



EÖTVÖS LORÁND TUDOMÁNYEGYETEM

INFORMATIKAI KAR

PROGRAMOZÁSELMÉLET ÉS SZOFTVERTECHNOLÓGIAI  
TANSZÉK

## Pénzügyeket rendszerező webes alkalmazás

*Témavezető:*

Bende Imre  
adjunktus

*Szerző:*

Németh Zsófia Réka  
programtervező informatikus BSc

*Budapest, 2025*

# Tartalomjegyzék

<b>1. Bevezetés</b>	<b>3</b>
<b>2. Felhasználói dokumentáció</b>	<b>4</b>
2.1. Rendszerkövetelmények . . . . .	4
2.2. Felhasználói esetek . . . . .	4
2.2.1. Fiók . . . . .	5
2.2.2. Profil . . . . .	5
2.2.3. Pénzügyek . . . . .	6
2.2.4. Csoportok . . . . .	7
<b>3. Fejlesztői dokumentáció</b>	<b>8</b>
3.1. Architektúra . . . . .	8
3.1.1. Architektúra leírása . . . . .	9
3.2. Backend . . . . .	10
3.2.1. appsettings.json . . . . .	10
3.2.2. Program.cs . . . . .	10
3.2.3. Modellek . . . . .	11
3.2.4. API . . . . .	11
3.3. Frontend . . . . .	11
3.3.1. Layout . . . . .	11
3.3.2. static . . . . .	13
3.4. Adatbázis . . . . .	13
3.4.1. Backend - Adatbázis kapcsolat . . . . .	13
3.4.2. Migrations . . . . .	15
3.5. Tesztek . . . . .	15
3.6. UML diagramok . . . . .	15
<b>4. Összegzés</b>	<b>16</b>

5. Továbbfejlesztési lehetőségek	17
Köszönetnyilvánítás	18
Irodalomjegyzék	18
Ábrajegyzék	19
Táblázatjegyzék	20
Forráskódjegyzék	21

# 1. fejezet

## Bevezetés

A választott témám egy személyes pénzügyeket rendszerező webalkalmazás. A dolgozat a full-stack webfejlesztés témaköréhez kapcsolódik, mivel frontend, backend és adatbázis kezelést is magában foglal. A backendet egy C# alapú ASP.NET Web App (Razor Pages) keretrendszerrel valósítottam meg, melyhez egy MySQL relációs adatbázist csatoltam. A frontend részt pedig egy egyszerű HTML/CSS/JS (JavaScript) kombináció alkotja, Bootstrap (CSS Framework) elemeket felhasználva.

Az alkalmazás fő lényege, hogy költségeinket és bevételeinket tudjuk nyomon követni, illetve spórolási segítséget nyújt, kimutatásokat készít a pénzkezelési szokásainkról, mindezt egy kifinomult, egyszerű és felhasználóbarát felületen keresztül. A jelenlegi megoldások erre a célra véleményem szerint nem elég egyszerűek, vagy pedig túl drágák egy hétköznapi ember számára.

A következő fejezetekben fogom kifejteni a projektet bővebben a felhasználói és fejlesztői dokumentációkon keresztül.

## 2. fejezet

# Felhasználói dokumentáció

A következő fejezetben fogom bemutatni az alkalmazás elérését, egyes komponenseit, illetve felhasználási lehetőségeit.

Ez a pénzügyeket rendszerező alkalmazás alapvetően magánszemélyeknek készült, személyes felhasználásra, de mivel lehetőséget nyújt csoportos használatra is, ezáltal akár egy kisebb vállalat igényeit is elláthatja.

A főbb funkciók közé tartozik, hogy bevételeket és kiadásokat lehet rögzíteni, kategóriák szerint csoportosítva, illetve kimutatásokat nézhet a felhasználó a pénzügyi szokásairól, melyeket exportálni is tudja. Az alkalmazás egyik nagy előnye, hogy nem csak egyének használhatják, hanem csoportok (például háztartások) is.

### 2.1. Rendszerkövetelmények

Mivel egy webes alkalmazásról van szó, ezért különleges gépigény nem szükséges. Szinte az összes böngésző támogatott.

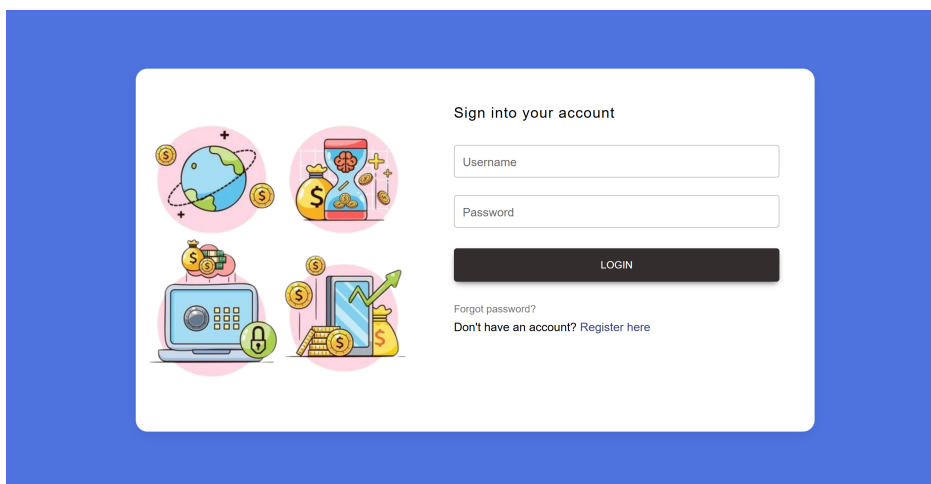
### 2.2. Felhasználói esetek

Az alkalmazás felhasználói eseteit ez a bejegyzés fogja taglalni jobban, képernyőképekkel magyarázva.

## 2.2.1. Fiók

Funkció	Leírás
<i>Bejelentkezés</i>	Bejelentkezés egy egyedi felhasználónévvel és egy jelszóval lehetséges.
<i>Regisztráció</i>	Regisztrálni lehet az oldalra a következő adatok megadásával: e-mail cím, felhasználónév (egyedi), teljes név, jelszó (minimum 8 karakter, legalább 1 szám, legalább 1 nagy betű, legalább 1 speciális karakter).
<i>Elfelejtett jelszó</i>	Ha a felhasználó elfelejtette a jelszavát, akkor a Bejelentkezés felületen az "Elfelejtett jelszó" hivatkozásra kattintva megjelenik egy új felület, csak egy e-mail beviteli mezővel. A rendszer küldeni fog egy egyedi tokennel ellátott linket a megadott e-mail címre (amely 30 percig érvényes), és ezen a linken tudja a felhasználó majd megadni az új jelszavát (szintén a biztonsági kritériumoknak megfelelő formában), és sikeres bejelentkezést követően már bent is lesz a fiókjában.
<i>Kijelentkezés</i>	Ha a felhasználó befejezte kívánt tevékenységét, akkor a bal oldali menüsáv alján található "Logout" gomb megnyomásával tud kijelentkezni. Alapvetően 30 perc inaktivitás után "dob le" az oldal (azaz jár le az adott session).

2.1. táblázat. Fiók műveletek



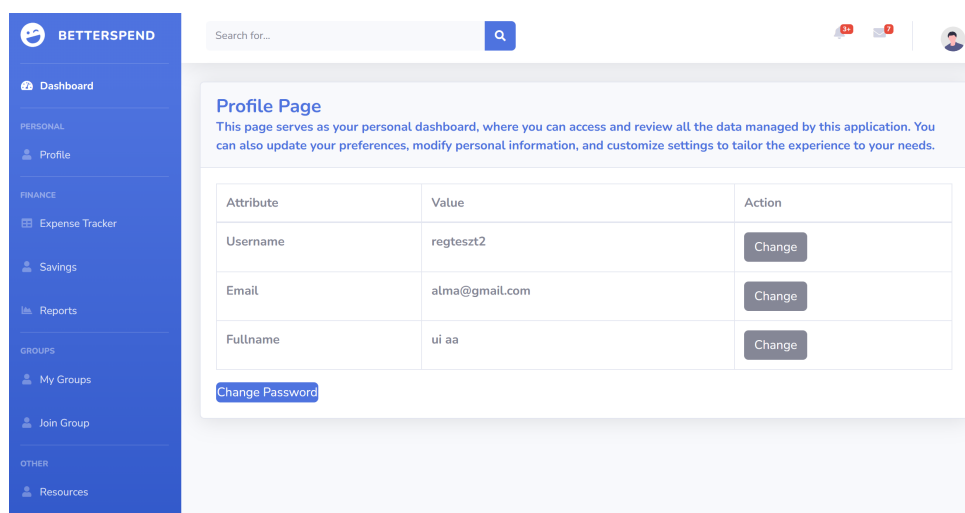
2.1. ábra. Screenshot: Bejelentkező felület

## 2.2.2. Profil

A profil oldalt a bal menüsávból érhetjük el.

Funkció	Leírás
<i>Adatok megtekintése</i>	A regisztrációnál megadott személyes adatait a felhasználó itt tekintheti meg.
<i>Adatok módosítása</i>	A személyes adatok módosítására is itt van lehetőség, egyszerű beviteli mezőkkel lehet módosítani a felhasználónevet (egyedi), teljes nevet és e-mail címet.
<i>Jelszó módosítás</i>	Ha a felhasználó módosítani kívánja a jelszavát, akkor a "Jelszó módosítása" gombra kattintva megjelenik egy új felület, két jelszó beviteli mezővel (új jelszó, új jelszó megerősítése). Itt szintén érvényesek a korábban taglalt (2.1 táblázat) biztonsági kritériumok. Ezután sikeres bejelentkezést követően már bent is lesz a fiókjában.

2.2. táblázat. Profil oldal



2.2. ábra. Screenshot: Profil felület

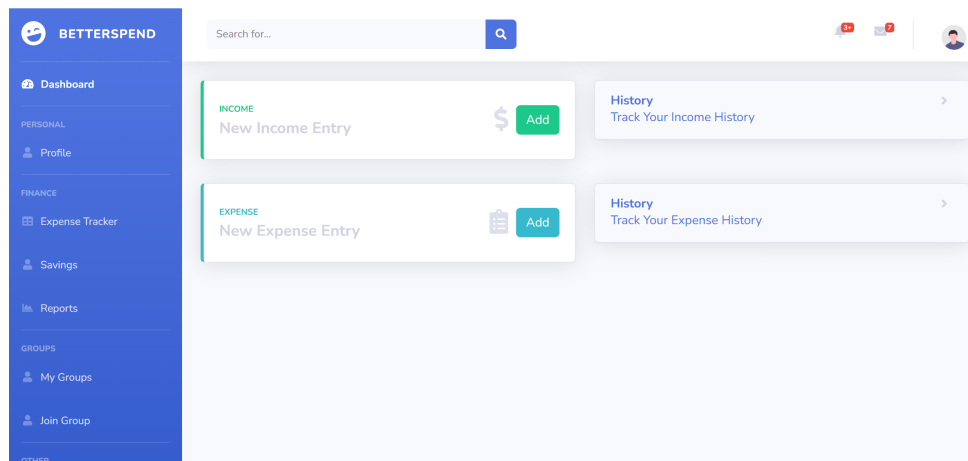
### 2.2.3. Pénzügyek

A pénzügyeket kezelő oldalakat a bal menüsávból érhetjük el. Ezen a főmenüponton belül is 3 almenüpont lett kialakítva:

- Expense Tracker (Bevételek és kiadások vezetése, és előzmények megtekintése)
- Savings (Megtakarítások megtekintése és kezelése)
- Reports (Kimutatások megtekintése, személyre szabása és exportálása)

Funkció	Leírás
<i>Kiadás/bevétel rögzítése</i>	Külön dobozokban lehet rögzíteni a kiadásokat, és a bevételeket. Egyszerre egyet lehet bevinni, melynek adni kell egy összeget, egy leírást, opcionálisan lehet kategóriát is megadni.
<i>Előzmények</i>	A költségi és bevételi előzményeket is itt lehet megtekinteni.

2.3. táblázat. Expense Tracker oldal



2.3. ábra. Screenshot: Expense Tracker felület

Funkció	Leírás
-	-

2.4. táblázat. Savings oldal

Funkció	Leírás
-	-

2.5. táblázat. Reports oldal

## 2.2.4. Csoportok

-



## 3. fejezet

# Fejlesztői dokumentáció

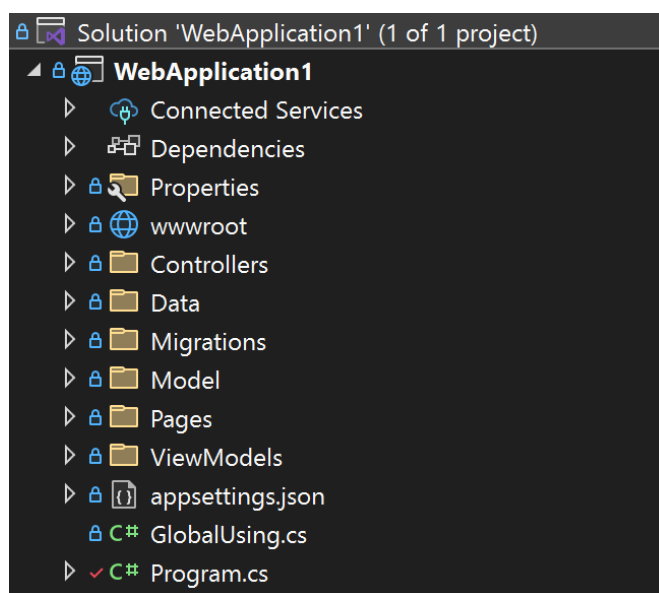
A következő fejezetben az alkalmazást fejlesztői szemszögből mutatom be. Részletesen kitérek az alkalmazás felépítésére, a használt technológiákra, az adatbázis-struktúrára, valamint a fejlesztés során követett elvekre és megoldásokra.

### 3.1. Architektúra

- Frontend: HTML<sup>1</sup>/CSS/JavaScript (és Bootstrap)
- Backend: C# (ASP.NET Web App (Razor Pages))
- Adatbázis: MySQL relációs adatbázis

---

<sup>1</sup>Mivel a C# projekten belül lett létrehozva a frontend is, ezért a .html helyett .cshtml kiterjesztésű fájlok vannak. Ezek annyiban különböznek a HTML-től, hogy vannak bizonyos tagek, kulcsszavak, melyekkel különleges dolgokat tudunk csinálni. A Frontend című fejezetben kerül ez a téma bővebb kifejtésre.



3.1. ábra. Projekt struktúra

### 3.1.1. Architektúra leírása

A .NET-es integrált frontend és backend fejlesztés hatalmas előnye a rendszerezettség, a szabályszerű kommunikáció az egyes rétegek között (illetve ezen rétegek helyes elkülönölése), és a rengeteg beépített segédfüggvény/konfiguráció/cshtml tag. Kiemelném még a modellek használatát is, amelyek egyszerűbb és biztonságosabb (például SQL injection elleni védelem) adatbázis kezelést biztosítanak.

Az egyes route-ok konfigurálása is meglehetősen letisztult ebben a keretrendszerben. A "Pages" mappa<sup>2</sup> fájlstruktúrája alapján automatikusan létrejönnek a route-ok, ha a "Program.cs"-ben megadjuk a programnak, hogy hozza létre őket:

```
1 app.UseRouting();
```

3.1. forráskód. Route-ok konfigurálása

A frontend és a backend közötti hagyományos JavaScript segítségével történő kommunikáción kívül, a cshtml formátum miatt lehetőség nyílik egyszerűbb esetekre közvetlen kapcsolatot is létesíteni a backend és a frontend között. Például:

```
1 <h1>Username</h1>
2 <p>@Model.UserData.Username</p>
```

3.2. forráskód. Frontend modell egy alkalmazása

<sup>2</sup>A "Pages" mappa, ahogy a neve is mutatja, tartalmazza a weboldal egyes oldalait. Bővebb kifejtés a Frontend című fejezetben.

Itt ugye mint látható a modellből tudunk adatot lekérdezni. De mi is a "Model" pontosan? Ugyebár az ASP .NET Web App keretrendszer úgy működik, hogy minden oldal (Razor page) egy .cshtml és egy .cs fájl együtteséből áll össze. A .cs kiterjesztésű fájl az adott oldal modellje. Itt definiálhatunk adatszerkezeteket és függvényeket, mint például az OnGet() és OnPost(), amelyek beépített (opcionális) metódusok, és ahogy a nevük is mutatja, az oldalról érkező GET és POST requesteket kezelik. Tehát például, ha az adott oldalon egy darab form-unk van, amit POST metódussal be akarunk küldeni a szervernek, ezt JS (JavaScript) kód írása nélkül biztonságosan meg tudjuk tenni.

## 3.2. Backend

Először is nézzük meg az alkalmazás egyes backendet felépítő elemeit, és azok funkcionalitását, kezelését.

### 3.2.1. appsettings.json

Az appsettings.json egy konfigurációs fájl. Ebben lehet alkalmazásszintű beállításokat tárolni, például: adatbáziskapcsolati stringeket, API kulcsokat, logolási beállításokat, vagy bármilyen egyedi, fejlesztő által definiált értéket. Az értékeket a program könnyen kiolvassa a beépített konfigurációs rendszer segítségével (pl. Configuration["Kulcs"] vagy erősen típusos binding-gel). Ez segít elkülöníteni a kódot a konfigurációtól, így könnyebb a karbantartás, környezetenkénti eltérés kezelése (pl. fejlesztői vs. éles környezet).

### 3.2.2. Program.cs

Ez a fő program fájl. Ebben készítjük elő a webalkalmazásunk tulajdonságait (persze a C# egyszerűségének hála ez nem sok plusz feladattal jár, csupán a helyes függvényeket kell meghívni helyes sorrendben, attól függően, hogy hogyan akarjuk konfigurálni az alkalmazást.) Itt lehet beállítani a korábban említett routing-ot, az autentikációt, a cookie-kat, a session-t, a fejlesztői és éles környezeteket, és még sok minden mást. Itt tudjuk felkonfigurálni a modellt, az adatbázist és az oldalakat (Pages) is. ezután az app.Run() függvény hívással tudjuk ténylegesen elindítani a weboldalt, és ekkor localhoston el is kezd futni.

### 3.2.3. Modellek

Minden adatbázis táblának létrehozunk egy modellt (bővebb kifejtés az "adatbázis" pontban). Ezek mind C# osztályok táblánként, és minden oszlop egy külön propertynek feleltethető meg.

### 3.2.4. API

Az ASP.NET-es struktúra úgy működik, hogy miután a Program.cs-ben felkonfiguráltuk a szükséges dolgokat, megalkottuk a modelleket, létrehoztuk a HTML lapokat, már csak egy fontos dolog maradt hátra. A frontend-backend kommunikáció. Említés esett már arról, hogy egyszerűbb esetekben ezt a HTML-ben bizonyos tagekkel meg tudjuk tenni, de a nem triviális esetek megoldására is egyszerű módot kínál a .NET. A Controllers mappában minden modellnek (vagyis minden adatbázis táblának) létrehozunk egy Controller fájlt is. Ebben fogunk API-végpontokat definiálni, melyeket a frontenden JavaScript segítségével tudunk meghívni, és ezáltal adatot közvetíteni a felhasználó és a szerver között (oda-vissza). Az API-endpointok tesztelésére és leírására egy hasznos, szintén beépített, megoldást használhatunk, a Swaggert. Ezt is a Program.cs fájlban állíthatjuk be.<sup>3</sup>

## 3.3. Frontend

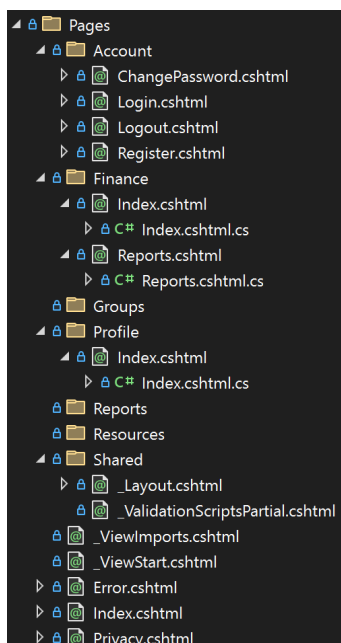
Térjünk át a frontend réteg alkotóelemeire, és ezek működésére. Itt is komponensenként fogom bemutatni a felhasznált technológiákat.

### 3.3.1. Layout

A "Pages" mappa tartalmazza az oldalakat. A keretrendszer ugyebár úgy működik, hogy minden oldal (Razor page) egy .cshtml és egy .cs fájl együtteséből áll össze. A Pages mappa ezeket a párosokat tartalmazza, további kisebb mappákra bontva a funkció csoportok alapján. Az alább mellékelt ábrából (3.2. ábra) látható tehát, hogy például a bejelentkező felületet a következő linken tudjuk elérni: "localhost:«port»/Account/Login".

---

<sup>3</sup>Szigorúan csak Development módban jelenjen meg, adatvédelmi/adat hitelességi okokból.



3.2. ábra. Pages struktúra

Találhatók még ebben a mappában egyéb segédfájlok is, mint például az Error.cshtml, ami az esetleges hibák esetén jelenik meg. Kiemelendő még a Shared mappa, amiben található egy Layout.cshtml fájl is (modell nélkül). Ez, ahogy a neve is mutatja egy alap sablont biztosít az alkalmazás felületének. A weboldal egy (a képernyő baloldán lévő) főmenüből, egy kis (az oldal tetején található) menüsávból, a fő tartalmi részből áll, illetve egy footer részből áll. A footer és a menüsávok kódját tartalmazza a Layout.cshtml között a tartalomnak "kihagyott" rész:

```

1 <div class="container-fluid">
2 @RenderBody()
3 </div>

```

### 3.3. forráskód. Layout extension ASP.NET keretrendszerben

A RenderBody() függvény (szintén beépített .NET metódus) behelyezi az adott oldal HTML kódját ebbe a div HTML elembe. Ez automatikusan megtörténik az összes oldal esetében amelyre navigál a felhasználó. Ha valami miatt éppen nem ezt a sablont akarjuk követni egy oldalunkkal (például: Bejelentkező felület), akkor az adott oldal HTML kódjába beírhatjuk hogy ne sablont kövessen:

```

1 @{
2     Layout = null; // Remove layout if you want a standalone page
3 }

```

### 3.4. forráskód. Layout extension ASP.NET keretrendszerben

### 3.3.2. static

A "wwwroot" mappában található az összes statikus erőforrás (static resource). Itt a hagyományosan használt mappastruktúrát követtem az alkalmazás készítésekor, azaz létre lettek hozva a következő mappák:

- "js": JavaScript kódok
- "css": CSS stylesheetek
- "img": Képek

A frontend rész formázását Bootstrap keretrendszer felhasználásával valósítottam meg. Ezt úgy importáltam a projektbe, hogy a kész CSS/JS fileok vannak letöltve ugyanide, a wwwroot mappába, ezen kívül egy külső hivatkozás van egy betűtípusra.

## 3.4. Adatbázis

Az alkalmazás adatbázisa egy localhoston futó MySQL adatbázis. A MySQL-t sok szempont miatt szokták kisebb alkalmazásokhoz használni, ezek közé tartozik például az a nem elhanyagolható indok, hogy teljesen ingyenes a használata. Illetve szintén az egyszerűségét tudnám kiemelni, és a kompatibilitását az ASP.NET keretrendszerrel. Egy darab adatbázis lett létrehozva, azon belül több tábla található, melyek természetesen kapcsolódnak egymáshoz. Az adatmodell a relációs adatbázisok normalizálási elvein alapul, és a harmadik normálformáig (3NF) került kialakításra.

### 3.4.1. Backend - Adatbázis kapcsolat

A backend és az adatbázis kapcsolata relatív egyszerűen megvalósítható az ASP.NET keretrendszerben. A NuGet package managerből az EntityFramework egyes csomagjait, illetve a MySQL Connector csomagot letöltve már alig van dolgunk. Az appsettings.json fájlban tudjuk a connection stringet definiálni.

```
1  "ConnectionStrings": {  
2    "MySQLConnection": "server=localhost;port=3306;database=  
    betterspend;user=root;password=<<password>>;"  
3  }
```

3.5. forráskód. Adatbázis Connection String

majd a Program.cs fájlban tudunk ténylegesen csatlakozni:

```
1 // Get MySQL connection string from configuration
2 string connectionString = builder.Configuration.
    GetConnectionString("MySQLConnection");
3
4 // Add DB Context to the application
5 builder.Services.AddDbContext<ApplicationDbContext>(options =>
6 {
7     options.UseMySQL(connectionString, ServerVersion.AutoDetect(
8         connectionString));
9 });
```

### 3.6. forráskód. Program.cs: Adatbázis csatlakozás

A tényleges adatbázis modellje:

```
1 using WebApplication1.Model;
2 namespace WebApplication1.Data
3 {
4     public class ApplicationDbContext : DbContext
5     {
6         public ApplicationDbContext(DbContextOptions<
7             ApplicationDbContext> options)
8             : base(options)
9         {
10
11             public DbSet<User> Users { get; set; }
12             public DbSet<Transaction> Transactions { get; set; }
13             public DbSet<Category> Categories { get; set; }
14
15         }
16     }
```

### 3.7. forráskód. Adatbázis modell

Ezután már az adatbázisunk össze van kötve teljesen az alkalmazással. ASP.NET-ben közvetlenül az adatbázishoz (lekérdezésekkel, parancsokkal) nem tudunk hozzáférni. Amihez hozzá tudunk férni azok a modell fájlok (pl. Model/User). Ezeket tudjuk módosítani LINQ segítségével egyszerűen, és ez szinkronizálni fogja a tényleges adatbázisunkkal.

### 3.4.2. Migrations

A későbbi fejlesztések során migrációkat használhatunk az adatbázis verziókövetésére és a változtatások biztonságos, konzisztens bevezetésére. A migrációk használata Entity Framework Core segítségével történik, ahol a DbContext osztály definiálja az adatbázis szerkezetét. A módosításokat a `dotnet ef migrations add` paranccsal lehet rögzíteni, majd a `dotnet ef database update` parancs segítségével alkalmazni az adatbázisra. Ez lehetővé teszi a séma változásainak verziókövetését és biztonságos frissítését.

## 3.5. Tesztek

---

## 3.6. UML diagramok

---



## 4. fejezet

# Összegzés

Összegezve tehát az alkalmazás elsősorban a spórolást, tudatos pénzköltést szeretné promótálni, és erre kínál magánszemélyeknek vagy háztartásoknak/családoknak egy hasznos eszközt. A fő funkció a költségeink és bevételeink vezetése, amely az első lépés a tudatosság és a pénzügyi szabadság irányába. Ezután a következő lépésnek tekinthetjük azt, amikor már az okosabb költekezés következményeképpen többlet bevételünk is lesz, melyet elkezdünk félrerakni. A megtakarításainkat is nyomon tudjuk követni az alkalmazáson belül, sőt még kimutatásokat is láthatunk, melyek rendszeres tanulmányozásával különböző mintázatokat vélhetünk felfedezni pénzügyi szokásainkból, amelyekből a tanulságokat levonva, tovább tudjuk optimalizálni az anyagiak kezelését.

A webalkalmazást egy C# alapú keretrendszerrel (ASP.NET Web App, Razor Pages) valósítottam meg, amely egy adatbázissal is természetesen összeköttetésben áll. A fejlesztés során végig törekedtem a szabványos és jól strukturált felépítésre, a Clean Code elveinek, valamint az adatbázistervezés bevált gyakorlatainak követésére. A felhasználói felület modern és letisztult megjelenést kapott, így nem csupán az élményre helyeztem a hangsúlyt, hanem az alkalmazás jövőbeni továbbfejleszthetőségére, a kód olvashatóságára és könnyű karbantarthatóságára is.

## 5. fejezet

### Továbbfejlesztési lehetőségek

---

# Köszönetnyilvánítás

—

# Ábrák jegyzéke

2.1. Screenshot: Bejelentkező felület . . . . .	5
2.2. Screenshot: Profil felület . . . . .	6
2.3. Screenshot: Expense Tracker felület . . . . .	7
3.1. Projekt struktúra . . . . .	9
3.2. Pages struktúra . . . . .	12

# Táblázatok jegyzéke

2.1. Fiók műveletek . . . . .	5
2.2. Profil oldal . . . . .	6
2.3. Expense Tracker oldal . . . . .	7
2.4. Savings oldal . . . . .	7
2.5. Reports oldal . . . . .	7

# Forráskódjegyzék

3.1. Route-ok konfigurálása . . . . .	9
3.2. Frontend modell egy alkalmazása . . . . .	9
3.3. Layout extension ASP.NET keretrendszerben . . . . .	12
3.4. Layout extension ASP.NET keretrendszerben . . . . .	12
3.5. Adatbázis Connection String . . . . .	13
3.6. Program.cs: Adatbázis csatlakozás . . . . .	14
3.7. Adatbázis modell . . . . .	14