kategória oszlop, majd a Split metódussal elválasztja az értékeket. A TryParse biztosítja, hogy csak érvényes egész számokat adunk hozzá.

❖ Clean Code jellemzők:

- **Kis, jól elkülöníthető felelősség:** A kategória feldolgozás egy jól elkülönített logikai egység, ami a későbbi módosításokat is egyszerűvé teszi.
- **Robusztus input kezelés:** A trim-elés és a TryParse használata biztosítja, hogy a nem megfelelő értékek ne okozzanak hibát.

3. Jelszó Erősség Számítása (React/TypeScript)

Mit csinál?

A getPasswordProgress függvény kiszámítja a megadott jelszó erősségét, azáltal, hogy három kritérium teljesülését ellenőrzi: tartalmaz-e nagybetűt, számot, illetve speciális karaktert. Az

eredmény egy százalékos érték, melyet például egy progress bar segítségével jelenít meg a felhasználó.

Kiemelt megoldások a kódban:

- **Egyszerű, deklaratív megoldás:**

typescript

Másolás

```
const getPasswordProgress = (password: string): number => {  
  let count = 0;  
  if (/[A-Z]/.test(password)) count++;  
  if (/[a-z]/.test(password)) count++;  
  if (/[\d]/.test(password)) count++;  
  if (/[@#%&*]/.test(password)) count++;  
  return (count / 3) * 100;  
};
```

A függvény világosan leírja a feltételeket, és a végén kiszámolja a százalékos értéket.

Clean Code jellemzők:

- **Egyértelmű funkcionális feladat:** A függvény csak azt csinálja, amire elnevezték, és jól dokumentált azáltal, hogy a kód olvasható és önmagáért beszél.
- **Egyszerűség és tisztaság:** Kis, jól izolált logika, mely könnyen tesztelhető és karbantartható.

4. Hibakódok Leképezése Felhasználóbarát Üzenetekre (React/TypeScript)

Mit csinál?

A `mapErrorCodeToMessage` függvény az API által visszaadott hibakódokat alakítja át felhasználó számára érthető üzenetekké. Az elágazások segítségével minden ismert hibakódhoz egyértelmű üzenet tartozik.

Kiemelt megoldások a kódban:

- **Deklaratív switch-case szerkezet:**

typescript

Másolás

```
const mapErrorCodeToMessage = (errorCode: string): string => {  
  const code = errorCode.trim();  
  switch (code) {  
    case "400":  
      return "Érvénytelen kérés.";  
    case "401":  
      return "Hibás hitelesítés.";  
    case "USER_EXISTS":  
      return "Ez az e-mail cím már regisztrálva van.";  
    case "INVALID_EMAIL":  
      return "Érvénytelen e-mail cím.";  
    case "WEAK_PASSWORD":  
      return "A jelszó túl gyenge. Legalább 6 karakter, és  
      tartalmazzon nagy betűt, számot, speciális karaktert.";  
    default:  
      return code;  
  }  
};
```

A kód könnyen bővíthető új hibakódokkal, és a felhasználóbarát üzenetek közvetlenül tükrözik az adott hibát.

Clean Code jellemzők:

- **Könnyen olvasható logika:** A switch-case struktúra miatt az egyes esetek külön sorban jelennek meg, ami elősegíti az átláthatóságot.
- **Egyszerű karbantarthatóság:** Új hibakódok hozzáadása egyszerű, mert egy jól elkülönített helyen van megírva a leképezés.

Tesztelési dokumentáció

A tesztelési folyamat során többféle, különböző körülményt és esetet vizsgáltunk, hogy biztosítsuk a rendszer stabilitását és robusztus hibakezelését. Az alábbiakban részletezzük a legfontosabb teszteseteket:

- **Regisztrációs folyamat:**
 - **Sikeres regisztráció:** A teszt azt vizsgálja, hogy új felhasználó esetén a rendszer helyesen generálja a megerősítő token-t és elküldi az e-mailt.
 - **Ütközés esetén:** Ellenőrizzük, hogy ha a megadott e-mailhez már tartozik felhasználó, akkor a rendszer Conflict választ ad vissza a megfelelő üzenettel („USER_EXISTS”).
- **Bejelentkezés:**
 - **Nem létező felhasználó:** A rendszer azonnal Unauthorized választ ad, ha a felhasználó nem található.
 - **Helytelen jelszó:** Ha a jelszó ellenőrzése sikertelen, szintén Unauthorized választ kap a felhasználó.
- **Felhasználói adatok módosítása:**
 - **Azonosító egyezőség ellenőrzése:** Ha az URL-ben kapott azonosító nem egyezik a kérésben szereplő DTO azonosítójával, a rendszer BadRequest hibát ad vissza az „Azonosító eltérés.” üzenettel.

További teszteset (opcionális):

A tesztelési dokumentáció kiterjeszthető olyan esetelemzéssel, amelyben ellenőrizzük az importálási folyamatot hibás vagy hiányos CSV fájl esetén, valamint a rendszer válaszát extrém adatmennyiség esetén. Így meggyőződhetünk arról, hogy a rendszer nem csak optimális, de nem ideális használat esetén sem omlik össze.

A tesztek során a xUnit keretrendszert használtuk az automatizált egységteszteléshez, melynek segítségével a fejlesztési folyamat minden kritikus pontján garantált a rendszer helyes működése.


```
using System;
using System.Collections.Generic;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Mvc;
using Microsoft.Extensions.Configuration;
using Microsoft.AspNetCore.Identity.UI.Services;
using Moq;
using Xunit;
using backend.Controllers;
using backend.Models;
```

```
namespace backend.Tests
```

```
{
```

```
    public class FelhasznaloControllerTests
```

```
    {
```

```
        // Segédmetódus a UserManager mock létrehozásához
```

```
        private Mock<UserManager<Felhasznalo>>
```

```
GetMockUserManager()
```

```
    {
```

```
        var store = new Mock<IUserStore<Felhasznalo>>();
```

```
        return new Mock<UserManager<Felhasznalo>>(store.Object,
null, null, null, null, null, null, null);
```

```
    }
```

```
// Segédmetódus a SignInManager mock létrehozásához

private Mock<SignInManager<Felhasznalo>>
GetMockSignInManager(UserManager<Felhasznalo> userManager)
{
    var contextAccessor = new
Mock<Microsoft.AspNetCore.Http.IHttpContextAccessor>();

    var userClaimsFactory = new
Mock<IUserClaimsPrincipalFactory<Felhasznalo>>();

    return new Mock<SignInManager<Felhasznalo>>(
        userManager,
        contextAccessor.Object,
        userClaimsFactory.Object,
        null,
        null,
        null,
        null);
}
```

```
// Teszt konfigurációs értékek előállítása (pl. FrontendUrl, Jwt
beállítások)
```

```
private IConfiguration GetTestConfiguration()
{
    var inMemorySettings = new Dictionary<string, string> {
        {"FrontendUrl", "http://localhost:4200"},
    }
```

```

        {"Jwt:Key", "ThisIsASecretKeyForJwtTokenDontShare"},
        {"Jwt:Issuer", "TestIssuer"},
        {"Jwt:Audience", "TestAudience"}
    };

    return new ConfigurationBuilder()
        .AddInMemoryCollection(inMemorySettings)
        .Build();
}

// -----
// Test 1: Sikeres regisztráció esetén CreatedAtAction eredmény
// -----

[Fact]
public async Task
Register_ReturnsCreated_WhenRegistrationSucceeds()
{
    // Arrange

    var userManagerMock = GetMockUserManager();
    var signInManagerMock =
    GetMockSignInManager(userManagerMock.Object);
    var emailSenderMock = new Mock<IEmailSender>();
    var configuration = GetTestConfiguration();

    // A teszt feltételezi, hogy az adott e-mailhez még nincs
    regisztrált felhasználó

```

```
userManagerMock.Setup(um =>
um.FindByEmailAsync(It.IsAny<string>()))
    .ReturnsAsync((Felhasznalo)null);

// A CreateAsync metódus sikeres eredményt ad vissza
userManagerMock.Setup(um =>
um.CreateAsync(It.IsAny<Felhasznalo>(), It.IsAny<string>()))
    .ReturnsAsync(IdentityResult.Success);

// Az e-mail megerősítés token generálása visszaad egy teszt
token-t

userManagerMock.Setup(um =>
um.GenerateEmailConfirmationTokenAsync(It.IsAny<Felhasznalo>()))
    .ReturnsAsync("TestToken");

var controller = new FelhasznaloController(
    userManagerMock.Object,
    signInManagerMock.Object,
    configuration,
    emailSenderMock.Object);

var registerDto = new RegisterDto
{
    UserName = "testuser",
    Email = "test@example.com",
    Password = "Test@123"
};
```

```
// Act

var result = await controller.Register(registerDto);

// Assert

var createdResult =
Assert.IsType<CreatedAtActionResult>(result);

Assert.NotNull(createdResult.Value);

// Ellenőrizzük, hogy az e-mail küldés megtörtént
emailSenderMock.Verify(es => es.SendEmailAsync(
    It.Is<string>(email => email == "test@example.com"),
    It.IsAny<string>(),
    It.IsAny<string>()), Times.Once);
```

/* Magyarázat:

- A teszt azt vizsgálja, hogy ha nincs még felhasználó az adott e-mail címmel,

a CreateAsync sikeresen lefut, majd generálódik egy megerősítő token, és az e-mail küldése is megtörténik.

- A várt eredmény egy CreatedAtActionResult, mely a létrehozott felhasználó adatait tartalmazza.

```
*/
}

// -----
```

// Test 2: Regisztráció esetén, ha a felhasználó már létezik
(Conflict eredmény)

// -----

[Fact]

```
public async Task
Register_ReturnsConflict_WhenUserAlreadyExists()
{
    // Arrange
    var userManagerMock = GetMockUserManager();
    var signInManagerMock =
    GetMockSignInManager(userManagerMock.Object);
    var emailSenderMock = new Mock<IEmailSender>();
    var configuration = GetTestConfiguration();

    // Itt azt szimuláljuk, hogy az e-mail címhez már létezik
    felhasználó

    userManagerMock.Setup(um =>
um.FindByEmailAsync(It.IsAny<string>()))

.ReturnsAsync(new Felhasznalo { Email =
"test@example.com" });

    var controller = new FelhasznaloController(
        userManagerMock.Object,
        signInManagerMock.Object,
        configuration,
        emailSenderMock.Object);
```

```

var registerDto = new RegisterDto
{
    UserName = "testuser",
    Email = "test@example.com",
    Password = "Test@123"
};

// Act
var result = await controller.Register(registerDto);

// Assert
var conflictResult =
Assert.IsType<ConflictObjectResult>(result);
Assert.Equal("USER_EXISTS", conflictResult.Value);

```

/* Magyarázat:

- Ebben a tesztben az FindByEmailAsync metódus nem null-t ad vissza, tehát a felhasználó már létezik.

- Ekkor a regisztráció nem folytatódik, és a rendszer Conflict (ütközés) hibával reagál.

```

*/
}

// -----

```

// Test 3: Login esetén, ha a felhasználó nem található
(Unauthorized eredmény)

// -----

[Fact]

```
public async Task
Login_ReturnsUnauthorized_WhenUserNotFound()
{
    // Arrange

    var userManagerMock = GetMockUserManager();

    var signInManagerMock =
    GetMockSignInManager(userManagerMock.Object);

    var emailSenderMock = new Mock<IEmailSender>();

    var configuration = GetTestConfiguration();


    // Szimuláljuk, hogy a megadott e-mailhez nem tartozik
    felhasználó

    userManagerMock.Setup(um =>
um.FindByEmailAsync(It.IsAny<string>()))

        .ReturnsAsync((Felhasznalo)null);


    var controller = new FelhasznaloController(
        userManagerMock.Object,
        signInManagerMock.Object,
        configuration,
        emailSenderMock.Object);
```



```

var loginDto = new LoginDto
{
    Email = "nonexistent@example.com",
    Password = "AnyPassword"
};

// Act
var result = await controller.Login(loginDto);

// Assert
var unauthorizedResult =
Assert.IsType<UnauthorizedObjectResult>(result);

Assert.Equal("Hibás email vagy jelszó.",
unauthorizedResult.Value);

```

/* Magyarázat:

- Ebben a tesztben a Login metódust úgy hívjuk meg, hogy az adott e-mail címhez nincs felhasználó.

- Az elvárt viselkedés, hogy a rendszer Unauthorized választ ad vissza a "Hibás email vagy jelszó." üzenettel.

```

*/
}

// -----

```

// Test 4: Login esetén, ha a jelszó ellenőrzése sikertelen
(Unauthorized eredmény)

// -----

[Fact]

```
public async Task
Login_ReturnsUnauthorized_WhenPasswordIsInvalid()
{
    // Arrange

    var userManagerMock = GetMockUserManager();

    var signInManagerMock =
    GetMockSignInManager(userManagerMock.Object);

    var emailSenderMock = new Mock<IEmailSender>();

    var configuration = GetTestConfiguration();

    var testUser = new Felhasznalo
    {
        Email = "test@example.com",
        UserName = "testuser",
        EmailConfirmed = true,
        Id = "123"
    };
};
```

// Szimuláljuk, hogy a felhasználó megtalálható

```
userManagerMock.Setup(um =>
um.FindByEmailAsync(It.IsAny<string>()))
```

```
.ReturnsAsync(testUser);

// A jelszó ellenőrzése sikertelen (nem egyezik a megadott
jelszó)

userManagerMock.Setup(um =>
um.CheckPasswordAsync(testUser, It.IsAny<string>()))

.ReturnsAsync(false);

var controller = new FelhasznaloController(
    userManagerMock.Object,
    signInManagerMock.Object,
    configuration,
    emailSenderMock.Object);

var loginDto = new LoginDto
{
    Email = "test@example.com",
    Password = "WrongPassword"
};

// Act

var result = await controller.Login(loginDto);

// Assert

var unauthorizedResult =
Assert.IsType<UnauthorizedObjectResult>(result);
```

```
Assert.Equal("Hibás email vagy jelszó.",  
unauthorizedResult.Value);
```

```
/* Magyarázat:
```

- Ebben a tesztben a felhasználó létezik, azonban a
CheckPasswordAsync metódus azt jelzi,

hogy a megadott jelszó helytelen.

- Ennek hatására a Login metódus Unauthorized választ ad
vissza.

```
*/  
}
```

```
// -----
```

// Test 5: UpdateUser esetén, ha az URL-ben megadott ID és a
DTO-ban lévő ID nem egyezik (BadRequest eredmény)

```
// -----
```

```
[Fact]
```

```
public async Task  
UpdateUser_ReturnsBadRequest_WhenIdMismatch()
```

```
{
```

```
    // Arrange
```

```
    var userManagerMock = GetMockUserManager();
```

```
    var signInManagerMock =  
    GetMockSignInManager(userManagerMock.Object);
```

```
    var emailSenderMock = new Mock<IEmailSender>();
```

```
    var configuration = GetTestConfiguration();
```

```
var controller = new FelhasznaloController(  
    userManagerMock.Object,  
    signInManagerMock.Object,  
    configuration,  
    emailSenderMock.Object);
```

```
var updateDto = new UpdateUserDto  
{  
    Id = "DifferentId", // A DTO-ban lévő azonosító  
    Weight = 70,  
    Height = 175,  
    Age = 30,  
    Gender = "Male",  
    ActivityLevel = "Moderate",  
    GoalWeight = 68,  
    GoalDate = DateTime.Now.AddMonths(1),  
    IsProfileComplete = true  
};
```

```
// Act
```

```
var result = await controller.UpdateUser("OriginalId",  
updateDto);
```

```
// Assert  
var badRequestResult =  
Assert.IsType<BadRequestObjectResult>(result);  
Assert.Equal("Azonosító eltérés.", badRequestResult.Value);
```

/* Magyarázat:

- Ebben a tesztben azt ellenőrizzük, hogy ha az URL-ben kapott azonosító ("OriginalId")

nem egyezik a kérésben szereplő DTO-ban lévő azonosítóval ("DifferentId"), akkor a metódus

BadRequest hibát ad vissza a "Azonosító eltérés." üzenettel.

```
*/  
}  
}  
}
```

Részletes magyarázat a tesztekhez

Register_ReturnsCreated_WhenRegistrationSucceeds

Mi történik:

Ebben a tesztben szimuláljuk a sikeres regisztrációt. A UserManager.FindByEmailAsync visszaad null-t, így nincs ütköző felhasználó. A CreateAsync metódus sikeresen lefut, majd generálódik egy megerősítő token, és az e-mail elküldése is megtörténik.

Elvárt eredmény:

A Register metódus CreatedAtActionResult-ot ad vissza, és az e-mail elküldése egyszer hívódik meg.

Register_ReturnsConflict_WhenUserAlreadyExists

Mi történik:

Ebben a tesztben az UserManager.FindByEmailAsync egy már létező felhasználót ad vissza. Így a regisztráció nem folytatódik, és a metódus ConflictObjectResult-ot ad vissza a "USER_EXISTS" üzenettel.

Elvárt eredmény:

A rendszer helyesen azonosítja az ütközést, és a regisztráció megszakad.

Login_ReturnsUnauthorized_WhenUserNotFound

Mi történik:

A Login metódus meghívásakor az adott e-mail címhez nem található felhasználó, így a metódus azonnal UnauthorizedObjectResult-ot ad vissza a "Hibás email vagy jelszó." üzenettel.

Elvárt eredmény:

A hitelesítés nem sikerül, mivel nincs ilyen felhasználó.

Login_ReturnsUnauthorized_WhenPasswordIsInvalid

Mi történik:

Ebben a tesztben a felhasználó megtalálható, de a megadott jelszó ellenőrzése sikertelen. Emiatt a Login metódus UnauthorizedObjectResult-ot ad vissza, ugyanezzel az üzenettel, mint az előző tesztnél.

Elvárt eredmény:

A jelszó ellenőrzésének sikertelensége miatt a bejelentkezés nem megy végbe.

UpdateUser_ReturnsBadRequest_WhenIdMismatch

Mi történik:

Az UpdateUser metódusnál az URL-ben kapott azonosító ("OriginalId") nem egyezik a kérésben szereplő DTO-ban lévő azonosítóval ("DifferentId"). Ennek hatására a metódus BadRequest hibát ad vissza az "Azonosító eltérés." üzenettel.

Elvárt eredmény:

A metódus helyesen ellenőrzi az azonosítók egyezőségét, és eltérés esetén hibával reagál.

További fejlesztési lehetőségek:

A jelenlegi implementáció számos további bővítési lehetőséget kínál, amelyek révén a projekt jövőbeni fejlesztése még inkább a felhasználói igényekhez igazítható:

- **Történeti étkezési adatok megjelenítése:**
Az étrend táblázatban a felhasználó számára lehetőséget biztosítunk arra, hogy hetekre visszamenőleg is megtekintse a korábbi étkezési tervét, ezzel támogatva a hosszútávú egészségügyi monitorozást.
- **Felhasználói receptek és közösségi funkciók:**
A rendszer bővíthető olyan funkciókkal, amelyek lehetővé teszik a felhasználók számára, hogy saját recepteket osszanak meg, illetve előre elkészített recept ajánlásokat kapjanak a kalóriabevitelük alapján.
- **Mobil applikáció fejlesztése:**
Az asztali és webes megoldások mellett egy natív mobil applikáció kifejlesztése segítené a szélesebb felhasználói réteg elérését, így még inkább támogatva a folyamatos egészséges életmódot.
- **Analitikai modul integrálása:**
A felhasználói viselkedés részletes elemzésével – például a rendszer használati szokásainak monitorozásával – tovább finomítható az ajánlási algoritmus, amely így még személyre szabottabb megoldásokat nyújtana.

3. Felhasználói dokumentáció

A TestreSzabva egy közösségi weblap, amely személyre szabott étrendtervezést és egészségügyi monitorozást tesz lehetővé. Röviden:

- **Mi ez?** Egy modern, interaktív számítógépes szoftver, amely segítséget nyújt az egészséges életmód kialakításához.
- **Fő funkciók:**
 - **Regisztráció és bejelentkezés:** Az első lépés a felhasználói fiók létrehozása, mely során megerősítő e-mail érkezik.
 - **Onboarding folyamat:** A személyes adatok (pl. súly, magasság, életkor) bekérése, mely alapján a rendszer kiszámítja a napi kalóriaigényt.
 - **Dashboard:** Itt láthatja a felhasználó a napi kalória bevitelt, hozzáadhat ételeket a heti menühöz, illetve nyomon követheti a haladását.
 - **További oldalak:** Haladás (grafikonos súlykezelés), Receptek (előre elkészített, illetve közösségi receptek megtekintése) és Beállítások (felhasználói adatok módosítása).

Hardver:

- Számítógép Windows 10 vagy Windows 11 operációs rendszerrel (Linuxon is futtatható, amennyiben kompatibilis környezetet biztosítanak).
- Stabil internetkapcsolat a letöltések és online funkciók érdekében.

Szoftverek:

- **Fejlesztői környezethez:**
 - Visual Studio és Visual Studio Code
 - Git (a projekt klónozásához)
- **Futtatási környezethez:**

- Node.js és npm (a frontend modulok telepítéséhez)
- Böngésző (pl. Google Chrome, Mozilla Firefox) a weblap megtekintéséhez

Telepítés és Indítás

1. Klónozás GitHub-ról:

A teljes projekt a [GitHub-on](#) érhető el. A következő lépések szükségesek:

1. Nyisson meg egy parancssort (CMD vagy Git Bash) és navigáljon abba a mappába, ahová telepíteni szeretné a projektet. Például:

C:\Users\felhasználóneved\Documents\usermappa

git clone <https://github.com/szimaakos/TestreSzabva.git>

ha ezt beírjuk, nyomunk egy Enter-t.

ha sikeres a lefuttatás, akkor ezt fogom látni:

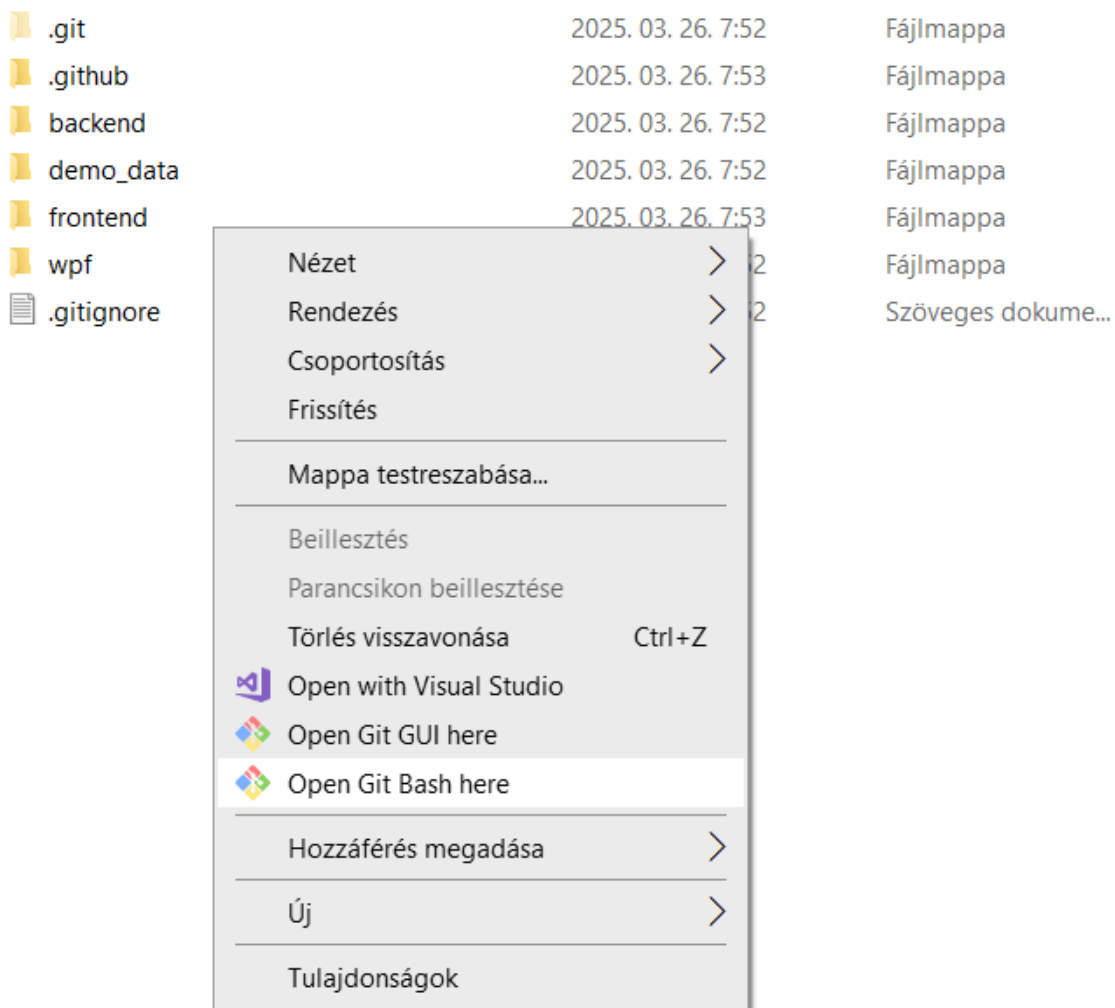
```
Cloning into 'TestreSzabva'...
remote: Enumerating objects: 465, done.
remote: Counting objects: 100% (465/465), done.
remote: Compressing objects: 100% (314/314), done.
remote: Total 465 (delta 242), reused 352 (delta 129), pack-reused 0 (from 0)
Receiving objects: 100% (465/465), 17.05 MiB | 23.34 MiB/s, done.
Resolving deltas: 100% (242/242), done.
```

Frontend indítása Visual Studio Code segítségével:

1. Nyissa meg a „Frontend” mappát: jobb klikk a mappára, majd válassza a „Git Bash megnyitása” opciót.
2. A megnyíló ablakban írja be:

code .

Ez elindítja a Visual Studio Code-ot.

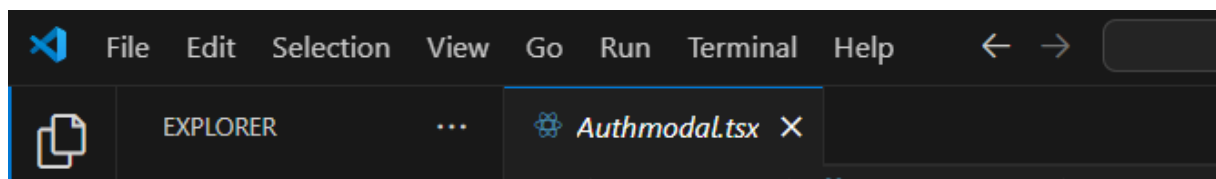


MINGW64:/c/Users/sutak.david.tamas/Documents/sutakdavid/TestreSzabva

```
sutak.david.tamas@BURG-1E03-01 MINGW64 ~/Documents/sutakdavid/TestreSzabva (main)  
$ code .|
```

Ezzel megnyitjuk a Visual Studio Code-ot.

A VSC-ben nyissa meg a Terminal fület, majd telepítse a szükséges Node modulokat:



Ha megnyitottuk a Terminal-t, kezdjük el letölteni a modulokat.

A legelső amit be kell írni:

- letöltjük a node modulokat

npm install

ha kész, ezt látjuk:

```
PS C:\Users\sutak.david.tamas\Documents\sutakdavid\TestreSzabva\frontend> npm i

added 190 packages, and audited 191 packages in 15s

43 packages are looking for funding
  run `npm fund` for details
```

- a következő:

npm install react-router-dom

```
PS C:\Users\sutak.david.tamas\Documents\sutakdavid\TestreSzabva\frontend> npm i react-router-dom

43 packages are looking for funding
  run `npm fund` for details

1 moderate severity vulnerability
```

- ez ha megvan, már csak egy lépés van hátra:

npm run dev

```
PS C:\Users\sutak.david.tamas\Documents\sutakdavid\TestreSzabva\frontend> npm run dev

> x@0.0.0 dev
> vite

VITE v6.2.1 ready in 286 ms

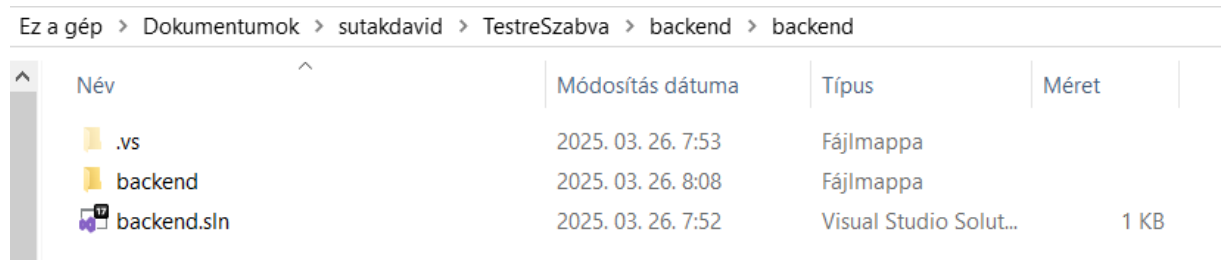
→ Local:   http://localhost:5173/
→ Network: use --host to expose
→ press h + enter to show help
```

utolsó lépés: rányomunk a linkre ami megjelenik, ebben az esetben a **localhost:5173**.

Visual Studio:

Backend indítása Visual Studio segítségével:

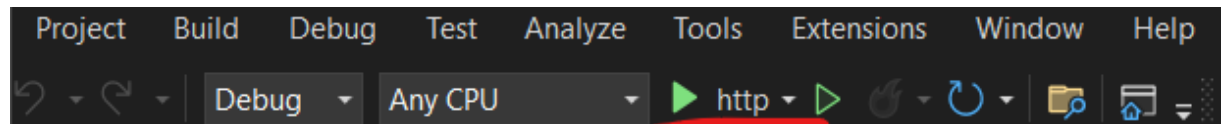
1. Keresse meg a projekt „Backend” mappáját, amíg meg nem találja a backend.sln (solution) fájlt.



Név	Módosítás dátuma	Típus	Méret
.vs	2025. 03. 26. 7:53	Fájlmappa	
backend	2025. 03. 26. 8:08	Fájlmappa	
backend.sln	2025. 03. 26. 7:52	Visual Studio Solut...	1 KB

rákattintunk, megnyílik a Visual Studio.

Nyissa meg a fájlt Visual Studio-ban, majd a képernyő tetején a „HTTP” gombra kattintva indítsa el a backend szerveret.



A backend indítása után a felhasználói műveletek (regisztráció, bejelentkezés, adatbevitel) támogatása élővé válik.

A Program Részletes Használata

Regisztráció és Bejelentkezés:

- **Regisztráció:**
 - Adja meg a szükséges adatokat: név, e-mail cím és jelszó.
 - A rendszer egy megerősítő e-mailt küld a megadott címre.
 - A megerősítő e-mailben található linkre kattintva aktiválhatja a fiókját.

-

Üdvözlünk sutakdavidteszt a TestreSzabva világában!

Köszöntünk a TestreSzabva közösségében!
Már csak egy lépés választ el attól, hogy
teljes mértékben kihasználhasd az
alkalmazás által nyújtott személyre szabott
élményt.

E-mail cím megerősítése

Az e-mail címed megerősítésével:

- Hozzáférsz az összes személyre szabott szolgáltatásunkhoz
- Biztonságban tudhatod fiókodat
- Értesülhetsz a legújabb fejlesztéseinkről és ajánlatainkról

Ha nem te regisztráltál oldalunkon, kérjük,
hagyd figyelmen kívül ezt az üzenetet.

Köszönjük, hogy minket választottál!

, rákattintunk az e-mail megerősítő gombra, akkor megerősítődik az e-mail, és bejelentkezhetünk az oldalra.



E-mail cím megerősítve!

Köszönjük, hogy megerősítetted az e-mail címedet.

Most már bejelentkezhetsz a TestreSzabva alkalmazásba és élvezheted az összes funkciót.

Tovább a bejelentkezéshez



Onboarding:

- A fiók aktiválása után az Onboarding oldalon kérjük a felhasználót, hogy adja meg személyes adatait (pl. súly, magasság, életkor, stb.), amelyek alapján a rendszer kiszámítja a napi kalóriaigényt.



Dashboard és További Oldalak:

- **Dashboard:** Megjeleníti a napi kalória bevitelt, és itt tud új ételeket hozzáadni a heti étkezési tervhez.
- **Haladás:** Grafikonon nyomon követheti a súlyváltozását, új súlyadat bevitelével frissítheti az információkat.
- **Receptek:** Megtekintheti az előre elkészített recepteket, illetve a közösségi felületen feltöltött új ajánlásokat.

- **Beállítások:** Személyre szabhatja regisztrációs adatait, illetve módosíthatja az egyéb preferenciáit.

Szövegbeviteli Mezők Részletes Leírása:

- Minden beviteli mező esetében a dokumentáció tartalmazza a várható adatformátumot:
 - **Név:** Maximum 50 karakter, engedélyezett az ékezetes betűk használata.
 - **E-mail cím:** Szabványos e-mail formátum (pl. example@domain.com).
 - **Jelszó:** Minimum 6 karakter, melynek tartalmaznia kell legalább egy nagybetűt, egy számot és egy speciális karaktert.
 - További mezők (pl. súly, magasság) esetén a megengedett értéktartományok és formátumok is szerepelnek.

Hibajelzések Magyarázata

A rendszer különböző hibajelzésekkel segíti a felhasználót, ha valamilyen adatbevitel hibás vagy hiányos:

- **Regisztrációs hiba:**
 - Ha egy e-mail cím már regisztrálva van, a rendszer „USER_EXISTS” üzenettel figyelmeztet.
- **Bejelentkezési hiba:**
 - Hibás e-mail vagy jelszó esetén a rendszer „Hibás email vagy jelszó.” üzenetet jelenít meg.
- **Űrlapkitöltési hibák:**
 - Ha egy kötelező mező üresen marad vagy az adat formátuma nem megfelelő (pl. nem érvényes e-mail cím), a rendszer egyértelmű figyelmeztetést ad, amely leírja a helyes formátumot.

A hibajelzések célja, hogy a felhasználó azonnal tudja, hol kell javítania az adatain, így biztosítva a hibamentes működést.

Admin Felhasználói Felület

A TestreSzabva rendszer külön admin felülettel is rendelkezik, amely kizárólag a jogosultsággal rendelkező felhasználók számára érhető el:

- **Fő funkciók:**

- Receptek hozzáadása
- Receptek szerkesztése, törlése
- Ételek hozzáadása
- Ételek szerkesztése, törlése

Belépési adatok:

- Az admin felület eléréséhez szükséges külön regisztráció vagy előre megadott belépési adatok (például: admin@domain.hu / Admin@123) szükségesek.

Az admin felület segítségével a rendszer karbantartása és a felhasználói problémák gyors megoldása válik lehetővé.

Információkérés és Támogatás

Amennyiben a felhasználó kérdésekkel vagy problémákkal fordulna hozzánk, az alábbi elérhetőségek állnak rendelkezésre:

- **E-mail:** testreszabvaapp@gmail.com

Az információkérés célja, hogy minden felmerülő kérdésre gyors és hatékony választ kapjon a felhasználó.

4. Összefoglalás és Köszönetnyilvánítás

Összefoglalás

A TestreSzabva projekt célja az volt, hogy egy innovatív, felhasználóbarát és hosszútávon fenntartható rendszert hozzunk létre, amely támogatja az egészséges életmód kialakítását és fenntartását. A fejlesztési folyamat során számos technikai kihívással szembesültünk, melyek során értékes tapasztalatokat szereztünk az adatmodellezés, algoritmusok kialakítása és a robusztus hibakezelés terén. A projekt eredménye azt mutatja, hogy a rendszer képes hatékonyan segíteni mindazokat, akik elkötelezettek az egészséges, sportos életvitel mellett, különösen azoknak, akik most szeretnének valódi változást elérni az életükben.

A felhasználói visszajelzések és teszteredmények alapján biztosak vagyunk abban, hogy a TestreSzabva nemcsak technikai értelemben, hanem a mindennapi életminőség javítása szempontjából is jelentős értéket képvisel. Tisztában vagyunk a jelenlegi megoldás limitációival, de a projekt jövőbeli fejlesztési irányai között szerepel többek között a bővített étkezési adatok visszatekintése, a közösségi funkciók erősítése és egy natív mobil applikáció kifejlesztése is.

Köszönetnyilvánítás

Szeretnénk megköszönni minden mentorunknak, kollégánknak és tesztfelhasználónknak a folyamatos támogatást, értékes visszajelzéseket és szakmai iránymutatást, amelyek nélkül ez a projekt nem jöhetett volna létre. A TestreSzabva projekt sikere egy csapatmunkának köszönhető, és büszkék vagyunk arra, hogy egy olyan közösséget építhetünk, amely elkötelezett az egészséges életmód mellett.