Eötvös Loránd Tudományegyetem

Informatikai Kar

Média- és Oktatásinformatika Tanszék

Pénzügyeket rendszerező webes alkalmazás

Témavezető:
Dr. Bende Imre
egyetemi adjunktus

Szerző:
Németh Zsófia Réka
programtervező informatikus BSc

Budapest, 2025

Tartalomjegyzék

1. fejezet

Bevezetés

A választott témám egy személyes pénzügyeket rendszerező webalkalmazás. A dolgozat a full-stack webfejlesztés témaköréhez kapcsolódik, mivel frontend-, backend- és adatbáziskezelést is magában foglal. A backendet és a frontendet egy projekt részeként, egy C# alapú ASP.NET Web App (Razor Pages) (.NET 8) keretrendszerrel valósítottam meg, amelyhez egy MySQL relációs adatbázist csatoltam az Entity Framework segítségével. A .NET lehetőséget nyújt különböző frontend keretrendszerek használatára is, azonban a webalkalmazásomat jelenleg egy egyszerű HTML/CSS/JavaScript kombináció alkotja, Bootstrap (CSS framework) elemek felhasználásával.

Az alkalmazás fő lényege, hogy a felhasználó költéseit és bevételeit nyomon tudja követni, megtakarításait kezelhesse akár egyénileg, akár csoportokban, mindezt egy felhasználóbarát felületen. Az alkalmazás az alap funkciókon kívül még kimutatásokat is készít a felhasználó pénzkezelési szokásairól, melyeket akár PDF formátumban is le lehet tölteni.

Véleményem szerint a már erre a célre létrehozott megoldások egy része túlságosan leegyszerűsített, másik része pedig annyira összetett és funkciógazdag, hogy hétköznapi felhasználók számára nehezen átláthatóvá válik. Jelen alkalmazás ezt az ellentétet próbálja kiegyensúlyozni, egy egyszerű, de funkcionális megoldást kínálva.

A következő fejezetekben részletesen bemutatom a projektet 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, melyeket utána egy összefoglaló - rendezhető és kereshető - adattáblázatban lehet látni. Az alkalmazás készít ezekről a pénzügyi szokásokról kimutatásokat, melyeket PDF formátumba is lehet exportálni. Az alap funkciókon kívül még kialakításra került egy megtakarításokkal foglalkozó oldal, ahol megadható tetszőleges mennyiségű pénzügyi gyűjtés, és utána ezeket lehet kezelni és folyamatosan monitorozni. Az imént felsorolt valamennyi funkció nem csupán egyéni felhasználásra áll rendelkezésre, hanem csoportok számára is. Csoportokból is tetszőleges mennyiségűt lehet létrehozni.

2.1. Rendszerkövetelmények

Mivel egy webes alkalmazásról van szó, ezért különleges gépigény nem szükséges. Szinte az összes böngésző támogatott, ahogyan azt a 2.1. táblázat is mutatja.[1]

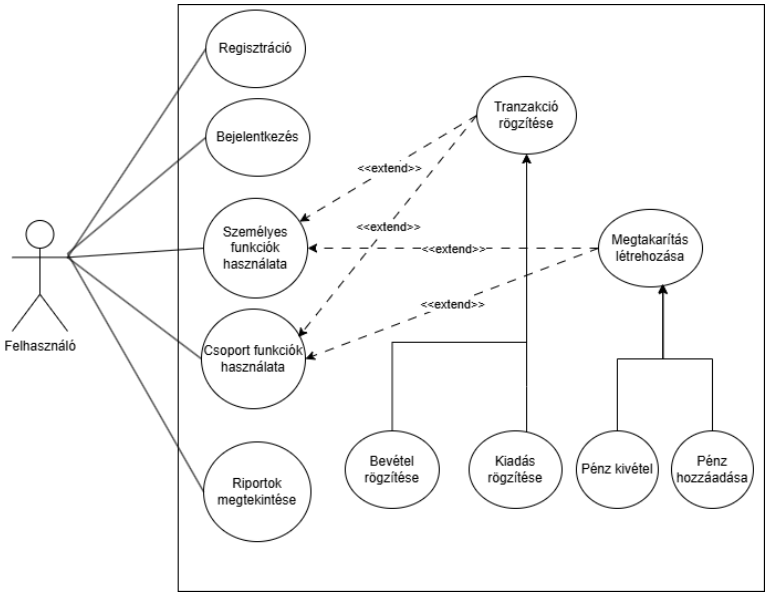
[1]Megjegyzés: Mivel a Microsoftnak nem található hivatalos adata a böngészők legrégebbi támogatott verzióira (a legújabb támogatott verzió ezek közül mindig a "current", vagyis az éppen legfrissebb), ezért a táblázat csupán egy általános modern webapplikáció böngésző kritériumait mutatja.

Böngésző	Minimum Verzió	Támogatás
Google Chrome	70+	Teljes
Mozilla Firefox	68+	Teljes
Microsoft Edge (Chromium)	79+	Teljes (a régi Edge (EdgeHTML) nem támogatott)
Safari	13+	Teljes
Opera	57+	Teljes
Internet Explorer	Nem alkalmazható	Nem támogatott

2.1. táblázat. Böngésző támogatás

2.2. Felhasználói esetek

Az alkalmazás felhasználói eseteit ez a bejegyzés fogja részletesen taglalni, képernyőképekkel kiegészítve. A felhasználói esetek bemutatását a 2.1. ábra mutatja be, és az utána következő fejezetek fejtik ki.



2.1. ábra. Használati eset diagram: Általános

2.2.1. Fiók

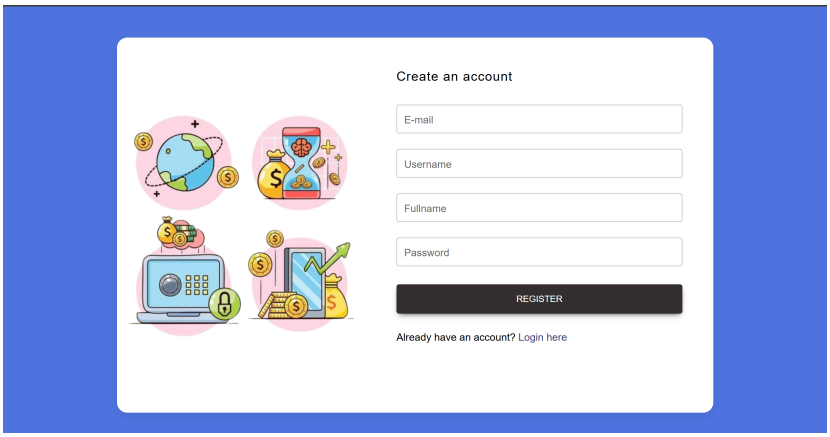
A felhasználói fiókok kezelése a szokásos Regisztráció – Bejelentkezés – Profil funkciókra épül. A felhasználónak a webalkalmazás elérése érdekében regisztrálnia kell, majd sikeres regisztrációt követően bejelentkezhet a felületre. A belső oldalra érve elérhető egy "Profil" oldal a bal menüsávból is, illetve a jobb felső sarokban az avatar ikonra kattintva a legördülő menüből is kiválasztható ez a menüpont. Szintén

ezen a két helyen találjuk a kijelentkezés gombot is, amellyel megszüntethetjük az adott munkamenetet és kiléphetünk a fiókból. Ezen funkciók részletes leírását a 2.2. táblázat foglalja össze.

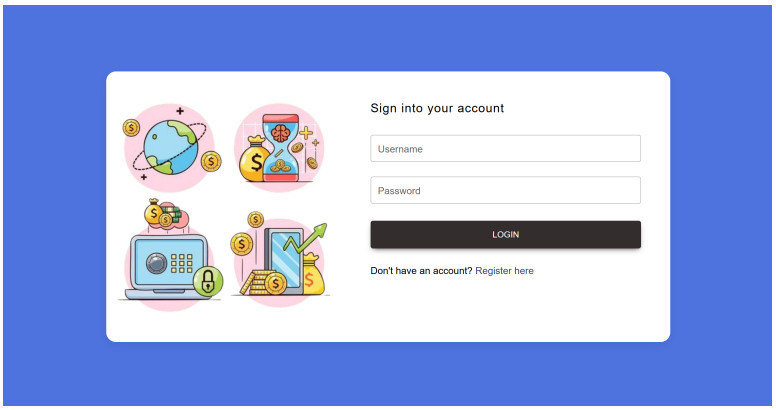
Fiók kezelés

Funkció	Leírás
Bejelentkezés	Bejelentkezés egy egyedi felhasználónévvel és egy jelszóval lehetséges. (2.3. ábra)
Regisztráció	Regisztrálni lehet az oldalra a következő adatok megadásával: e-mail cím, felhasználónév (egyedi), teljes név, jelszó. Jelszó kritériumok: legalább 8 karakter, nagybetűt, kisbetűt, számot és egy speciális karaktert is tartalmaznia kell. (2.2. ábra)
Profil	A Profil oldalon lehet a fiókhoz tartozó adatokat módosítani: a felhasználónevet, a teljes nevet, illetve az e-mail címet is. Itt elérhető még egy "Change Pass" gomb is, amely elnavigálja a felhasználót a jelszó megváltoztató felületre. A profil oldal használatát a 2.3. táblázat fejti ki bővebben. (2.4. ábra)
Jelszó módosítás	A Profil oldalról lehet a jelszó változtató felületet elérni a "Change Pass" gomb megnyomásával. Itt meg kell adni a következő adatokat: régi jelszó, új jelszó, új jelszó megerősítése.
Kijelentkezés	Ha a felhasználó befejezte kívánt tevékenységét, akkor a bal oldali menüsáv alján található "Logout" gomb megnyomásával tud kijelentkezni. A kijelentkező gomb még elérhető a jobb felső sarokban található ember ikonra kattintva legördülő menüben is.

2.2. táblázat. Fiók műveletek



2.2. ábra. Screenshot: Regisztrációs felület

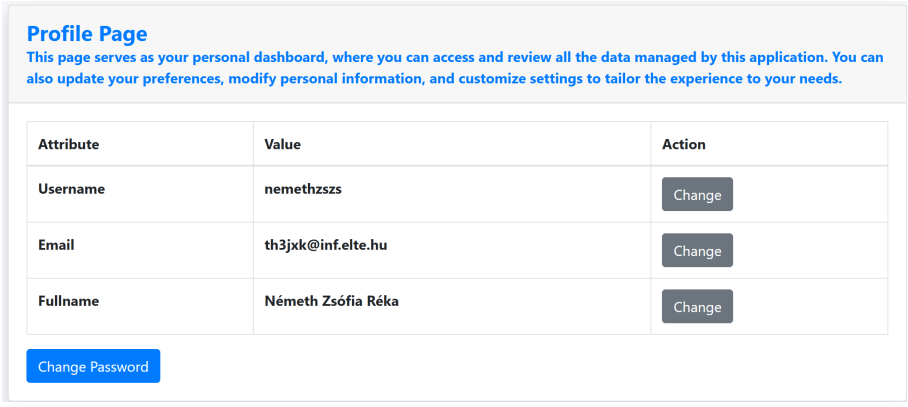


2.3. ábra. Screenshot: Bejelentkező felület

Profil

Funkció	Leírás
Adatok megtekintése	A regisztrációnál megadott személyes adatait a felhasználó itt tekintheti meg.
Adatok módosítása	A személyes adatok módosítására is itt van lehetőség, egyszerű beviteli mezőkkel lehet módosítani a felhasználónevet (egyedi), teljes nevet és e-mail címet. A "Change" gombra kattintva előugrik egy beviteli mező egy "Save" és "Cancel" gombbal kiegészítve. Az előbbi megpróbálja végrehajtani a kért módosítást, és jelzi az esetlegesen fellépő hibát (érvénytelen e-mail cím, foglalt felhasználónév, stb.) a felhasználó felé.
Jelszó módosítás	Ha a felhasználó módosítani kívánja a jelszavát, akkor a "Change Password" gombra kattintva megjelenik egy új felület, három jelszó beviteli mezővel (régi, új, új megerősítése). Jelszó kritériumok: legalább 8 karakter, nagybetűt, kisbetűt, számot és egy speciális karaktert is tartalmaznia kell.

2.3. táblázat. Profil oldal



2.4. ábra. Screenshot: Profil felület

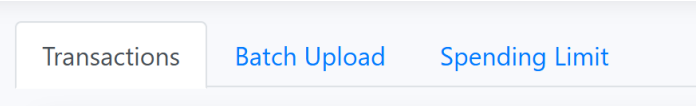
2.2.2. Pénzügyek

A pénzügyeket kezelő oldalakat a kezdőképernyőről, illetve a baloldalon lévő menüből érhetjük el. A "Finance" cím alatt 3 különböző lap került kialakításra.

- Tracker - Bevételek és kiadások vezetése, havi költési limit kezelése és előzmények megtekintése
- Savings - Megtakarítások megtekintése és kezelése
- Reports - Kimutatások megtekintése, személyre szabása és exportálása

Tracker felület

A Tracker felület foglalja magában a bevételek és kiadások vezetését, havi költési limit módosítását és monitorozását, illetve az előzmények megtekintését és korábbi tranzakciók eltávolítását. Az oldal 3 különböző lapból áll, ezek között lehet navigálni az oldal tetején lévő gombok segítségével (2.5. ábra).



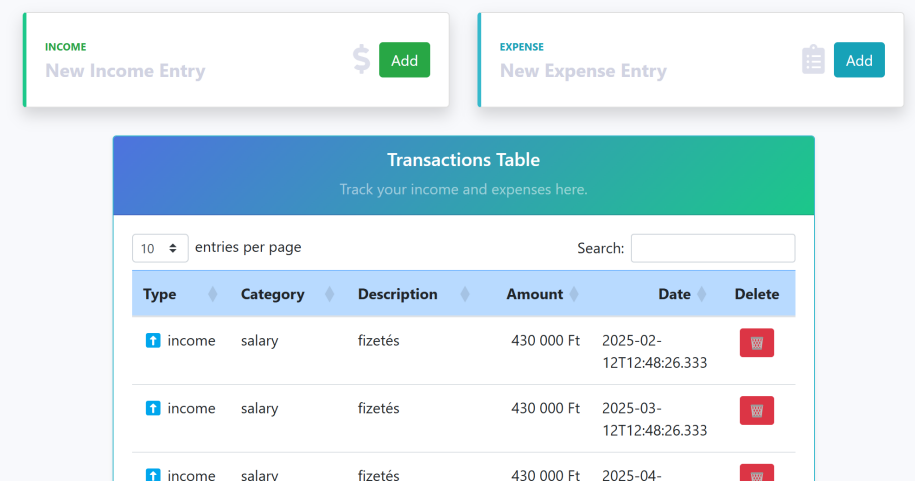
2.5. ábra. Screenshot: Tracker felület navigációs sáv

- Transactions (2.4. táblázat és 2.6. ábra) (Bevételek és kiadások vezetése, és előzmények megtekintése)
- Batch Upload (2.5. táblázat és 2.7. ábra) (Tömeges bejegyzés-feltöltés)
- Monthly Spending Limit (2.6. táblázat és 2.8. ábra) (Költési limit kezelése)

Transactions A tranzakciókkal kapcsolatos funkciók leírását a 2.4. táblázat tartalmazza.

Funkció	**Leírás**
Kiadás/bevétel rögzítése	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. Ha kiadást rögzítünk, akkor egy felugró ablak jelzi minden alkalommal, hogy mennyi maradt még a havi költési limitből, illetve ha már túl lett lépve, akkor egy figyelmeztető ablak ugrik fel.
Előzmények	A költési és bevételi előzményeket is itt lehet megtekinteni egy összesítő táblázatban. A táblázatban felül a "Search" mező módosításával lehet szűrni címszó alapján, azaz tudunk bármire (összegre, címre, kategóriára, stb.) keresni. Illetve az oszlopok címeire kattintva rendezni is lehet őket. A táblázat utolsó ("Delete") oszlopa arra szolgál, hogy egy adott sorban a kuka ikonra rákattintva, az a bejegyzés törlődik.

2.4. táblázat. Transactions lap



2.6. ábra. Screenshot: Transactions lap

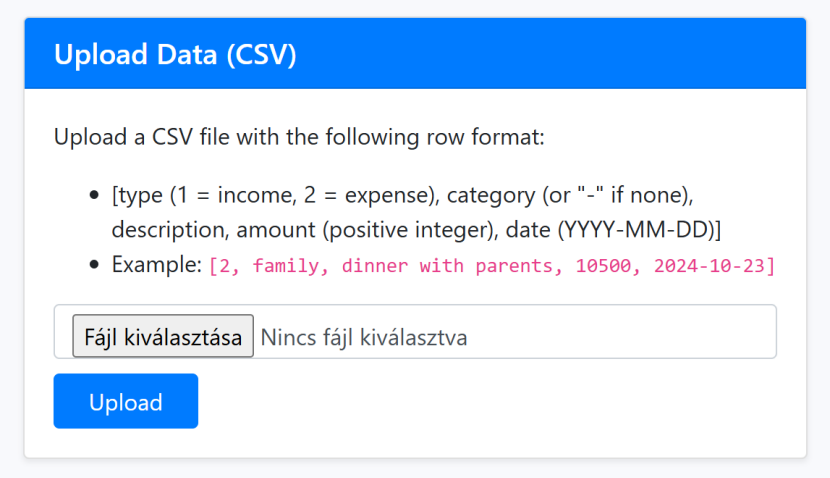
Batch Upload A tömeges feltöltéssel kapcsolatos funkciókat a 2.5. táblázat tartalmazza.

Funkció	**Leírás**
CSV feltöltés	A felhasználó számára lehetőség adott tömeges tranzakció bejegyzés feltöltésére. Egy ("," vagy ";" karakterekkel tagolt) .csv kiterjesztésű fájl feltöltésével automatikusan rögzítésre kerülnek a benne szereplő tranzakciók. A fájl helyes felépítése a 2.1. forráskód ábrán látható. Minden mező kitöltése kötelező, illetve kategóriából csak a felületen már létező kategória érték adható meg. Az általános sor struktúra: [típus (1 = bevétel, 2 = kiadás), kategória (vagy "-" ha nem kívánja megadni), leírás, összeg (pozitív egész szám), dátum (formátum: YYYY-MM-DD)]

2.5. táblázat. Batch upload lap

```
2, family, vacsi petivel, 10500, 2024-10-12
2, family, botanikus kert, 12300, 2024-11-15
1, -, ruha, 4500, 2024-09-24
```

2.1. forráskód. Tömeges feltöltés: csv fájl struktúra

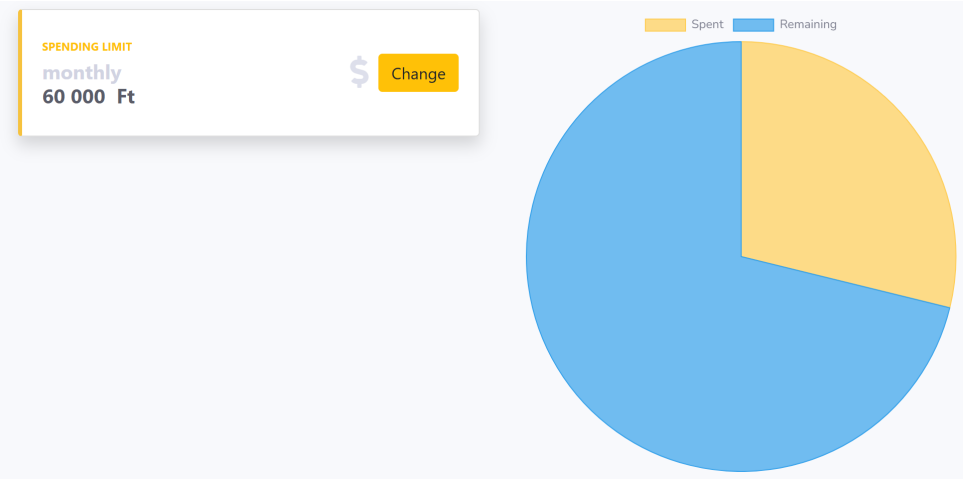


2.7. ábra. Screenshot: Tömeges feltöltés lap

Spending Limit A havi költési limittel kapcsolatos funkciók leírását a 2.6. táblázat tartalmazza.

Funkció	Leírás
Havi költési limit módosítása	Itt tudja a felhasználó a havi költési limitjét módosítani. A "Change" gombra kattintva előugrik egy beviteli mező és egy "Save" (mentés) gomb.
Diagram	A költési limit vizuális nyomon követését segíti a lapon elhelyezett kördiagram, amely egyszerű módon mutatja, hogy mennyit költött már a felhasználó a limitből.

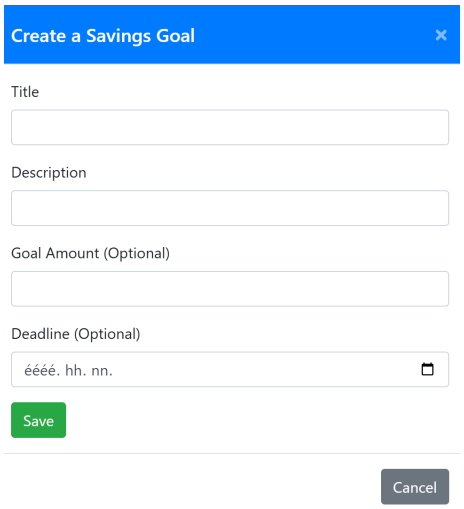
2.6. táblázat. Monthly Spending limit lap



2.8. ábra. Screenshot: Monthly Spending Limit lap

Megtakarítás felület

Gyűjtés létrehozása A felhasználó új megtakarítási célt hozhat létre, például egy konkrét vásárlás vagy vészhelyzeti alap létrehozásához. A cél tartalmazhat megnevezést, opcionális leírást, célösszeget és határidőt. A cél a főoldalon egy külön kártyaként jelenik meg. Egy új gyűjtést a 2.9. ábrán látható felület kitöltésével tud hozzáadni a felhasználó, amely az "Add new saving goal" gomb megnyomásával ugrik fel.

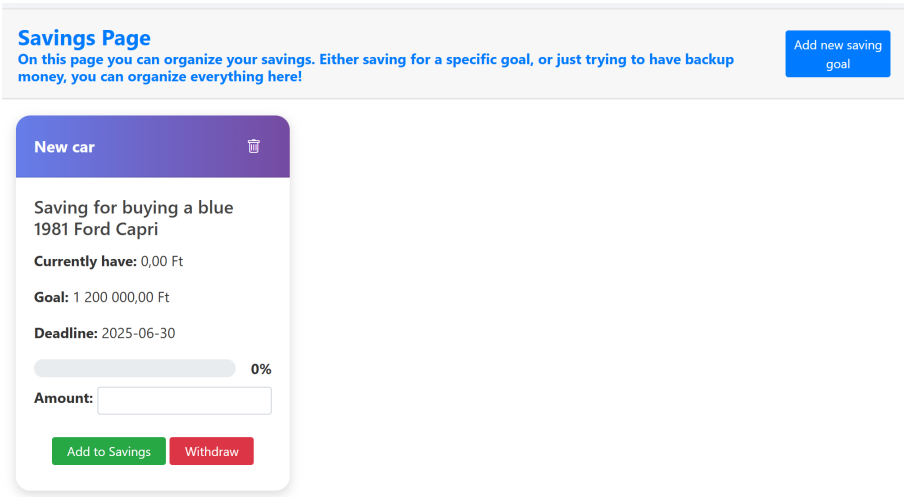


2.9. ábra. Screenshot: Gyűjtés létrehozása

Gyűjtések kezelése A gyűjtéseket az egyes kártyákon belül elhelyezett gombok, és beviteli mezők segítségével lehet kezelni. Ezeket a funkciókat a 2.7. táblázat.

Funkció	**Leírás**
Összeg hozzáadása a gyűjtéshez	Lehetővé teszi, hogy a felhasználó megtakarított összeget adjon hozzá egy kiválasztott célhoz. A kártyán található beviteli mező és "Add to Savings" gomb segítségével történik a művelet. Az aktuális megtakarítás értéke és a haladás százalékos formában is megjelenik.
Összeg kivonása a gyűjtésből	Amennyiben a felhasználó el kíván távolítani egy összeget a gyűjtésből (pl. téves bevitel miatt), azt megteheti a "Withdraw" gomb segítségével. Ez a funkció csökkenti a gyűjtött összeget, de nem törli a célt.
Gyűjtés törlése	A gyűjtés jobb felső sarkában elhelyezett kuka ikon segítségével a felhasználó teljesen eltávolíthat egy megtakarítási célt. Ez véglegesen törli az adott kártyát és a hozzá tartozó adatokat.
Cél elérése vizualizáció	Ha a megtakarított összeg eléri vagy meghaladja a kitűzött célt, a kártya vizuálisan zöld színre vált.

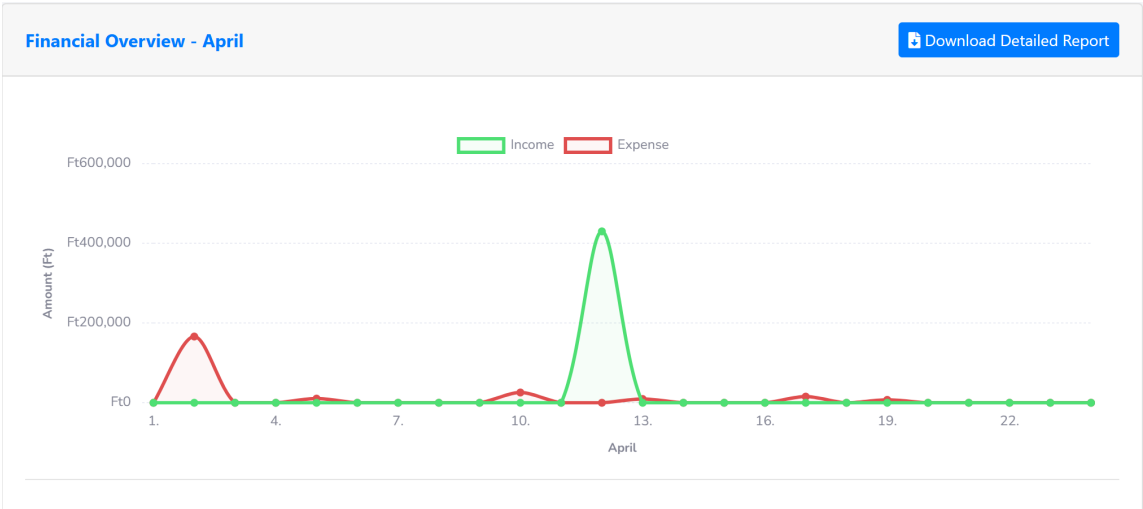
2.7. táblázat. A Savings oldal



2.10. ábra. Screenshot: Savings felület

Kimutatások felület

Havi riport Az oldal első szekciója egy vonaldiagramot jelenít meg (2.11. ábra), amely napi bontásban mutatja az adott hónap bevételeit és kiadásait. A diagram segít a felhasználónak átlátni a pénzmozgásokat, észrevenni a kiugró értékeket. A színek jól elkülönítik a bevételt (zöld) és a kiadást (piros).

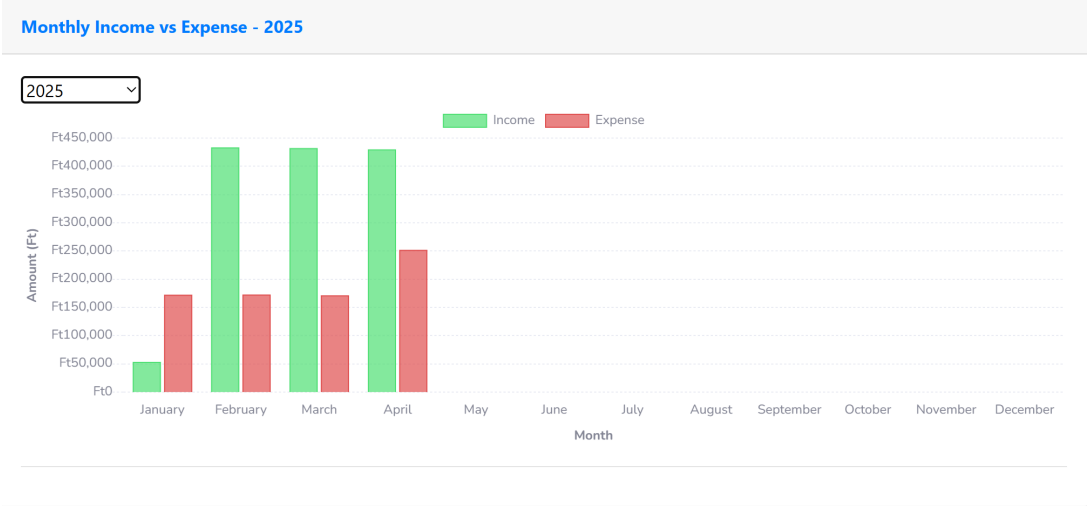


2.11. ábra. Screenshot: Aktuális havi riport

Letöltés A "Download Detailed Report" gomb lehetőséget ad arra, hogy a felhasználó exportálja a részletes pénzügyi riportját az adott hónapra (PDF formátumban). Ez hasznos lehet adminisztrációhoz vagy hó végi összesítéshez.

Éves riport Egy oszlopdiagram (2.12. ábra) összehasonlítja az egyes hónapok bevételi és kiadási értékeit. A felhasználó egy legördülő listából kiválaszthatja az

évet (de alapértelmezetten az idei év adatai láthatóak), amely alapján frissül a diagram. Ez segít az éves pénzügyi trendek nyomon követésében.



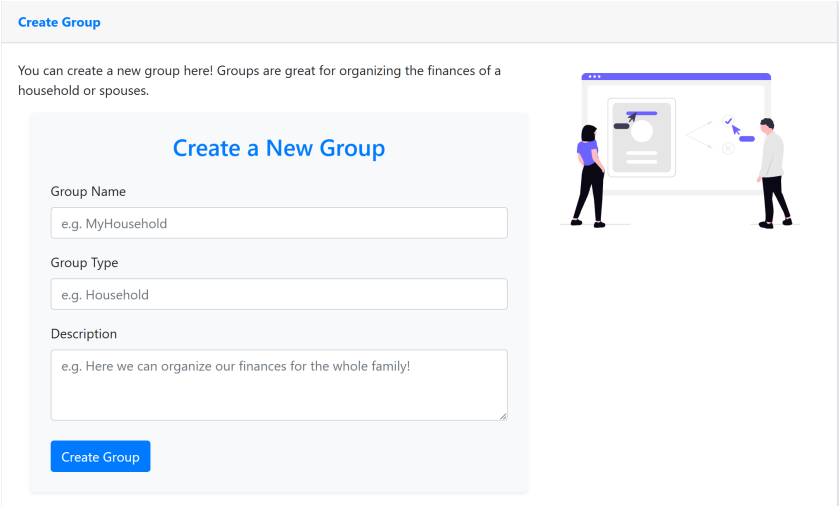
2.12. ábra. Screenshot: Éves összesítő riport

2.2.3. Csoportok

A csoportokban valamennyi funkció elérhető, melyek a személyes használat esetében is. Ezeket az előző fejezetek taglalták részletesebben. A következő bekezdésben a csoportok kezelését és elérését fogom bemutatni.

Csoport létrehozása

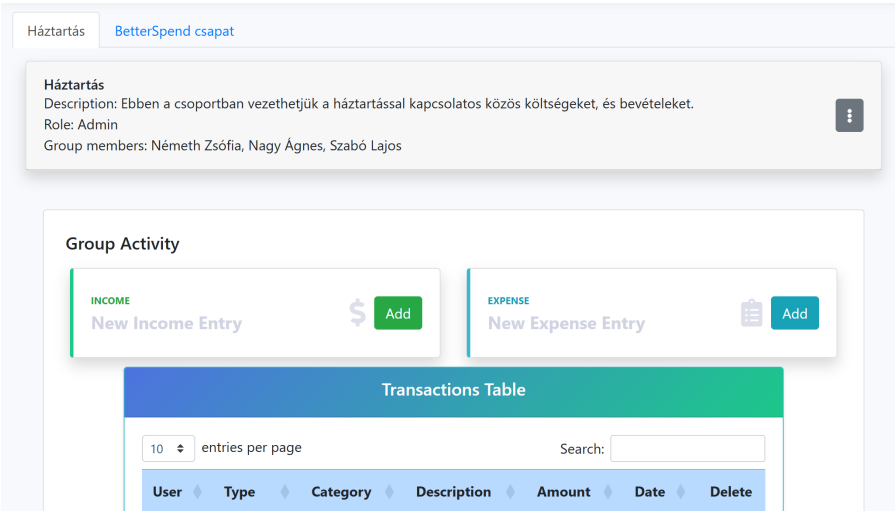
Az 2.13. ábrán látható felület a baloldali menüsávból érhető el ("Create Group" menüpont). Itt a következő adatok megadása szükséges: Title (csoport neve), Type (csoport típusa, ez csak a felhasználó számára egy rendszerezési lehetőség, jelentősége nincsen), Description (csoport részletesebb leírása). Ezután a "Create Group" gombra kattintva már kész is a csoport.



2.13. ábra. Screenshot: Create Group oldal

Saját csoportok

Megtekintés Az 2.14. ábrán látható felület a baloldali menüsávból érhető el ("My Groups" menüpont). Az oldalon belül minden csoport külön lapokon tekinthető meg, ezek között lehet navigálni az oldal tetején lévő gombok segítségével. Minden gomb felirata az adott csoport neve. A lapok tetején láthatóak az alap adatok, mint a csoport neve, leírása, tagok és a felhasználó szerepköre ("Admin" = ha ő hozta létre, "Member" = azaz "tag", ha a felhasználó csak meghívva lett a csoportba)

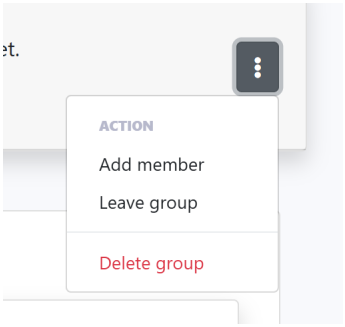


2.14. ábra. Screenshot: My Group oldal

Funkciók A csoportos funkciók megegyeznek a személyes funkciókkal (bevételek/-kiadások rögzítése és törlése, előzmények megtekintése, riport megtekintése,

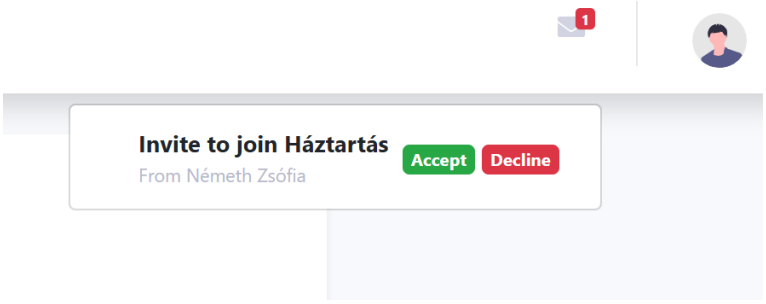
megtakarítások létrehozása és kezelése). Ezek részletes bemutatásra már megtörtént korábbi fejezetekben.

Kezelés Egy adott csoport fejlécének jobb oldalán található 3 pont megnyomása egy leugró menüt nyit meg. Itt lehet meghívót küldeni a csoportba egy másik felhasználónak ("Add member"), elhagyni a csoportot ("Leave group"), vagy ha Admin szerepkörrel rendelkezik az adott felhasználó akkor törölni is lehet a csoportot.



2.15. ábra. Screenshot: Screenshot: Csoport műveletek

Meghívó A meghívó küldése után a meghívott felhasználónak érkezni fog egy értesítése (mely az oldal tetején lévő levél ikon megnyomásával válik láthatóvá), amely tartalmazza a következő információkat: csoport neve, felhasználó teljes neve, aki meghívta oda. Itt az 2.16 ábrán látható módon egy "Accept" (elfogadás) és egy "Deny" (elutasítás) gomb található. Ha elfogadja a felhasználó a meghívást, akkor tagja lesz a csoportnak.



2.16. ábra. Screenshot: Értesítések

3. fejezet

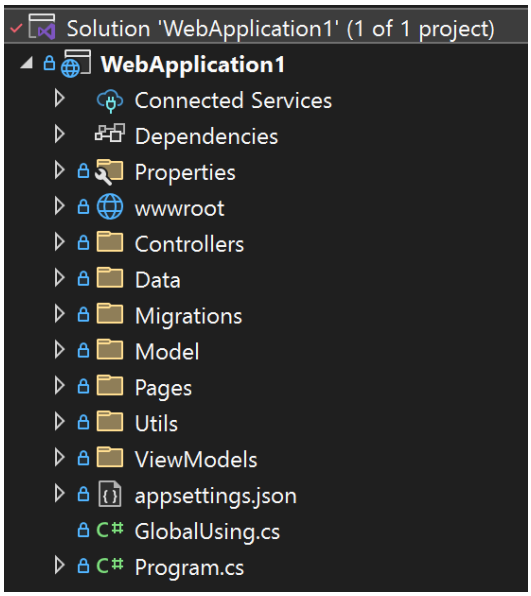
Fejlesztői dokumentáció

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[1]/CSS/JavaScript (és Bootstrap)
- Backend: C# (ASP.NET Web App (Razor Pages))
- Adatbázis: MySQL relációs adatbázis

[1]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. Külső függőségek, technológiák

Csomagok hozzáadása a projekthez történhet a NuGet Package Manager vagy CDN segítségével is. Az alkalmazás fejlesztése során az alábbi külső csomagok és könyvtárak kerültek felhasználásra:

- **Entity Framework Core (EF Core)** – ORM technológia, amely lehetővé teszi az adatbázissal való objektumorientált munkát. Kiemelten hasznos a LINQ támogatása és a migrációs rendszer.[2]
- **Pomelo.EntityFrameworkCore.MySql** – Az EF Core és a MySQL adatbázis közötti kapcsolatot biztosító NuGet csomag.
- **MySqlConnector** – Könnyűsúlyú, aszinkron támogatással rendelkező ADO.NET driver MySQL-hez.
- **Swashbuckle.AspNetCore** – A Swagger UI integrálására szolgáló csomag, amely lehetővé teszi az API-k dokumentálását és tesztelését.
- **Bootstrap** – Reszponzív front-end keretrendszer, amely előre definiált CSS osztályokat és JavaScript komponenseket biztosít a felhasználói felület egységes és gyors kialakításához. Az alkalmazás a Bootstrap 4-es verzióját használja, amely lehetővé teszi különböző vizuális elemek (például gombok, tábláza-

[2] A migrációk működéséről az adatbázissal foglalkozó fejezet fog részletesebb leírást nyújtani.

tok, rácsszerkezetek) egyszerű alkalmazását csupán megfelelő class attribútumok beállításával. Nem csak az esztétikus megjelenésben nyújt segítséget a Bootstraom hanem interaktív viselkedések (pl. modális ablakok, lenyíló menük) hozzáadásában is, beépített JavaScript funkciók segítségével.

- **FontAwesome (CDN)** – Ikonok használatához alkalmazott ikoncsomag.
- **QuestPDF** -Az alkalmazás lehetőséget nyújt az adatok exportálására nyomtatható formátumban, egy PDF fájl formájában. Erre a célra a QuestPDF nevű nyílt forráskódú .NET könyvtárat használtam.

  A QuestPDF lehetővé teszi professzionális minőségű PDF fájlok generálását teljes mértékben C# nyelvben, deklaratív stílusban. Előnyei:

  – Reszponzív dokumentumelrendezés
  – Könnyen tanulható, C#-os fluent API
  – Támogatja a táblázatokat, stílusokat, színeket, képeket

  A generálás egy Document.Create(...)'hívással történik, ahol meghatározásra kerül a dokumentum szerkezete, stílusa, tartalma. Például:

```
1    Document.Create(container =>
2    {
3      container.Page(page =>
4      {
5        page.Content()
6        .Column(col =>
7        {
8          col.Item().Text("Kiadasi osszesito");
9          col.Item().Text("Osszes tranzakcio: 42");
10       });
11     });
12   }).GeneratePdf("output.pdf");
```

3.1. forráskód. PDF generálás példa

A PDF fájl generálása szerveroldalon történik, a Utils/PDFGenerator.cs osztályban megvalósított logika segítségével. A kész fájl ezután egy HTTP végpont (Controller) segítségével érhető el és tölthető le a felhasználó által.

```
[HttpGet("DownloadReportForCurrent")]
public async Task<IActionResult> DownloadReport()
{
  // get data

  var pdf = new PdfGenerator(user.Fullname, transactions);
  var pdfBytes = pdf.GeneratePdf();

  return File(pdfBytes, "application/pdf", $"report_{user.
      Fullname}_{DateTime.Now:yyyyMMdd}.pdf");
}
```

3.2. forráskód. PDF generálás API végpont

- **Chart.js** – Az alkalmazás az adatok vizuális megjelenítésére a népszerű JavaScript könyvtárat használja, amely interaktív, reszponzív grafikonokat képes kirajzolni HTML canvas elemeken keresztül.

  Két típusú diagram került megvalósításra, melyek közül az utóbbi csoportokban nem elérhető:

  – Napi bontású bevétel és kiadás alakulása az aktuális hónapban (vonaldiagram)

  – Havi bontású bevétel–kiadás összehasonlító diagram (oszlopdiagram)

  A grafikonok kliensoldalon, JavaScript segítségével jönnek létre. Az adatok aszinkron módon kerülnek lekérésre az API végpontokról, a felhasználó azonosítója alapján. A lekért tranzakciós adatok formázása és csoportosítása után a rendszer automatikusan diagramot generál azok vizuális megjelenítésére.

  Az alábbi példában látható egy egyszerűbb diagram kirajzolása:

```
const ctx = document.getElementById("finance-chart");
// <canvas> elem
const monthlySpendingChart = new Chart(ctx, {
  type: 'bar',
  data: {
    labels: months, // Jan - Dec
    datasets: [{
      label: "Income",
```

```
9           data: incomeData // from backend
10        }]
11      }
12    });
```

3.3. forráskód. Chart.js alapú havi diagram kirajzolása

3.1.2. Architektúra leírása

Ahogy a 3.1. ábrán is látható, az egész webalkalmazás egy projekten belül lett kialakítva. 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[3] fájlstruktúrája alapján automatikusan létrejönnek a route-ok, ha a "Program.cs"-ben megadjuk a programnak, hogy hozza létre őket:

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

3.4. 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>
```

3.5. forráskód. Frontend modell egy alkalmazása

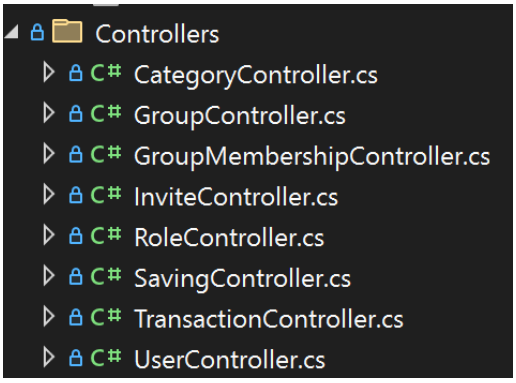
Itt ugye mint látható a modellből tudunk adatot lekérdezni. De mi is a "Model" pontosan? Ugyebár az ASP .NET Web App keretrendszer úgy működik, hogy minden oldal (Razor page) egy .cshtml és egy .cs fájl együtteséből áll össze. A .cs kiterjesztésű fájl az adott oldal modellje. Itt definiálhatunk adatszerkezeteket és függvényeket, mint például az OnGet() és OnPost(), amelyek beépített (opcionális)

[3]A "Pages" mappa, ahogy a neve is mutatja, tartalmazza a weboldal egyes oldalait. Bővebb kifejtés a Frontend című fejezetben.

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. Frontend

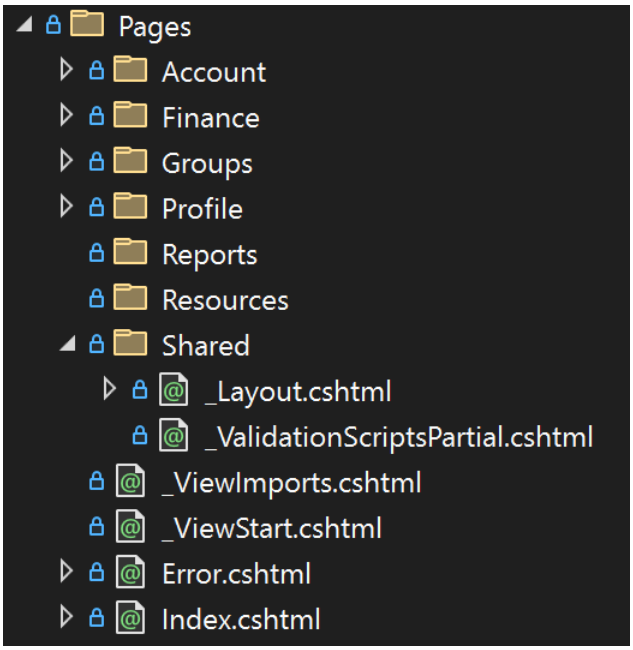
Térjünk át a frontend réteg alkotóelemeire, és ezek működésére. Itt is komponensenként fogom bemutatni a felhasznált technológiákat.



3.2. ábra. Pages struktúra

3.2.1. 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 (3.3. ábra) látható tehát, hogy például a bejelentkező felületet a következő linken tudjuk elérni: "localhost:«port»/Account/Login".



3.3. 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:

```
<div class="container-fluid">
@RenderBody()
</div>
```

3.6. 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:

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

3.7. forráskód. Layout extension ASP.NET keretrendszerben

3.2.2. Statikus erőforrások kezelése

A `wwwroot` mappa szolgál az alkalmazás összes statikus erőforrásának (static resources) tárolására. Ide kerülnek azok a fájlok, amelyeket a kliensoldal közvetlenül elérhet, például JavaScript állományok, CSS (stílus formázás) fájlok, képek vagy betűtípusok. A mappa struktúráját (a szokásokhoz híven) a következőképpen alakítottam ki:

- `js` – JavaScript fájlok
- `css` – CSS fájlok
- `img` – Képek (ikonok, háttérképek stb.)

Az alkalmazás formázásához a Bootstrap keretrendszer 4-es verzióját használtam. A Bootstrap szükséges CSS és JavaScript fájljait közvetlenül letöltve helyeztem el a `wwwroot` mappában, így külső CDN-től való függőség nélkül működik az alkalmazás. Egyetlen kivételt képez egy betűtípus, amely CDN-en keresztül kerül betöltésre.

A `wwwroot` mappában elhelyezett fájlok az alkalmazás nézeteiből (`.cshtml` fájlokból) egyszerűen beilleszthetők az alábbi módon:

```
1  <link rel="stylesheet" href="~/css/style.css" />
2  <script src="~/js/site.js"></script>
```

3.8. forráskód. Statikus fájlok hivatkozása .cshtml fájlban

A `"~"` jelölés a gyökérmappára hivatkozik, így biztosítva a helyes útvonalat (minden környezetben).

3.3. Backend

A következő fejezetben az alkalmazás backend architektúráját mutatom be, kifejtve az egyes elemek funkcióit és használatát.

3.3.1. 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 bármely részén könnyen ki lehet olvasni (szótár szintaktika használatával) (pl. Configuration["Kulcs"]). Ez segít elkülöníteni a kódot a konfigurációtól, így könnyebb a karbantartás, és egyszerűen kezelhető a környezetenkénti eltérés (pl. fejlesztői vs. éles környezet).

3.3.2. Program.cs

Ez a fő program fájl, és egyben az alkalmazás belépési pontja. Ebben készítjük elő a webalkalmazásunk tulajdonságait (persze a Razor Pages 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.

3.3.3. 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.

Az alkalmazásban minden adatbázis tábla egy hozzá tartozó C# osztállyal, azaz modellel van leképezve. Ezek az osztályok tartalmazzák az adott tábla mezőit, mint property-ket, valamint opcionálisan kapcsolatok (navigációs tulajdonságok) is definiálhatók a táblák között, például egy tranzakcióhoz tartozó felhasználó vagy kategória esetén.

A modellek létrehozása kulcsfontosságú az Entity Framework Core működéséhez, mivel ezek alapján tudja az ORM keretrendszer leképezni az adatbázis szerkezetét, valamint automatikusan kezelni a lekérdezéseket, beszúrásokat és módosításokat. A modellek az `ApplicationDbContext` osztályban kerülnek regisztrálásra, amely az adatbázis-kapcsolatot is kezeli.

Például egy egyszerű `Transaction` modell így nézhet ki:

```
1  public class Transaction
2  {
3    public int Id { get; set; }
```

```
4     public int UserId { get; set; }
5     public decimal Amount { get; set; }
6     public DateTime Date { get; set; }
7     public string Description { get; set; }
8   }
```

3.9. forráskód. Egyszerű modell osztály példa

3.3.4. API-kezelés és végpontok kialakítása

Az ASP.NET alkalmazás szerkezete lehetővé teszi, hogy a frontend és a backend között tisztán elválasztott, jól strukturált kommunikáció jöjjön létre. Miután a `Program.cs` fájlban megtörténik a szükséges szolgáltatások regisztrálása és konfigurációja (pl. routing, dependency injection), a fejlesztés következő lépése az API-végpontok kialakítása.

Az alkalmazás `Controllers` mappájában (3.2. ábra) minden releváns adatmodellhez külön kontrollerosztály tartozik, ahol HTTP-alapú végpontokat definiálok (pl. `GET`, `POST`, `PUT`, `DELETE` metódusok). Ezeket a végpontokat a frontend JavaScript kód hívja meg aszinkron módon `fetch` API segítségével, ezáltal biztosítva a felhasználó és a szerver közötti kétirányú adatkommunikációt.

Az egyszerűbb esetekben (például egy adatmodell értékeinek megjelenítése) az ASP.NET Razor Pages lehetőséget biztosít arra, hogy közvetlenül a `.cshtml` nézetfájlban jelenítsünk meg backendből érkező adatokat a modellobjektumok segítségével. Az összetettebb működések (pl. szűrés, exportálás, állapotváltoztatás) viszont külön API-végpontokat igényelnek.

A fejlesztés során az API-dokumentáció és tesztelés megkönnyítésére a **Swagger** (Swashbuckle.AspNetCore) könyvtár került integrálásra. Ez automatikusan generál egy böngészőben elérhető dokumentációs felületet, ahol a végpontok struktúrája, paraméterei és válaszai is megtekinthetők, valamint tesztelhetők. Fontos biztonsági szempont, hogy a Swagger csak `Development` környezetben legyen elérhető, nehogy érzékeny információk szivárogjanak ki az éles rendszerből.

Példa API Controller

Az alábbi kódrészlet bemutatja, hogyan történik egy egyszerű `GET` kérés kezelése, amely egy adott felhasználóhoz tartozó tranzakciókat ad vissza.

```
[ApiController]
[Route("api/[controller]")]
public class TransactionController : ControllerBase
{
  private readonly ApplicationDbContext _context;

  public TransactionController(ApplicationDbContext context)
  {
    _context = context;
  }

  // GET: api/Transaction/user/5/type/1
  [HttpGet("user/{userId}/type/{typeId}")]
  public IActionResult GetTransactionsByUserAndType(int userId,
      int typeId)
  {
    var transactions = _context.Transactions
    .Where(t => t.UserId == userId && t.TypeId == typeId)
    .OrderByDescending(t => t.Date)
    .ToList();

    return Ok(transactions);
  }
}
```

3.10. forráskód. Egyszerű Controller példa – tranzakciók lekérdezése

A fenti példában a kontroller egy adott felhasználó és tranzakciótípus alapján szűri az adatokat, majd egy JSON-formátumú listát ad vissza, amit a frontend JavaScript-kód aszinkron módon feldolgoz. Ez az endpoint tehát tökéletesen alkalmas például diagramok vagy riportok adatainak kiszolgálására.

3.4. Adatbázis

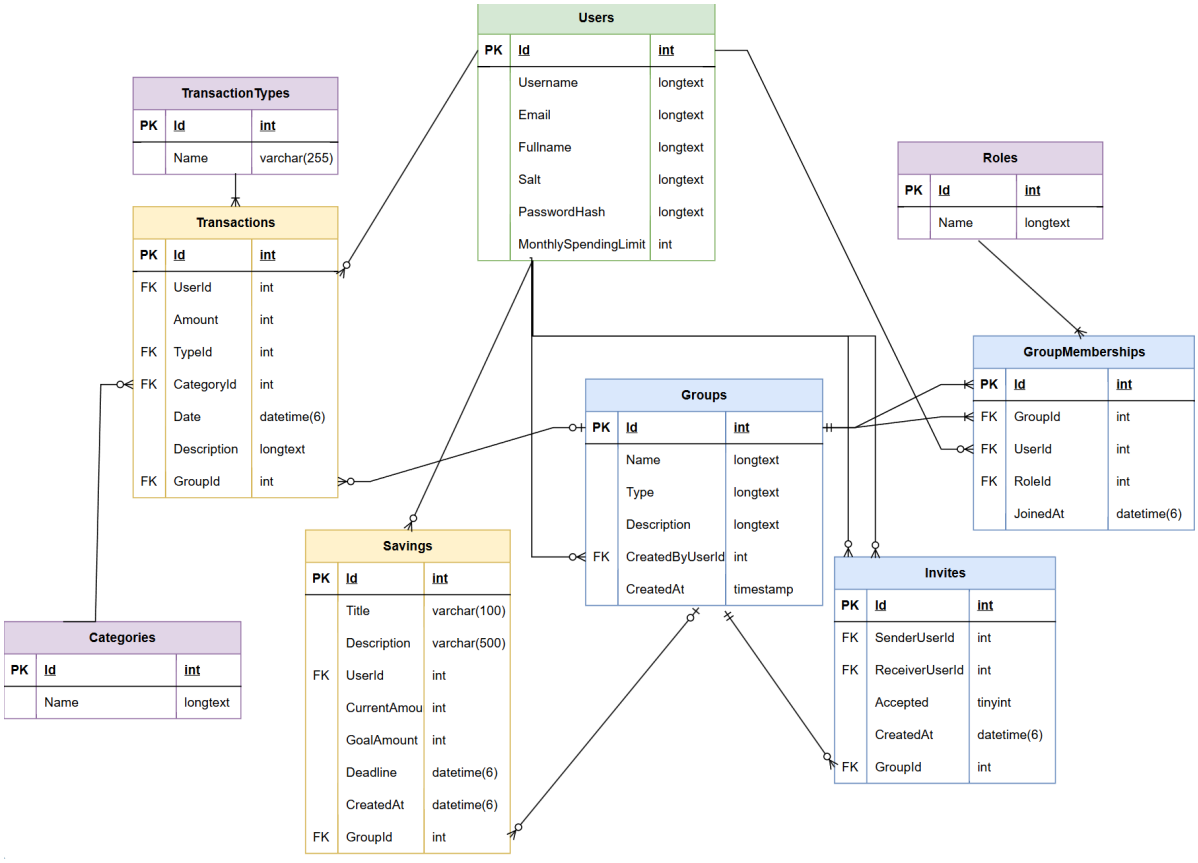
Az alkalmazás működésének egyik alapvető pillére az adatbázis, amely a rendszerben tárolt információk strukturált kezelését, hosszú távú perzisztenciáját és elérhetőségét biztosítja. Az adatbázis szerkezete úgy került kialakításra, hogy az hatékonyan támogassa az alkalmazás funkcionális igényeit, miközben jól skálázható és könnyen karbantartható maradjon.

Az adatbázis relációs modellre épül, és az adatok külön táblákban kerülnek tárolásra, melyeket kulcsok és kapcsolat típusok kötnek össze.

3.4.1. Szerkezet

Az alkalmazás adatbázisa egy localhost-on futó MySQL adatbázis. A MySQL sok szempont miatt kiváló választás kisebb alkalmazásokhoz, ezek közé tartozik például az a nem elhanyagolható indok, hogy teljesen ingyenes a használata. Emellett kiemelendő az egyszerű kezelhetősége és nagy mértékű kompatibilitása az ASP.NET keretrendszerrel, amely lehetővé teszi a zökkenőmentes integrációt az alkalmazás backendjével.

Az alkalmazáshoz 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. A szerkezetet és a logikai kapcsolatokat az egyes táblák között a 3.4. ábra szemlélteti. A táblák részletes leírását pedig a 3.1. táblázat tartalmazza.



3.4. ábra. Adatbázis struktúra

Tábla	**Leírás**
Users	A rendszer felhasználóit tartalmazza. Itt tároljuk az alapvető adatokat (felhasználónév, e-mail, teljes név), a hitelesítéshez szükséges információkat (salt és jelszó-hash), valamint a havi költési limitet is.
Groups	A csoportokat kezeli, amelyekbe a felhasználók beléphetnek. Minden csoportnak van neve, típusa (pl. család, baráti társaság, ez szabad szöveges érték), leírása, létrehozó felhasználója és létrehozási dátuma.
GroupMemberships	A felhasználók csoporttagságait rögzíti. Megmutatja, hogy ki melyik csoporthoz tartozik, milyen szerepkörben (pl. admin vagy tag), és mióta tagja a csoportnak.
Roles	Az egyes csoporttagságokhoz rendelhető szerepköröket definiálja, például adminisztrátor vagy sima tag. A szerepkörök szabályozzák a felhasználók jogosultságait a csoportban. A csoport létrehozójának "Admin" jogköre lesz automatikusan, a felvett tagnak pedig "Member".
Transactions	A felhasználók által rögzített pénzügyi tranzakciókat tárolja. Minden tranzakcióhoz tartozik összeg, típus (bevétel vagy kiadás), kategória (pl. étel, szórakozás), dátum, leírás, és adott esetben kapcsolódó csoport is. A "GroupId" értéke null, ha személyes a tranzakció.
TransactionTypes	A tranzakciók típusait (bevétel, kiadás) határozza meg.
Categories	A tranzakciókhoz tartozó kategóriákat tartalmazza, mint például élelmiszer, közlekedés, szórakozás stb. Segíti a kiadások és bevételek részletesebb elemzését.
Savings	A felhasználók által kitűzött megtakarítási célokat tárolja. A célösszeg és a határidő opcionális adatok, tehát lehet null értékük. A "GroupId" értéke is null, ha személyes megtakarítás.

Tábla	Leírás
Invites	A csoportokba történő meghívásokat kezeli. Menti, hogy ki küldött meghívót kinek, melyik csoportba, és hogy a meghívás elfogadásra került-e. Az elfogadott meghívók "Accepted" értéke 1-re vált a default 0-ról. Elutasítás esetén a meghívó törlődik a táblából.

3.1. táblázat. Adatbázis táblák magyarázata

3.4.2. Backend - Adatbázis kapcsolat

A backend és az adatbázis közötti kapcsolat kialakítása az ASP.NET keretrendszerben viszonylag egyszerűen megvalósítható. Ehhez elegendő a szükséges NuGet csomagok – például az Entity Framework Core és a MySQL Connector – telepítése. Az adatbázishoz való csatlakozáshoz mindössze egy kapcsolati karakterláncot (connection string) kell megadni az appsettings.json konfigurációs fájlban, amely tartalmazza a szerver nevét, az adatbázis elérési adatait és a hitelesítéshez szükséges információkat.

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

3.11. 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(
       connectionString));
8  });
```

3.12. forráskód. Program.cs: Adatbázis csatlakozás

A tényleges adatbázis modellje:

```
using WebApplication1.Model;

namespace WebApplication1.Data
{
  public class ApplicationDbContext : DbContext
  {
    public ApplicationDbContext(DbContextOptions<
        ApplicationDbContext> options)
    : base(options)
    {
    }

    public DbSet<User> Users { get; set; }
    public DbSet<Transaction> Transactions { get; set; }
    public DbSet<Category> Categories { get; set; }
    public DbSet<Role> Roles { get; set; }
    public DbSet<GroupMembership> GroupMemberships { get; set; }
    public DbSet<Group> Groups { get; set; }
    public DbSet<Invite> Invites { get; set; }
    public DbSet<Saving> Savings { get; set; }
    public DbSet<TransactionType> TransactionTypes { get; set; }

  }
}
```

3.13. forráskód. Adatbázis modell

A kapcsolat létrehozása után az adatbázis teljes mértékben integrálódik az alkalmazáshoz. Az Entity Framework lehetővé teszi, hogy ne közvetlenül SQL lekérdezésekkel vagy parancsokkal dolgozzunk, hanem a modell osztályokon keresztül, objektumorientált módon kezeljük az adatokat. Ezek az osztályok (pl. User, Transaction) reprezentálják az adatbázis tábláit, és LINQ segítségével egyszerűen végezhetünk lekérdezéseket és módosításokat. A keretrendszer gondoskodik arról, hogy ezek a módosítások automatikusan szinkronizálódjanak az adatbázissal.

3.4.3. Migrációk kezelése

Az alkalmazás későbbi bővítései és karbantartása során az adatbázis szerkezetének módosítása elkerülhetetlen. Ennek biztonságos és nyomon követhető megvalósí-

tásához az Entity Framework Core által biztosított migrációs rendszert alkalmaztam. A migrációk segítségével az adatmodell változásai szinkronban tarthatók az adatbázis szerkezetével, miközben lehetőség nyílik a változtatások verziókövetésére is. A működésének lényege, hogy ha módosítást végzünk az egyik adatbázistáblának megfelelő modell osztályban (például új mezőt adunk hozzá vagy megváltoztatjuk egy meglévő típusát vagy akár egy teljesen új modellt hozunk létre), akkor a migrációs rendszer segítségével ez a változás automatikusan átvezethető az adatbázis szerkezetébe is.

Egy új migráció létrehozása a következő paranccsal történik:

```
»dotnet ef migrations add <Név>
```

A migráció alkalmazása az adatbázisra pedig a következő parancs segítségével történik:

```
»dotnet ef database update
```

(vagy A Visual Studio Package Manager Console-ján belül: "Add-Migration <Név>" és "Update-Database")

Ez a megközelítés biztosítja, hogy a különböző fejlesztési vagy éles környezetekben is konzisztens adatbázis-struktúra álljon rendelkezésre.

3.5. Tesztek

Teszteset azonosító	TC001
Funkció	Regisztráció
Bemenet	Valid e-mail cím, egyedi felhasználónév, teljes név, jelszó
Elvárt eredmény	Sikeres regisztráció, átirányítás a bejelentkezés felületre, visszajelző üzenet a felhasználónak
Tényleges eredmény	Megfelel az elvárt eredménynek

3.2. táblázat. Sikeres regisztráció

Teszteset azonosító	TC002
Funkció	Regisztráció
Bemenet	Üres mezők beküldése vagy invalid e-mail cím, vagy nem elég erős jelszó
Elvárt eredmény	Hibajelzés, regisztráció sikertelen
Tényleges eredmény	Megfelel az elvárt eredménynek

3.3. táblázat. Sikertelen regisztráció

Teszteset azonosító	TC003
Funkció	Bejelentkezés
Bemenet	Helyes felhasználónév - jelszó páros megadása
Elvárt eredmény	Belépteti a felhasználót az oldalra
Tényleges eredmény	Megfelel az elvárt eredménynek

3.4. táblázat. Sikeres bejelentkezés

Teszteset azonosító	TC004
Funkció	Bejelentkezés
Bemenet	Üres mezők beküldése, vagy helytelen felhasználónév-jelszó páros
Elvárt eredmény	Hibajelzés, a bejelentkezés sikertelen
Tényleges eredmény	Megfelel az elvárt eredménynek

3.5. táblázat. Sikertelen bejelentkezés

Teszteset azonosító	TC005
Funkció	Profil oldal - adatok módosítása
Bemenet	Új, egyedi felhasználónév megadása VAGY új, valid e-mail cím megadása VAGY új teljes név megadása
Elvárt eredmény	Sikeres módosítás
Tényleges eredmény	Megfelel az elvárt eredménynek

3.6. táblázat. Személyes adatok sikeres módosítása

Teszteset azonosító	TC006
Funkció	Profil oldal - adatok módosítása
Bemenet	Helytelen e-mail cím megadás, foglalt felhasználónév megadás, üres mező beküldése
Elvárt eredmény	Hibajelzés, a módosítás nem lép érvénybe
Tényleges eredmény	Megfelel az elvárt eredménynek

3.7. táblázat. Személyes adatok sikertelen módosítása

Teszteset azonosító	TC007
Funkció	Profil oldal - Jelszó módosítása
Bemenet	Helyes régi jelszó, új jelszó, új jelszó megerősítése egyezően, új jelszó elég erős
Elvárt eredmény	Sikeres jelszóváltás, visszajelzés a felhasználónak
Tényleges eredmény	Megfelel az elvárt eredménynek

3.8. táblázat. Sikeres jelszó modósítás

Teszteset azonosító	TC008
Funkció	Profil oldal - Jelszó módosítása
Bemenet	Helytelen régi jelszó, új jelszó és megerősítés nem egyező, nem elég erős új jelszó
Elvárt eredmény	Hibajelzés, módosítás nem lép érvénybe
Tényleges eredmény	Megfelel az elvárt eredménynek

3.9. táblázat. Sikertelen jelszó módosítás

Teszteset azonosító	TC009
Funkció	Új bevétel rögzítése
Bemenet	Helyes amount (pl. 5000) és description (pl. "Fizetés")
Elvárt eredmény	Sikeres mentés, üzenet: "Transaction saved!"
Tényleges eredmény	Megfelel az elvárt eredménynek

3.10. táblázat. Új bevétel helyes adatokkal

Teszteset azonosító	TC010
Funkció	Új bevétel rögzítése
Bemenet	Üres vagy hibás amount/description mezők
Elvárt eredmény	Hibajelzés: "Failed to create transaction!" (Tranzakció mentése sikertelen.)
Tényleges eredmény	Megfelel az elvárt eredménynek

3.11. táblázat. Új bevétel hibás adatokkal

Teszteset azonosító	TC011
Funkció	Kiadás rögzítése, ha havi limit = 0
Bemenet	Új kiadás mentése 0 költési limit mellett
Elvárt eredmény	Üzenet: "Transaction saved!" (limit státusz nélkül)
Tényleges eredmény	Megfelel az elvárt eredménynek

3.12. táblázat. Kiadás rögzítése havi limit nélkül

Teszteset azonosító	TC012
Funkció	Kiadás rögzítése, ha van havi limit
Bemenet	Új kiadás mentése havi limit mellett
Elvárt eredmény	Limit státusz megjelenik: zöld (ha van keret), sárga (ha túllépés történt)
Tényleges eredmény	Megfelel az elvárt eredménynek

3.13. táblázat. Kiadás rögzítése havi limittel

Teszteset azonosító	TC013
Funkció	Előzmény táblázat ellenőrzése
Bemenet	Új bevétel/kiadás rögzítése
Elvárt eredmény	Az új tranzakció megjelenik az előzmények között
Tényleges eredmény	Megfelel az elvárt eredménynek

3.14. táblázat. Új tranzakció megjelenése az előzmények között

Teszteset azonosító	TC014
Funkció	CSV batch feltöltés
Bemenet	Helyes formátumú CSV fájl feltöltése (típus, kategória, leírás, összeg, dátum)
Elvárt eredmény	Minden sor külön tranzakcióként mentésre kerül, üzenet: "Transactions saved!" (Tranzakciók elmentve!)
Tényleges eredmény	Megfelel az elvárt eredménynek

3.15. táblázat. CSV batch feltöltés helyes fájllal

Teszteset azonosító	TC015
Funkció	CSV batch feltöltés
Bemenet	Helytelen formátumú CSV fájl feltöltése (típus, kategória, leírás, összeg, dátum)
Elvárt eredmény	Hibaüzenet: "Error! «a pontos hiba»" (Hiba!)
Tényleges eredmény	Megfelel az elvárt eredménynek

3.16. táblázat. CSV batch feltöltés helytelen fájllal

Teszteset azonosító	TC016
Funkció	Havi költési limit módosítása
Bemenet	Új poitív egész költési limit megadása (pl. 100000)
Elvárt eredmény	Sikeres módosítás, limit frissítése az adatbázisban
Tényleges eredmény	Megfelel az elvárt eredménynek

3.17. táblázat. Sikeres havi költési limit módosítása

Teszteset azonosító	TC017
Funkció	Havi költési limit módosítása
Bemenet	Érvénytelen költési limit megadása (pl. negatív szám)
Elvárt eredmény	Sikertelen módosítás
Tényleges eredmény	Megfelel az elvárt eredménynek

3.18. táblázat. Sikertelen havi költési limit módosítása

Teszteset azonosító	TC018
Funkció	Új megtakarítás létrehozása helyes adatok megadásával
Bemenet	Név, leírás, célösszeg (opcionális), határidő megadása (opcionális)
Elvárt eredmény	Sikeres megtakarítás létrejötte a megadott adatokkal
Tényleges eredmény	Megfelel az elvárt eredménynek

3.19. táblázat. Új megtakarítás létrehozása

Teszteset azonosító	TC019
Funkció	Összeg hozzáadása megtakarításhoz
Bemenet	Megtakarításhoz adott összeg megadása (pl. 10000)
Elvárt eredmény	Az összeg hozzáadódik a megtakarításhoz, frissül az aktuális összeg
Tényleges eredmény	Megfelel az elvárt eredménynek

3.20. táblázat. Összeg hozzáadása megtakarításhoz

Teszteset azonosító	TC020
Funkció	Összeg elvétele megtakarításból
Bemenet	Megtakarításból elvont összeg megadása (pl. 5000)
Elvárt eredmény	Az összeg levonásra kerül, frissül az aktuális összeg
Tényleges eredmény	Megfelel az elvárt eredménynek

3.21. táblázat. Összeg elvétele megtakarításból

Teszteset azonosító	TC021
Funkció	Megtakarítás törlése
Bemenet	Meglévő megtakarítás kártyán lévő kuka ikon megnyomása és a törlés megerősítése
Elvárt eredmény	A kiválasztott megtakarítás törlésre kerül
Tényleges eredmény	Megfelel az elvárt eredménynek

3.22. táblázat. Megtakarítás törlése

Teszteset azonosító	TC022
Funkció	Részletes riport letöltése
Bemenet	Letöltés gombra
Elvárt eredmény	A részletes riport fájl letöltésre kerül
Tényleges eredmény	Megfelel az elvárt eredménynek

3.23. táblázat. Részletes riport letöltése

Teszteset azonosító	TC023
Funkció	Év váltása az éves összefoglaló riporton
Bemenet	Másik év kiválasztása az éves riport nézetben
Elvárt eredmény	Az adott évre vonatkozó adatok jelennek meg
Tényleges eredmény	Megfelel az elvárt eredménynek

3.24. táblázat. Év váltása éves összefoglaló riporton

Teszteset azonosító	TC024
Funkció	Új csoport létrehozása
Bemenet	Csoportnév, típus, leírás megadása, létrehozás gomb megnyomása
Elvárt eredmény	A csoport sikeresen létrejön és megjelenik a listában
Tényleges eredmény	Megfelel az elvárt eredménynek

3.25. táblázat. Új csoport létrehozása

Teszteset azonosító	TC025
Funkció	Csoport törlése
Bemenet	Meglévő csoport törlésének megerősítése
Elvárt eredmény	A kiválasztott csoport törlésre kerül a listából
Tényleges eredmény	Megfelel az elvárt eredménynek

3.26. táblázat. Csoport törlése

Teszteset azonosító	TC026
Funkció	Csoportban új tranzakció rögzítése
Bemenet	Összeg és leírás megadása
Elvárt eredmény	A tranzakció megjelenik a táblázatban a felhasználó nevével és helyes adatokkal
Tényleges eredmény	Megfelel az elvárt eredménynek

3.27. táblázat. Csoportban tranzakció rögzítése

Teszteset azonosító	TC027
Funkció	Csoport elhagyása
Bemenet	Csoportból kilépés megerősítése
Elvárt eredmény	A felhasználó kikerül a csoportból
Tényleges eredmény	Megfelel az elvárt eredménynek

3.28. táblázat. Csoport elhagyása

Teszteset azonosító	TC028
Funkció	Tag meghívása csoporthoz
Bemenet	Új tag felhasználónevének megadása, meghívó küldése
Elvárt eredmény	Meghívó elküldve
Tényleges eredmény	Megfelel az elvárt eredménynek

3.29. táblázat. Tag meghívása csoporthoz - sikeres

Teszteset azonosító	TC029
Funkció	Tag meghívása csoporthoz
Bemenet	Nem létező felhasználónév, üres mezők megadása
Elvárt eredmény	Hibajelzés, meghívó nem kerül küldésre
Tényleges eredmény	Megfelel az elvárt eredménynek

3.30. táblázat. Tag meghívása csoporthoz - sikertelen

Teszteset azonosító	TC030
Funkció	Meghívó elfogadása
Bemenet	Meghívó elfogadása az értesítések fülön
Elvárt eredmény	A felhasználó bekerül a csoportba
Tényleges eredmény	Megfelel az elvárt eredménynek

3.31. táblázat. Meghívó elfogadása

Teszteset azonosító	TC031
Funkció	Meghívó elutasítása
Bemenet	Meghívó elutasítása az értesítések fülön
Elvárt eredmény	A felhasználó nem kerül be a csoportba, meghívás megszűnik
Tényleges eredmény	Megfelel az elvárt eredménynek

3.32. táblázat. Meghívó elutasítása

Teszteset azonosító	TC032
Funkció	Csoport megtakarítás hozzáadása
Bemenet	Új megtakarítás név és cél megadása csoport szinten
Elvárt eredmény	A csoport megtakarítás létrejön és megjelenik a listában
Tényleges eredmény	Megfelel az elvárt eredménynek

3.33. táblázat. Csoport megtakarítás hozzáadása

Teszteset azonosító	TC033
Funkció	Összeg elvétele csoport megtakarításból
Bemenet	Összeg megadása, amely levonásra kerül a csoport megtakarításból
Elvárt eredmény	Az aktuális összeg csökken a megadott összeggel
Tényleges eredmény	Megfelel az elvárt eredménynek

3.34. táblázat. Összeg elvétele csoport megtakarításból

Teszteset azonosító	TC034
Funkció	Összeg berakása csoport megtakarításba
Bemenet	Összeg megadása, amely hozzáadódik a csoport megtakarításhoz
Elvárt eredmény	Az aktuális összeg nő a megadott összeggel
Tényleges eredmény	Megfelel az elvárt eredménynek

3.35. táblázat. Összeg berakása csoport megtakarításba

Teszteset azonosító	TC035
Funkció	Csoport megtakarítás törlése
Bemenet	Kártya melletti kuka ikonra kattintás
Elvárt eredmény	A kiválasztott csoport megtakarítás törlődik
Tényleges eredmény	Megfelel az elvárt eredménynek

3.36. táblázat. Csoport megtakarítás törlése

Teszteset azonosító	TC036
Funkció	Kijelentkezés
Bemenet	Baloldali menüsávban található kijelentkezés gomb megnyomása
Elvárt eredmény	Munkamenet befejezése, átirányítás a bejelentkező felületre
Tényleges eredmény	Megfelel az elvárt eredménynek

3.37. táblázat. Kijelentkezés

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 költéseink 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

Egy alkalmazást véleményem szerint gyakorlatilag soha nem lehet teljesen befejezettnek tekinteni. Mindig lehet hova fejleszteni, és mindig lesznek új ötletek hozzá, illetve ha más nem is, újabb technológiák egészen biztosan. És egy programozó is folyamatosan tanul, egyre jobb és jobb termékeket igyekszik kiadni a kezei közül.

Ezen webalkalmazás továbbfejlesztésére számos olyan ötletem van, amelyek megvalósítása már nem esett jelen projekt keretei közé. A következő fejezetben ezeket fogom röviden kifejteni.

5.1. Statisztikák

A "Reports" oldal jelenleg 2 darab egyszerű, összegző diagramot tartalmaz csak, de itt azt gondolom sokkal nagyobb potenciál is lehetne. Akár AI modellekkel előállítani statisztikákat, elemezni a költési trendeket, és ezáltal személyre szabott pénzügyi tanácsadást biztosítani a felhasználónak.

Megjegyezném, hogy AI integráció nélkül is érdemes lenne a statisztika sokszínű eszközeit jobban kihasználni, és az adattengerből hasznos következtetéseket levonni.

5.2. Tranzakciók

A tranzakciók lekövetésére és megjelenítésére is számos jobb módszer lehetne, mint amit én ebben a projektben megvalósítottam. A felhasználónak például jelentősen megkönnyítené a dolgát, ha nem manuálisan kellene bevinnie mindig a kiadásait. Erre nyújthat megoldást egy beépített AI technológia, amely egy lefotózott

blokk/számla tartalmát értelmezni tudja, és automatikusan, tételenként, rögzíti az alkalmazásba. Ezen felül még lehetne ismétlődő bevételeket és kiadásokat beállítani (például fizetés, lakbér, stb.), ezzel is csökkentvén a felhasználói beavatkozást.

5.3. Csoportok

A csoportos funkciókban szerintem rengeteg potenciál bújik meg. A közös megtakarítások lehetőséget adnak arra, hogy barátok, családtagok vagy akár munkatársak együtt kezeljék bizonyos pénzügyeiket, megtakarításaikat. Az alap koncepción nem sokat változtatnék, inkább a kivitelezésen és a plusz funkciókon. A csoportos takarékoskodás nemcsak anyagi szempontból lehet hatékonyabb, hanem pszichológiai szempontból is motiváló. Ha mások is részt vesznek benne, az ösztönözheti az embert, hogy ő is kitartóbb legyen. Ehhez kapcsolódhatnak különféle motivációs taktikák, mint például haladási grafikonok, ranglisták, vagy gamification-elemek (jelvények, célkitűzések, stb.).

Illetve a jövőben érdemes lehet olyan funkciókat is beépíteni, mint például a csoportos üzenetküldés és kommentelés, és automatikus értesítések a tagoknak a határidőkről, befizetésekről.

Ez a modul nemcsak a közösségi élményt erősítené, hanem a felhasználók aktivitását és elköteleződését is jelentősen növelhetné.

5.4. UI

A felhasználói élmény szempontjából kiemelten fontos, hogy egy alkalmazás átlátható, letisztult és esztétikus legyen. A modern webes elvárások világában a vanilla HTML, CSS és JavaScript (még Bootstrap-tel kiegészítve is) már nem feltétlenül elegendő ahhoz, hogy igazán igényes, vizuálisan is kiemelkedő felhasználói felületet hozzunk létre. Emellett például egy React vagy más frontend keretrendszer alkalmazása nemcsak látványosabb UI-t tesz lehetővé, de strukturáltabb, könnyebben karbantartható kódot is eredményez. Ezért érdemes lehet a felhasználói felületet (UI[1]) egy modern frameworkben újraalkotni.

[1]UI = User Interface (felhasználói felület)

Ábrák jegyzéke

Táblázatok jegyzéke

Forráskódjegyzék