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

1. Bevezetés **3**

2. Felhasználói dokumentáció **4**

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

3. Fejlesztői dokumentáció **8**

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 10

3.2.4. API 10

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

4. Összegzés **16**

Köszönetnyilvánítás **17**

A. Szimulációs eredmények **18**

Irodalomjegyzék **20**

Ábrajegyzék **20**

Táblázatjegyzék **21**

Algoritmusjegyzék **22**

Forráskódjegyzék **23**

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 költéseinket 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.

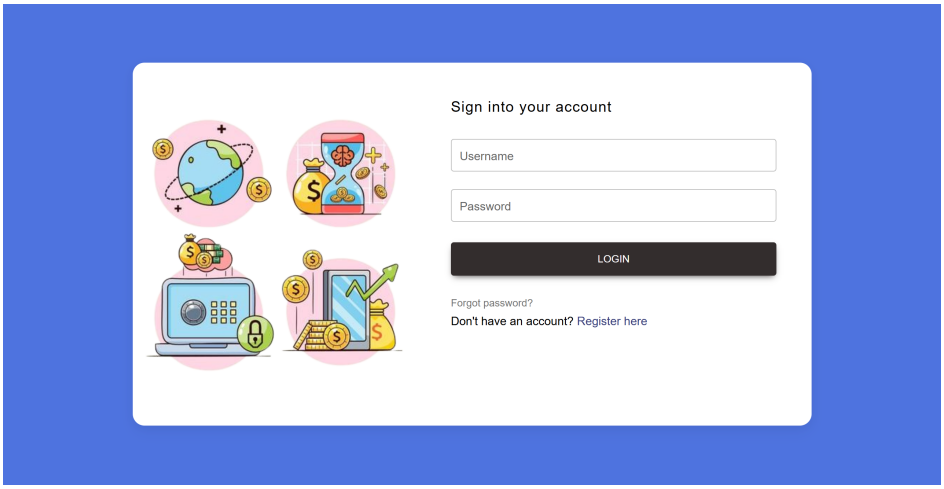
2.2. Felhasználói esetek

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

2.2.1. Fiók

Funkció	Leírás
Bejelentkezés	Bejelentkezés egy egyedi felhasználónévvel és egy jelszóval lehetséges.
Regisztráció	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).
Elfelejtett jelszó	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.
Kijelentkezés	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



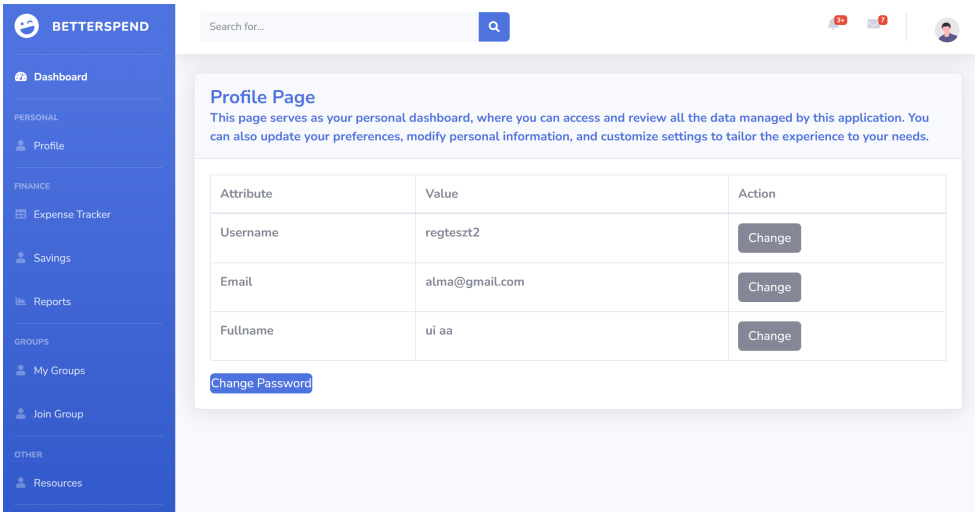
2.1. ábra. Screenshot: Bejelentkező felület

2.2.2. Profil

A profil oldalt a bal menüsávból érhetjük el.

Funkció	Leírás
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.
Jelszó módosítás	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

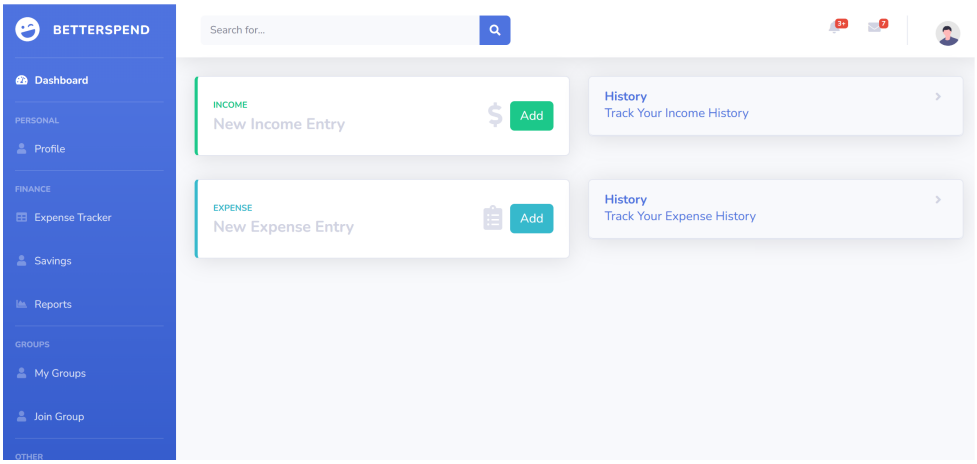
2.2.3. Pénzügyek

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
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.
Előzmények	A költési 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

-

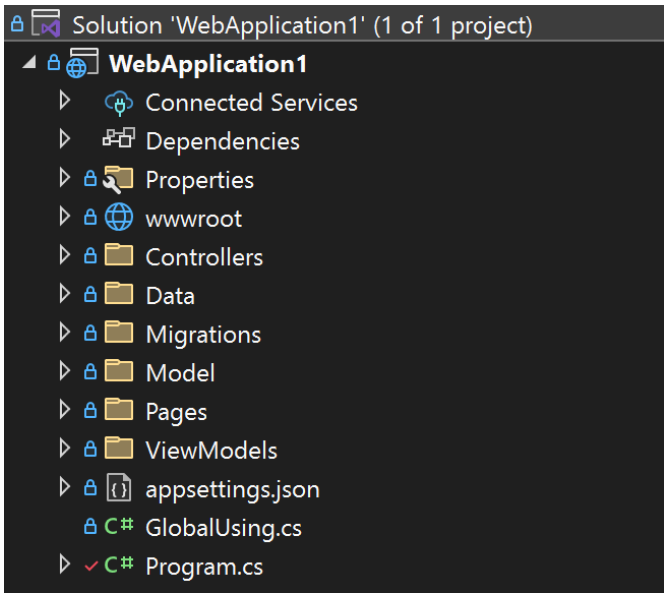
3. fejezet

Fejlesztői dokumentáció

A következő fejezetben fogom az alkalmazást fejlesztői oldalról bemutatni.

3.1. Architektúra

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



3.1. ábra. Projekt struktúra

[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.

3.1.1. 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[2] fájlstruktúrája alapján automatikusan létrejönnek a route-ok, ha a "Program.cs"-ben megadjuk a programnak, hogy hozza létre őket egy szimpla sorral:

```
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

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.

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

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 kinál 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.[3]

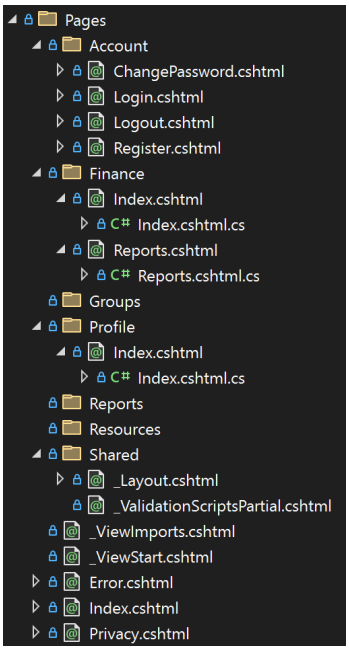
3.3. Frontend

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**??** látható tehát, hogy például a bejelentkező felületet a következő linken tudjuk elérni: "localhost:«port»AccountLogin".

[3]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:

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

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

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:

```
// Get MySQL connection string from configuration
string connectionString = builder.Configuration.
    GetConnectionString("MySqlConnection");

// Add DB Context to the application
builder.Services.AddDbContext<ApplicationDbContext>(options =>
{
  options.UseMySql(connectionString, ServerVersion.AutoDetect(
      connectionString));
});
```

3.6. forráskód. Hello World in C#

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; }

  }
}
```

3.7. forráskód. Hello World in C#

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

Lorem ipsum dolor sit amet, consectetur adipiscing elit. In eu egestas mauris. Quisque nisl elit, varius in erat eu, dictum commodo lorem. Sed commodo libero et sem laoreet consectetur. Fusce ligula arcu, vestibulum et sodales vel, venenatis at velit. Aliquam erat volutpat. Proin condimentum accumsan velit id hendrerit. Cras egestas arcu quis felis placerat, ut sodales velit malesuada. Maecenas et turpis eu turpis placerat euismod. Maecenas a urna viverra, scelerisque nibh ut, malesuada ex.

Aliquam suscipit dignissim tempor. Praesent tortor libero, feugiat et tellus porttitor, malesuada eleifend felis. Orci varius natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Nullam eleifend imperdiet lorem, sit amet imperdiet metus pellentesque vitae. Donec nec ligula urna. Aliquam bibendum tempor diam, sed lacinia eros dapibus id. Donec sed vehicula turpis. Aliquam hendrerit sed nulla vitae convallis. Etiam libero quam, pharetra ac est nec, sodales placerat augue. Praesent eu consequat purus.

Köszönetnyilvánítás

Amennyiben a szakdolgozati / diplomamunka projekted pénzügyi támogatást kapott egy projektből vagy az egyetemtől, jellemzően kötelező feltüntetni a dolgozatban is. A dolgozat elkészítéséhez segítséget nyújtó oktatók, hallgatótársak, kollégák felé is nyilvánítható külön köszönet.

A. függelék

Szimulációs eredmények

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Pellentesque facilisis in nibh auctor molestie. Donec porta tortor mauris. Cras in lacus in purus ultricies blandit. Proin dolor erat, pulvinar posuere orci ac, eleifend ultrices libero. Donec elementum et elit a ullamcorper. Nunc tincidunt, lorem et consectetur tincidunt, ante sapien scelerisque neque, eu bibendum felis augue non est. Maecenas nibh arcu, ultrices et libero id, egestas tempus mauris. Etiam iaculis dui nec augue venenatis, fermentum posuere justo congue. Nullam sit amet porttitor sem, at porttitor augue. Proin bibendum justo at ornare efficitur. Donec tempor turpis ligula, vitae viverra felis finibus eu. Curabitur sed libero ac urna condimentum gravida. Donec tincidunt neque sit amet neque luctus auctor vel eget tortor. Integer dignissim, urna ut lobortis volutpat, justo nunc convallis diam, sit amet vulputate erat eros eu velit. Mauris porttitor dictum ante, commodo facilisis ex suscipit sed.

Sed egestas dapibus nisl, vitae fringilla justo. Donec eget condimentum lectus, molestie mattis nunc. Nulla ac faucibus dui. Nullam a congue erat. Ut accumsan sed sapien quis porttitor. Ut pellentesque, est ac posuere pulvinar, tortor mauris fermentum nulla, sit amet fringilla sapien sapien quis velit. Integer accumsan placerat lorem, eu aliquam urna consectetur eget. In ligula orci, dignissim sed consequat ac, porta at metus. Phasellus ipsum tellus, molestie ut lacus tempus, rutrum convallis elit. Suspendisse arcu orci, luctus vitae ultricies quis, bibendum sed elit. Vivamus at sem maximus leo placerat gravida semper vel mi. Etiam hendrerit sed massa ut lacinia. Morbi varius libero odio, sit amet auctor nunc interdum sit amet.

Aenean non mauris accumsan, rutrum nisi non, porttitor enim. Maecenas vel tortor ex. Proin vulputate tellus luctus egestas fermentum. In nec lobortis risus,

sit amet tincidunt purus. Nam id turpis venenatis, vehicula nisl sed, ultricies nibh. Suspendisse in libero nec nisi tempor vestibulum. Integer eu dui congue enim venenatis lobortis. Donec sed elementum nunc. Nulla facilisi. Maecenas cursus id lorem et finibus. Sed fermentum molestie erat, nec tempor lorem facilisis cursus. In vel nulla id orci fringilla facilisis. Cras non bibendum odio, ac vestibulum ex. Donec turpis urna, tincidunt ut mi eu, finibus facilisis lorem. Praesent posuere nisl nec dui accumsan, sed interdum odio malesuada.

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 8
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

Algoritmusjegyzék

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. Hello World in C# 14
3.7. Hello World in C# 14