



INFORMATIKA ÉS TÁVKÖZLÉS ÁGAZAT



A szakdolgozat címe

Készítette

Kiss Ákos

Maklári Gábor

Minka Dominik Elemér

Szoftverfejlesztő és -tesztelő szak

Témavezető

Kerényi Róbert Nándor

oktató

MISKOLC, 2024

Tartalomjegyzék

Bevezetés	5
1. Technológiák	6
1.1. Relációs adatbáziskezelő	6
1.1.1. Felépítés	6
1.1.2. Főbb jellemzők	6
1.2. PHP-MyAdmin - MySQL - MariaDB	7
1.2.1. PHPMyAdmin	7
1.2.2. MySQL	7
1.2.3. MariaDB	8
1.3. React Native	9
1.3.1. A React	9
1.3.2. React DOM és Virtual DOM	10
1.3.3. Renderelési folyamat	10
1.3.4. React renderelés menete	11
2. Projekt bemutatása	12
2.1. Adatbázis bemutatása	13
2.1.1. Tranzakciók	13
2.1.2. Hitelkártyák	14
2.1.3. Számlák	14
2.1.4. Ügyfelek	14
2.1.5. Felhasználók	15
2.1.6. Kapcsolatok az Adatbázisban	15
2.2. Backend bemutatása	16
2.2.1. Login és Logout	16
2.2.2. Regisztráció	16
2.2.3. Számlák	17
2.2.4. Tranzakciók	19
2.2.5. Ügyfelek	20
2.3. Frontend bemutatása	22
2.3.1. Weboldal áttekintése	22

2.3.2. Megjelenés és design	23
2.3.3. Funkciók	23
2.3.4. Banki személyes ügyek modális ablak	24
2.3.5. Biztonság és Ügyfélszolgálat	24
2.3.6. Funkciók és Jellemzők	25
2.3.7. Kapcsolat és Céginformációk	25
2.4. WPF bemutatása	26
2.5. C# .Net Web API	27
2.5.1. A Postman:	27
2.5.2. Unit tesztelés	27
2.5.3. Hibakeresés (Debugging)	28
2.6. Swagger API Dokumentáció	28
2.6.1. BackupRestore	28
2.6.2. FileUpload	28
2.6.3. Jelző	28
2.7. Entity Framework	29
2.7.1. Az ADO.NET Entitás-keretrendszer	29
2.7.2. CRUD:	30
Összegzés	31
Irodalomjegyzék	33

Bevezetés

MAMIK Bank

Miért választottuk ezt a témát?

Közös megegyezés alapján, arra jutottunk, hogy a mai banki alkalmazások nem felelnek meg egy átlag ember elvárásainak. Ezért úgy döntöttünk, hogy az összes banki applikációt kivesézzük az előnyeiről-hátrányairol, és az alkalmazások által tanultakat megpróbáljuk szinkronba hozni hogy megalkossunk egy innovatív alkalmazást. Meg szeretnénk ismerkedni a bankok minden napi tranzakcióival, működésével, felépítésével és nehézségeivel. Mire nyújt megoldást?

Szerettünk volna egy olyan banki tranzakciós applikációt létrehozni, aminek kikezdhetetlen a biztonsági rendszere, biometrikus beléptetést nyújt az extra biztonság érdekében, felhasználóbarát (minden korosztálynak könnyű kezelést nyújt), modern kezelő-felülettel rendelkezik a futurisztikus érzés átadásáért, egyszerűsíti a pénzgazdálkodást, reszponzív és bárhonnan (nemzetközileg is) hozzáférhető. Mitől életszerű?

Ez a banki alkalmazás életszerű, mert a felhasználók minden napি pénzügyi szükségleteit szolgálja ki, legyen szó tranzakcióról, fizetésekről, megtakarításról vagy biztonságos pénzkezelésről. Az emberek gyors, kényelmes és biztonságos megoldásokat keresnek a pénzügyeik kezelésére, és egy banki alkalmazás pontosan ezt nyújtja. [?]

1. fejezet

Technológiák

1.1. Relációs adatbáziskezelő

1.1.1. Felépítés

A relációs adatbázisok olyan rendszerek, amelyek adatokat tárolnak és kezelnak, és ezeket táblák formájában szervezik. Gondolj rájuk úgy, mint egy jól rendezett könyvtárra, ahol minden könyv (adat) egy adott kategóriában (táblában) van elhelyezve.

1.1.2. Főbb jellemzők

1. Táblák: Az adatokat táblákba szervezzük, ahol minden tábla sorokból (rekordok) és oszlopokból (mezők) áll. Például, ha van egy „Könyvek” tábla, az oszlopok lehetnek: „Cím”, „Szerző”, „Kiadási év”.
2. Kapcsolatok: A relációs adatbázisok lehetővé teszik a táblák közötti kapcsolatokat. Például egy „Könyvek” tábla és egy „Szerzők” tábla között létrejöhet egy kapcsolat, amely megmutatja, hogy melyik könyv ki által lett írva.
3. Kérdezés: Az adatok lekérdezésére általában SQL-t (Structured Query Language) használnak. Ez egy speciális nyelv, amivel adatokat tudsz lekérdezni, hozzáadni, módosítani vagy törölni.
4. Adatbiztonság: A relációs adatbázisok sokféle biztonsági megoldást kínálnak, például jogosultságokat, amelyek meghatározzák, ki férhet hozzá az adatokhoz.

1. Példák: –

Webáruházak: A termékek, vásárlók és rendelési adatok nyilvántartására.

– Iskolák: A diákok, tanárok és tantárgyak adatainak kezelésére.

1.2. PHP-MyAdmin - MySQL - MariaDB

1.2.1. PHPMyAdmin

1.1. *Megjegyzés.* A PHPMyAdmin egy webalapú alkalmazás, amely lehetővé teszi a MySQL és MariaDB adatbázisok kezelését. PHP-ban íródott, és a felhasználók grafikus felületen keresztül érhetik el az adatbázisokat.

1. Előnyök.

Felhasználóbarát: Könnyen használható grafikus felületet biztosít, amely segíti a felhasználókat az adatbázisok kezelésében.

- Funkciók: Támogatja a legtöbb MySQL/MariaDB funkciót, beleértve az adatbázisok létrehozását, módosítását, táblák kezelését, SQL lekérdezések futtatását stb.
- Platformfüggetlen: Mivel webalapú, bármilyen eszközről elérhető, amely rendelkezik böngészővel.

1. Hátrányok.

Teljesítmény: Nagy adatbázisok esetén lassabb lehet, mint a parancssori megoldások.

- Biztonsági kockázatok: Mivel webes alkalmazás, ha nem megfelelően van konfigurálva, biztonsági kockázatokat jelenthet (pl. SQL injection).
- Függőség: PHP és a webkiszolgáló (pl. Apache, Nginx) konfigurálása szükséges.

1.2.2. MySQL

1.2. *Megjegyzés.* A MySQL egy népszerű, nyílt forráskódú relációs adatbázis-kezelő rendszer. A webfejlesztésben és más alkalmazásokban széles körben használják.

2. Előnyök.

Stabilitás: Széles körben használt és megbízható, sok nagyvállalat is használja.

- Teljesítmény: Optimális teljesítményt nyújt, különösen nagy adatmennyiségek esetén.
- Széles körű támogatás: Nagy közösségi támogatás és dokumentáció, sokféle eszköz és könyvtár támogatja.

2. Hátrányok.

Licencdíjak: A MySQL Enterprise verziója díjköteles, míg a közösségi verzió korlátosabb funkciókkal rendelkezik.

- Zárt forráskód: A MySQL egyes részei zárt forráskódúak, ami korlátozhatja a testreszabhatóságot.

1.2.3. MariaDB

1.3. *Megjegyzés.* A MariaDB a MySQL forkja, amelyet az eredeti MySQL fejlesztői hoztak létre, miután a MySQL-t megvásárolta az Oracle. Célja, hogy a MySQL-hez hasonló, de nyílt forráskódú alternatívát kínáljon.

3. Előnyök.

Teljesítmény: Számos optimalizálás van benne, ami javítja a teljesítményt, például a párhuzamos feldolgozás.

- Nyílt forráskód: Teljes mértékben nyílt forráskódú, így a felhasználók szabadon módosíthatják és terjeszthetik.
- Kompatibilitás: Nagy mértékben kompatibilis a MySQL-lel, így könnyen át lehet téni róla.

3. Hátrányok.

Fiatalabb közösség: Bár a közössége növekszik, még nem olyan nagy és elterjedt, mint a MySQL-é.

- Egyes funkciók hiánya: Néhány MySQL funkció nem érhető el a MariaDB-ban, bár ez folyamatosan változik.



1.3. React Native



1.3.1. A React

1. Története.

-A React.js-t eredetileg a Facebook fejlesztette, ebből adódóan a Facebook is azt használja, valamint az Instagram. 2013-ban jelent meg.

1. Mi az a React?.

-A React egy deklaratív, effektív, és rugalmas JavaScript könyvtár, felhasználói felületek készítéséhez. Lehetővé teszi komplex felhasználói felületek összeállítását izolált kódrészletekből, amiket "komponenseknek" hívunk.

1. Mit is jelent ez?.

Egy JavaScript könyvtár, amelyet felhasználói felületek, különösen egyoldalas alkalmazások "SPA" készítésére használnak.

komponensalapú: az alkalmazások kisebb, újrafelhasználható komponensekből állnak. hatékonyan kezeli a DOM-ot a "virtual DOM" segítségével.

Deklaratív megközelítés, ahol a komponensek állapotváltozásai automatikusan frissítik a felhasználói felületet.

Virtual DOM használata gyorsabb frissítéseket eredményez a felhasználói felületen.

Nagyobb, összetett alkalmazások, amelyek sok felhasználói interakciót igényelnek.

1. Miért hasznos ez nekünk?.

-Felmerülhet bennünk a kérdés, hogy milyen esetekben érdemes használni ezeket a keretrendszerket, könyvtárakat ?

A mai modern frontend keretrendserek előnyei többek között, hogy

- (pont)Könnyen használhatók
- (pont)Komponensekből állnak, melyeket újra hasznosíthatunk
- (pont)Strukturált, áttekinthető kódot biztosítanak számunkra
- (pont)Lehetőség nyílhat velük nem csupán "single-page" applikációk írására, így az "üzleti logika" jobban elkülöníthető
- (pont)Különböző állapotok jobban leírhatók vele
- (pont)Időt spórolnak meg számunkra

1.3.2. React DOM és Virtual DOM

- React fő jellemzője a Virtual DOM használata, amely egy memóriában található DOM-reprezentáció.
- Amikor a React-ben valami változik, először ezt a Virtual DOM-ot frissíti.
- A Virtual DOM-ot aztán összehasonlítja a tényleges DOM-mal (diff algoritmus), és csak azokat a részeket frissíti a valódi DOM-ban, amelyek ténylegesen megváltoztak.

1.3.3. Renderelési folyamat

- A komponens először a kezdeti állapottal (Helló, világ!) renderel.
- A felhasználó megnyomja a gombot, ami kiváltja a setMessage funkciót, amely frissíti az állapotot.
- A React látja, hogy az állapot változott, ezért újra rendereli a komponenst az új értékkel "Üzenet frissítve!".
- A DOM csak az új üzenetet frissíti, minden más változatlan marad.
- Az előbbi példa bemutatja a React alapvető renderelési logikáját, és azt, hogyan frissíti hatékonyan a felhasználói felületet az állapotváltozások alapján.

1.3.4. React renderelés menete

- A React renderelési folyamata az, ami a komponensekből felépített virtuális DOM-ot összehasonlítja a valós DOM-mal, és hatékonyan frissíti azt, csak ott, ahol változások történtek. A React komponensek alapértelmezés szerint minden újra renderelnek, amikor az állapotuk "state" vagy a kapott props-ok változnak.
- Főbb pontok:

Renderelés: A komponensek JSX szyntaxssal írnak HTML-szerű struktúrát, amit a React render() vagy funkcionális komponensek esetén maga a komponens visszatérési értéke kezel.

- Virtual DOM: A React a változtatásokat először egy memória alapú reprezentáció (virtual DOM) alkalmazza, majd csak azokat a részeket frissíti a tényleges DOM-ban, amelyek ténylegesen változtak.
- Újrarenderelés: A komponensek újra renderelődnek, ha az állapot (state) vagy a props változik, de a React hatékony diff algoritmussal kezeli a DOM frissítését.

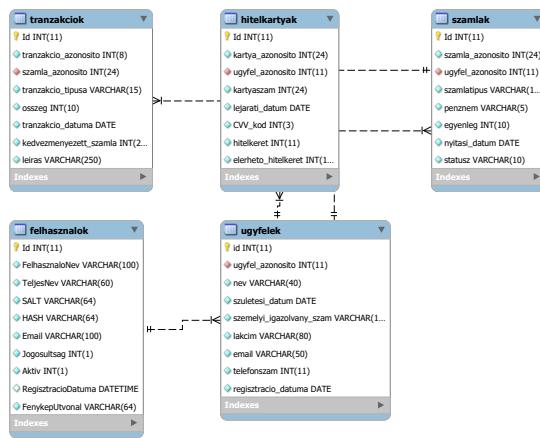
2. fejezet

Projekt bemutatása

A MAMIK Bank egy modern online banki platform, amely felhasználóbarát szolgáltatásokat kínál ügyfelei számára. Az oldal célja a pénzügyi biztonság és a könnyű pénzkezelés biztosítása.

2.1. Adatbázis bemutatása

Ez az adatbázis egy banki rendszert modellez, amely különböző entitásokat kezel, mint például ügyfelek, számlák, tranzakciók, hitelkártyák és felhasználók.



2.1.1. Tranzakciók

- **Id**: Egyedi azonosító
- **tranzakcio_azonosito**: Tranzakció azonosító
- **szamla_azonosito**: Kapcsolat a számlákkal
- **tranzakcio_tipusa**: A tranzakció típusa (pl. befizetés, kifizetés)
- **osszeg**: Tranzakció összege
- **tranzakcio_datum**: Tranzakció időpontja
- **kedvezmenyezett_szamla**: Kedvezményezett számla azonosítója
- **leiras**: Tranzakció részletei

2.1.2. Hitelkártyák

- **Id**: Egyedi azonosító
- **karta_azonosito**: Kártya azonosító
- **ugyfel_azonosito**: Kapcsolat az ügyfelekkel
- **kartyaszam**: Hitelkártya száma
- **lejarati_datum**: Lejárat dátum
- **CVV_kod**: Biztonsági kód
- **hitelkeret**: Hitelkeret összege
- **elerheto_hitelkeret**: Elérhető hitelkeret

2.1.3. Számlák

- **Id**: Egyedi azonosító
- **szamla_azonosito**: Számla azonosító
- **ugyfel_azonosito**: Kapcsolat az ügyfelekkel
- **szamlatipus**: Számlatípus (pl. folyószámla, megtakarítási számla)
- **penznem**: Pénznem (pl. HUF, EUR)
- **egyenleg**: Számla egyenlege
- **nyitasi_datum**: Nyitási dátum
- **statusz**: Számla állapota (pl. aktív, lezárt)

2.1.4. Ügyfelek

- **Id**: Egyedi azonosító
- **ugyfel_azonosito**: Ügyfél azonosító
- **nev**: Név
- **szuletesi_datum**: Születési dátum
- **szemelyi_igazolvany_szam**: Személyi igazolvány szám
- **lakcim**: Lakcím
- **email**: Email cím
- **telefonszam**: Telefonszám
- **regisztracio_datum**: Regisztráció dátuma

2.1.5. Felhasználók

- **Id:** Egyedi azonosító
- **FelhasznaloNev:** Felhasználónév
- **TeljesNev:** Teljes név
- **SALT:** Titkosítás
- **HASH:** Jelszó hash
- **Email:** Email cím
- **Jogosultsag:** Jogosultsági szint
- **Aktiv:** Aktivitási állapot
- **RegisztracioDatuma:** Regisztráció dátuma
- **FenykepUtvonal:** Profilkép útvonala

2.1.6. Kapcsolatok az Adatbázisban

- Az **ügyfelek** kapcsolódnak a **számlákhoz** és **hitelkártyákhoz**.
- A **számlák** kapcsolódnak a **tranzakciókhoz**.
- A **felhasználók** a rendszer adminisztrátori oldalához kapcsolódnak.

2.2. Backend bemutatása

- A MAMIK Bank backend egy ASP.NET Core alapú, MySQL adatbázist használó rendszer, amely biztonságosan kezeli az ügyféladatokat és tranzakciókat. A RESTful API biztosítja a gyors kommunikációt a frontend és backend között, míg a JWT token alapú hitelesítés és a szerepkör-alapú jogosultságkezelés védi az adatokat. A rendszer HTTPS és TLS titkosítással, bcrypt jelszóhasheléssel és SQL injection elleni védelemmel garantálja a biztonságot. Az indexelt adatbázis, caching és terheléselosztás növeli a teljesítményt és skálázhatóságot. Külső API-integrációkat támogat, lehetőséget adva további pénzügyi szolgáltatások beépítésére.

2.2.1. Login és Logout

- Bejelentkezés:** Az ügyfél fiókjába való belépés.
- Kijelentkezés:** Az ügyfél biztonságos kilépése a rendszerből.

```
public async Task<ActionResult> Login(LoginDTO loginDTO)
{
    using (var cx = new MamikBankContext())
    {
        try
        {
            string Hash = Program.CreateSHA256(loginDTO.Temptoken);
            Felhasznalok loggedUser = await cx.Felhasznalok.FirstOrDefaultAsync(f => f.Felhasznalnev == loginDTO.LoginName && f.Hash == Hash);
            if (loggedUser != null && loggedUser.Aktiv == 1)
            {
                string token = Guid.NewGuid().ToString();
                lock (Program.LoggedInUsers)
                {
                    Program.LoggedInUsers.Add(token, loggedUser);
                }
                return Ok(new LoggedInUser { Name = loggedUser.TeljesNev, Email = loggedUser.Email,
                                            Permission = loggedUser.Jogosultsag, ProfilePicturePath = loggedUser.FenykepUtval, Token = token, UgyfelAzonosito=loggedUser.Id });
            }
            else
            {
                return BadRequest("Nincs név vagy jelszó/inaktiv felhasználó!");
            }
        }
        catch (Exception ex)
        {
            return BadRequest(new LoggedUser { Permission = -1, Name = ex.Message, ProfilePicturePath = "", Email = "" });
        }
    }
}
```

2.2.2. Regisztráció

- Új ügyfél regisztrációja:** Új felhasználók hozzáadása a rendszerhez.

```
public async Task<ActionResult> Regisztracio(Felhasznalok felhasznalok)
{
    using (var cx = new MamikBankContext())
    {
        try
        {
            if (cx.Felhasznalok.FirstOrDefault(f => f.Felhasznalnev == felhasznalok.Felhasznalnev) != null)
            {
                return Ok("Már létezik ilyen felhasználónév!");
            }
            if (cx.Felhasznalok.FirstOrDefault(f => f.Email == felhasznalok.Email) != null)
            {
                return Ok("Ezzel az E-mail címmel már regisztráltak!");
            }
            felhasznalok.Jogosultsag = 1;
            felhasznalok.Aktiv = 0;
            felhasznalok.Hash = Program.CreateSHA256(felhasznalok.Hash);
            await cx.Felhasznalok.AddAsync(felhasznalok);
            await cx.SaveChangesAsync();

            Program.SendEmail(felhasznalok.Email, "Regisztráció", $"http://"
                + $":5000/api/regisztracio?felhasznalnev={felhasznalok.Felhasznalnev}&email={felhasznalok.Email}");
            return Ok("Sikeres regisztráció. Fejezze be a regisztrációt az e-mail címére küldött link segítségével!");
        }
        catch (Exception ex)
        {
            return BadRequest(ex.Message);
        }
    }
}
```

2.2.3. Számlák

- **Számla lekérdezése:** Meglévő bankszámla adatainak lekérése.

```
[HttpGet("{token},{ugyfelAzonosito}")]
0 references
public async Task<IActionResult> Get(string token, int ugyfelAzonosito)
{
    using (var cx = new MamiBankContext())
    {
        try
        {
            if (Program.LoggedInUsers.ContainsKey(token) && Program.LoggedInUsers[token].Jogosultsag == 9)
            {
                return Ok(await cx.SzamlaFirstOrDefaultAsync(f => f.UgyfelAzonosito == ugyfelAzonosito));
            }
            else
            {
                return BadRequest("Nincs jogod hozzá!");
            }
        }
        catch (Exception ex)
        {
            //return BadRequest(ex.Message);
            return BadRequest(ex.InnerException?.Message);
        }
    }
}
```

- Számla létrehozása: Új bankszámla megnyitása.

```
public async Task<IActionResult> Post(string token, Szamlak szamla)
{
    using (var cx = new MamikBankContext())
    try
    {
        if (Program.LoggedInUsers.ContainsKey
            (token) && Program.LoggedInUsers
            [token].Jogosultsag == 9)
        {
            await cx.Szamlaks.AddAsync(szamla);
            await cx.SaveChangesAsync();
            return Ok("Új számla felvétele.");
        }

        else
        {
            return BadRequest("Nincs jogosultságod hozzá!");
        }
    }
    catch (Exception ex)
    {
        //return BadRequest(ex.Message);
        return BadRequest
        (ex.InnerException?.Message);
    }
}
```

- Számla módosítása: Meglévő számlaadatok frissítése.

```
public IActionResult Put(string token, Szamlak szamlak)
{
    using (var cx = new MamikBankContext())
    try
    {
        if (Program.LoggedInUsers.ContainsKey
            (token) && Program.LoggedInUsers
            [token].Jogosultsag == 9)
        {
            cx.Szamlaks.Update(szamlak);
            cx.SaveChanges();
            return Ok("A számla adatai módosítva..");
        }

        else
        {
            return BadRequest("Nincs jogosultságod hozzá!");
        }
    }
    catch (Exception ex)
    {
        //return BadRequest(ex.Message);
        return BadRequest
        (ex.InnerException?.Message);
    }
}
```

- Számla törlése: Meglévő számla törlése az adatbázisból.

```
[HttpDelete("{token},{id}")]
0 references
public IActionResult Delete(string token, int id)
{
    using (var cx = new MamikBankContext())
    try
    {
        if (Program.LoggedInUsers.ContainsKey
            (token) && Program.LoggedInUsers
            [token].Jogosultsag == 9)
        {
            cx.Szamlaks.Remove(new Szamlak { Id =id });
            cx.SaveChanges();
            return Ok("SZÁMLA adatai törölve.");
        }

        else
        {
            return BadRequest("Nincs jogosultságod hozzá!");
        }
    }
    catch (Exception ex)
    {
        //return BadRequest(ex.Message);
        return BadRequest
        (ex.InnerException?.Message);
    }
}
```

2.2.4. Tranzakciók

- **Tranzakciók lekérdezése:** Egy adott számlához kapcsolódó tranzakciók megtekintése.

```
[HttpGet("megtekartitas/{token},{szamlaAzonosito},{tranzakcioTipusa}")]
0 references
public async Task<IActionResult> Get(string token, int szamlaAzonosito, string tranzakcioTipusa)
{
    using (var cx = new MamikBankContext())
    {
        try
        {
            if (Program.LoggedInUsers.ContainsKey(token) && Program.LoggedInUsers[token].Jogosultsag == 9)
            {
                return Ok(await cx.Tranzakcioks.Where(f => f.SzamlaAzonosito == szamlaAzonosito
                && f.TranzakcioTipusa==tranzakcioTipusa).SumAsync(f=>f.Osszeg));
            }
            else
            {
                return BadRequest("Nincs jogod hozzá!");
            }
        }
        catch (Exception ex)
        {
            //return BadRequest(ex.Message);
            return BadRequest(ex.InnerException?.Message);
        }
    }
}
```

- **Új tranzakció létrehozása:** Új pénzügyi tranzakció indítása.

```
[HttpPost("{token}")]
0 references
public async Task<IActionResult> Post(string token, Tranzakciok tranzakcio)
{
    using (var cx = new MamikBankContext())
    {
        try
        {
            if (Program.LoggedInUsers.ContainsKey
            (token) && Program.LoggedInUsers
            [token].Jogosultsag == 9)
            {
                await cx.Tranzakcioks.AddAsync(tranzakcio);
                await cx.SaveChangesAsync();
                return Ok("Új tranzakcio felvéve.");
            }

            else
            {
                return BadRequest("Nincs jogosultságod hozzá!");
            }
        }
        catch (Exception ex)
        {
            //return BadRequest(ex.Message);
            return BadRequest
            (ex.InnerException?.Message);
        }
    }
}
```

- **Tranzakció módosítása:** Egy meglévő tranzakció frissítése.

```
[HttpPut("{token}")]
0 references
public IActionResult Put(string token, Tranzakciok tranzakcio)
{
    using (var cx = new MamikBankContext())
    {
        try
        {
            if (Program.LoggedInUsers.ContainsKey
            (token) && Program.LoggedInUsers
            [token].Jogosultsag == 9)
            {
                cx.Tranzakcioks.Update(tranzakcio);
                cx.SaveChanges();
                return Ok("A tranzakcio adatai módosítva..");
            }

            else
            {
                return BadRequest("Nincs jogosultságod hozzá!");
            }
        }
        catch (Exception ex)
        {
            //return BadRequest(ex.Message);
            return BadRequest
            (ex.InnerException?.Message);
        }
    }
}
```

- **Tranzakció törlése:** Egy adott tranzakció törlése a rendszerből.

```
[HttpDelete("{token},{id}")]
0 references
public IActionResult Delete(string token, int id)
{
    using (var cx = new MamikBankContext())
    try
    {
        if (Program.LoggedInUsers.ContainsKey(token) && Program.LoggedInUsers[token].Jogosultsag == 9)
        {
            cx.Tranzakciocks.Remove(new Tranzakcio { Id = id });
            cx.SaveChanges();
            return Ok("Tranzakcio adatai törölve.");
        }

        else
        {
            return BadRequest("Nincs jogosultságod hozzá!");
        }
    }
    catch (Exception ex)
    {
        //return BadRequest(ex.Message);
        return BadRequest(ex.InnerException?.Message);
    }
}
```

2.2.5. Ügyfelek

- **Ügyféladatok lekérdezése:** Egy adott ügyfél adatainak megtekintése.

```
[HttpGet("{token},{ugyfelAzon}")]
0 references
public async Task<IActionResult> Get(string token, int ugyfelAzon)
{
    using (var cx = new MamikBankContext())
    {
        try
        {
            if (Program.LoggedInUsers.ContainsKey(token) && Program.LoggedInUsers[token].Jogosultsag == 9)
            {
                return Ok(await cx.Ugyfeleks.FirstOrDefaultAsync(f => f.UgyfelAzonito == ugyfelAzon));
            }
            else
            {
                return BadRequest("Nincs jogod hozzá!");
            }
        }
        catch (Exception ex)
        {
            //return BadRequest(ex.Message);
            return BadRequest(ex.InnerException?.Message);
        }
    }
}
```

- **Új ügyfél létrehozása:** Új ügyfél hozzáadása a rendszerhez.

```
[HttpPost("{token}")]
0 references
public async Task<IActionResult> Post(string token, Ugyfelek ugyfel)
{
    using (var cx = new MamikBankContext())
    try
    {
        if (Program.LoggedInUsers.ContainsKey(token) && Program.LoggedInUsers[token].Jogosultsag == 9)
        {
            await cx.Ugyfeleks.AddAsync(ugyfel);
            await cx.SaveChangesAsync();
            return Ok("Új felhasználó felvétele.");
        }

        else
        {
            return BadRequest("Nincs jogosultságod hozzá!");
        }
    }
    catch (Exception ex)
    {
        //return BadRequest(ex.Message);
        return BadRequest(ex.InnerException?.Message);
    }
}
```

- **Ügyféladatok frissítése:** Meglévő ügyféladatok módosítása.

```
[HttpPut("{token}")]
0 references
public IActionResult Put(string token, Ugyfelek ugyfel)
{
    using (var cx = new MamikBankContext())
    try
    {
        if (Program.LoggedInUsers.ContainsKey
            (token) && Program.LoggedInUsers
            [token].Jogosultsag == 9)
        {
            cx.Ugyfeleks.Update(ugyfel);
            cx.SaveChanges();
            return Ok("A felhasználó adatai módosítva..");
        }

        else
        {
            return BadRequest("Nincs jogosultságod hozzá!");
        }
    }
    catch (Exception ex)
    {
        //return BadRequest(ex.Message);
        return BadRequest
        (ex.InnerException?.Message);
    }
}
```

- **Ügyfél törlése:** Egy adott ügyfél eltávolítása az adatbázisból.

```
[HttpDelete("{token},{id}")]
0 references
public IActionResult Delete(string token, int id)
{
    using (var cx = new MamikBankContext())
    try
    {
        if (Program.LoggedInUsers.ContainsKey
            (token) && Program.LoggedInUsers
            [token].Jogosultsag == 9)
        {
            cx.Ugyfeleks.Remove(new Ugyfelek { Id = id });
            cx.SaveChanges();
            return Ok("Új felhasználó adatai törölve.");
        }

        else
        {
            return BadRequest("Nincs jogosultságod hozzá!");
        }
    }
    catch (Exception ex)
    {
        //return BadRequest(ex.Message);
        return BadRequest
        (ex.InnerException?.Message);
    }
}
```

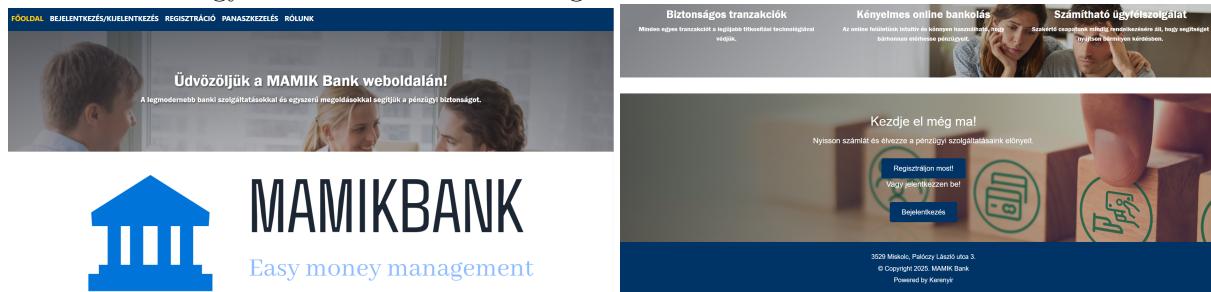
2.3. Frontend bemutatása

A MAMIK Bank frontend React.js és React Native alapokra épül, biztosítva a gyors, reszponzív és platformfüggetlen fejlesztést. A Virtual DOM technológia hatékony renderelést biztosít, míg TypeScript növeli a kódbiztonságot. Az API-kommunikációt RESTful API és JWT alapú hitelesítés támogatja, garantálva a biztonságos adatkezelést.

2.3.1. Weboldal áttekintése

A weboldal főbb elemei:

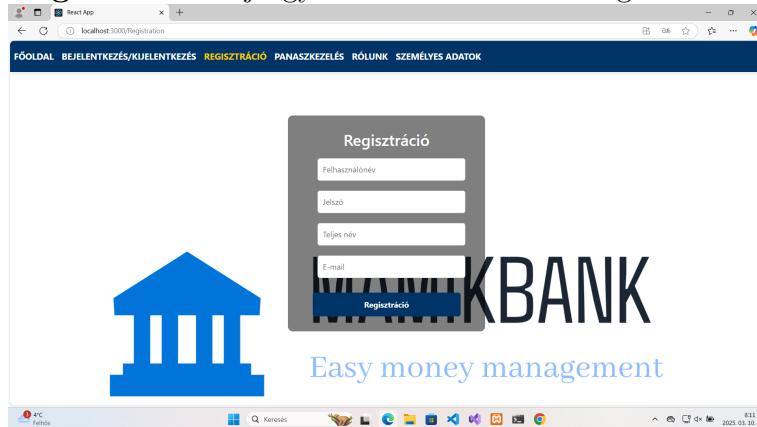
- **Főoldal:** Az ügyfelek üdvözlése és a szolgáltatások rövid bemutatása.



- **Bejelentkezés/Kijelentkezés:** Az ügyfelek bejelentkezhetnek vagy kijelentkezhetnek fiókjukból.



- **Regisztráció:** Új ügyfelek számára lehetőség fiók létrehozására.



- **Panaszkezelés:** Ügyfelek panaszaikat és észrevételeiket kezelhetik.

- **Rólunk:** Információ a bank történetéről és céljairól.

2.3.2. Megjelenés és design

A weboldal modern, letisztult megjelenéssel rendelkezik:

- A fejlécként szolgáló kék sávban találhatók a navigációs menüpontok.
- A főoldalon egy nagy üdvözlő szöveg és egy háttérkép látható.
- A bank logója egy kék színű ikon, amely egy oszlopos épületet ábrázol.
- A szlogen: „*Easy money management*” (Egyszerű pénzkezelés).

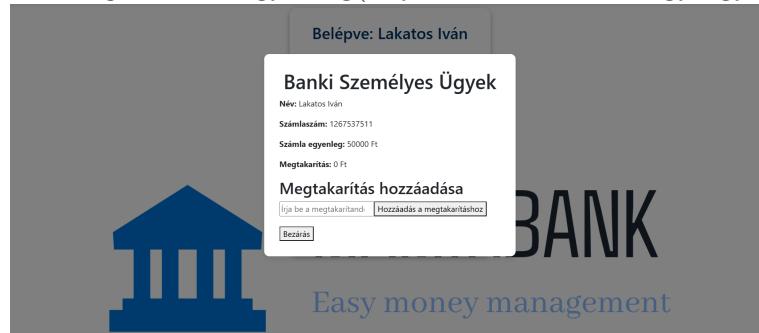
2.3.3. Funkciók

A weboldal fő funkciói a következők:

- Biztonságos bejelentkezés és kijelentkezés.
- Új ügyfelek számára regisztrációs lehetőség.
- Panaszkezelési rendszer az ügyfelek visszajelzéseinek feldolgozására.
- Banki szolgáltatások bemutatása és hozzáférhetővé tétele.

2.3.4. Banki személyes ügyek modális ablak

- Egyenlegünk tárolása megtakarításként is akár.
- Jelenlegi számla egyenleg(folyamatosan frissül egy-egy tranzakcióknál)

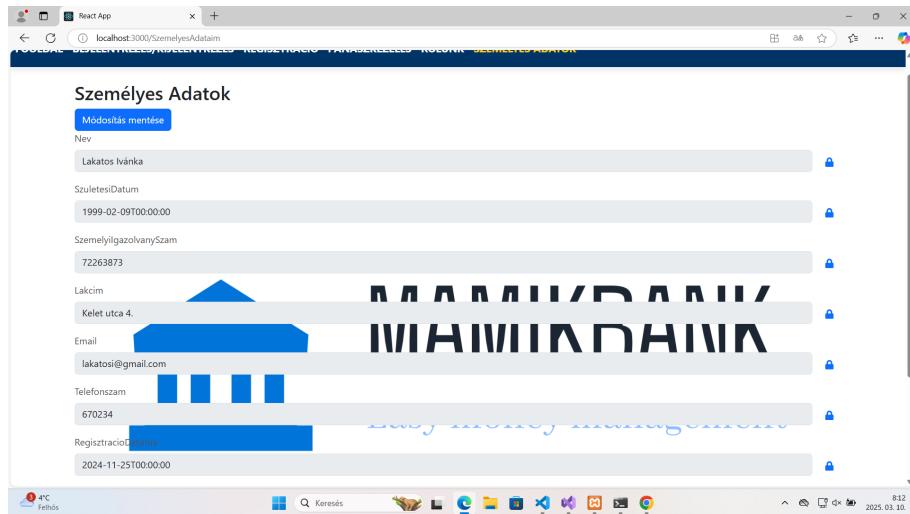


2.3.5. Biztonság és Ügyfélszolgálat

A MAMIK Bank kiemelt figyelmet fordít a biztonságos bankolásra és az ügyfélszolgálati támogatásra:

- **Biztonságos tranzakciók:** A legújabb titkosítási technológiákat alkalmazzuk minden egyes tranzakció védelmére.
- **Kényelmes online bankolás:** Intuitív és könnyen használható felületet biztosítunk, amely bárhonnan elérhető.
- **Számítható ügyfélszolgálat:** Szakértő csapatunk minden rendelkezésére áll, hogy segítséget nyújtsan bármilyen kérdésben.

2.3.6. Funkciók és Jellemzők



- Személyes adatok megtekintése:** Az ügyfelek elérhetik és ellenőrizhetik alapvető személyes adataikat, mint például nevüket, születési dátumukat, lakkímüket és elérhetőségeiket.
- Biztonságos hozzáférés:** A személyes adatok szerkesztése csak engedélyvel lehetséges. A zárolt mezők biztosítják, hogy érzékeny információk ne módsuljanak illetéktelenül.
- Adatvédelmi intézkedések:** A rendszer titkosított adatkezelést alkalmaz, így a bizalmas ügyféladatok védve vannak az illetéktelen hozzáféréstől.
- Egyszerű szerkesztés:** A „Módosítás mentése” gomb lehetővé teszi az ügyfelek számára az adatok frissítését, ha az adminisztrációs szabályok ezt engedélyezik.
- Intuitív kezelőfelület:** A világos és könnyen olvasható űrlapok, valamint a jól strukturált adatelrendezés segítik az ügyfelek gyors eligazodását.

2.3.7. Kapcsolat és Céginformációk

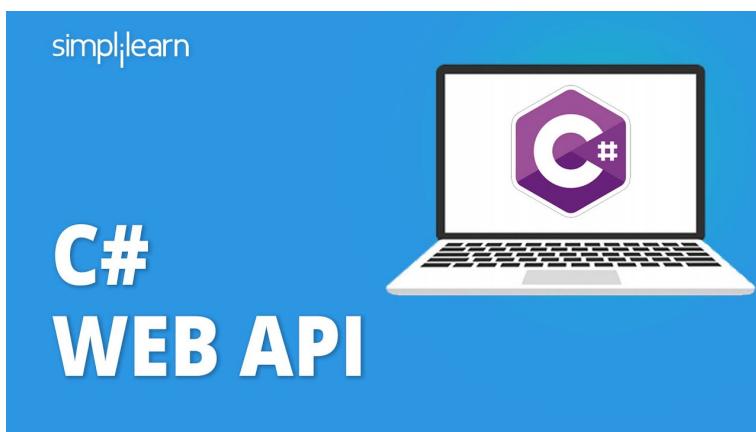
A MAMIK Bank elérhetősége:

- Cím: 3529 Miskolc, Palóczy László utca 3.
- Weboldal: <https://mamikbank.hu>
- Copyright 2025. MAMIK Bank
- Powered by Kerenyir

3529 Miskolc, Palóczy László utca 3.
© Copyright 2025. MAMIK Bank
Powered by Kerenyir

2.4. WPF bemutatása

2.5. C# .Net Web API



2.5.1. A Postman:

- API végpontok tesztelése: HTTP kérések küldése (GET, POST, PUT, DELETE stb.), válaszok megtekintése. Autentikációs módszerek támogatása: Alapértelmezett autentikáció (pl. Basic Auth, Bearer Token), OAuth 2.0 stb. Automatizált tesztelés: Automatizált tesztek írása és futtatása JavaScript segítségével. Kérés és válaszok dokumentálása: Az API végpontok és azok paramétereinek dokumentálása.
- A Postman nagyszerű eszköz a Web API-k fejlesztésében és tesztelésében. Segít a fejlesztőknek az API végpontok kipróbálásában, hibakeresésében, automatizált tesztek futtatásában és az API dokumentálásában. A fent bemutatott módon könnyen tesztelheted a C# .NET Web API-kat a Postman segítségével, így gyorsan validálhat

2.5.2. Unit tesztelés

- A Web API alkalmazások fejlesztésénél fontos a unit tesztelés. A .NET Core-ban használhatunk különböző tesztelési keretrendsereket:
- xUnit: Az egyik legnépszerűbb tesztelési keretrendszer a .NET alkalmazásokhoz. NUnit vagy MSTest: Más tesztelési keretrendszer, amelyek szintén támogatják a .NET-tel való integrációt. Az unit teszteléshez gyakran használt eszközök:
- Moq: A mock-objektumok készítésére szolgáló könyvtár. FluentAssertions: Kifejező, olvasható állításokat (assertion) biztosít a tesztekhez.

2.5.3. Hibakeresés (Debugging)

- A hibakeresés során a Visual Studio vagy Visual Studio Code lehetőséget ad a breakpointok elhelyezésére, az alkalmazás lépésekénti futtatására és változók nyomon követésére.

2.6. Swagger API Dokumentáció

Swagger (más néven OpenAPI) egy szabványos dokumentációs keretrendszer, amely lehetővé teszi a Web API automatikus dokumentálását és interaktív tesztelését. A Swashbuckle.AspNetCore csomag segítségével könnyen integrálhatjuk a Swagger-t a .NET Web API-ba.

Az alábbiakban bemutatjuk a MAMIK Bank backend API végpontjait és azok funkcióit:

2.6.1. BackupRestore

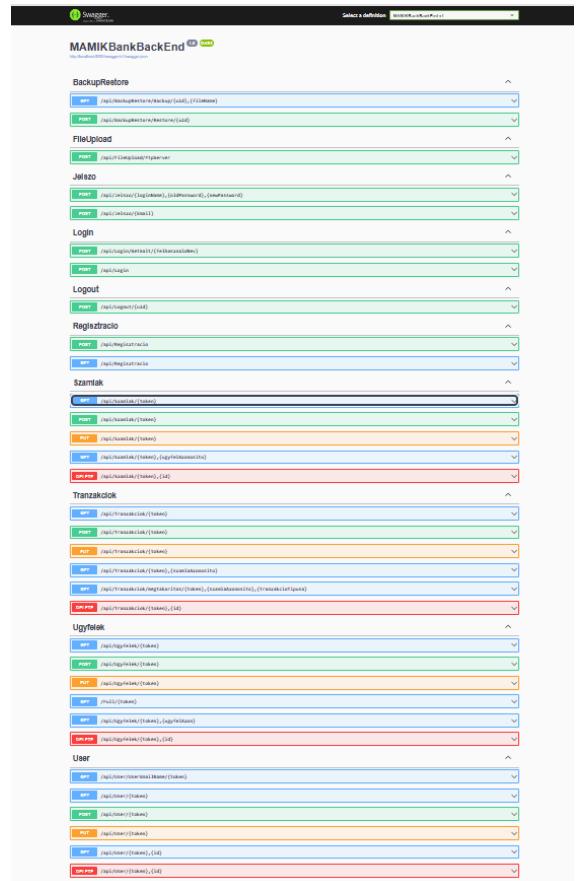
- **Backup készítése:** Lehetővé teszi a banki adatok biztonsági mentését.
- **Backup visszaállítása:** A korábban elmentett adatok visszatöltése az adatbázisba.

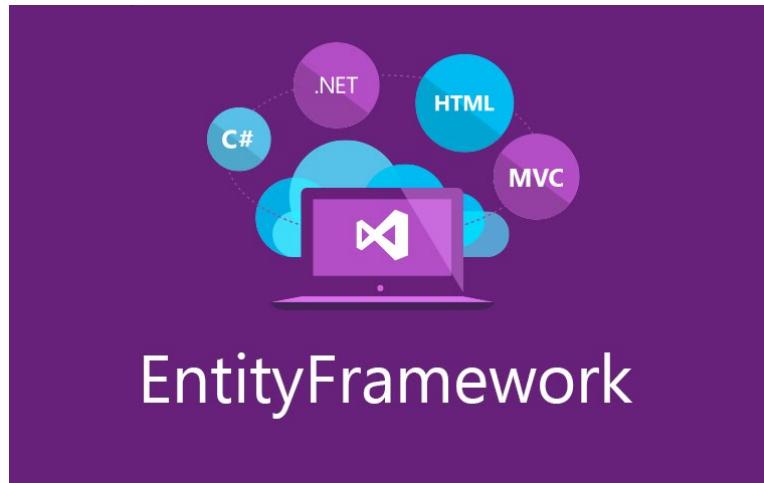
2.6.2. FileUpload

- **Fájlok feltöltése:** A rendszerbe történő fájl feltöltési lehetőség biztosítása.

2.6.3. Jelző

- **Jelzők kezelése:** Figyelmeztetések és egyéb banki értesítések beállítása és frissítése.





2.7. Entity Framework

2.7.1. Az ADO.NET Entitás-keretrendszer

- (EF, Entity Framework) egy objektum-relációs leképező keretrendszer a .NET keretrendszerhez.
- Az ADO.NET Entitás Keretrendszer architektúrája az alábbiakból áll össze növekvő absztrakciós sorrendben:
 - Adatforrás-specifikus kiszolgálók, amely absztrakciós ADO.NET felületeket használnak fel az adatbázis kapcsolathoz, amikor sémák szerint programozunk.
 - Leképzés kiszolgáló, egy adatbázis függő kiszolgáló, amely lefordítja az Entitás SQL utasításfát lekérdezésekkel az adatbázis natív SQL nyelvének megfelelően. Ez magába foglalja a tároló függő hidat, azt a komponenst, amely felelős a generikus utasítások tárolófüggő utasításokra való fordításáért.
 - EDM elemző és nézet leképezés, amely felhasználja az adatmodell SDL specifikációját, és az alapján, ahogyan az leképez a relációs modellre, lehetővé teszi az elméleti modellben történő programozást. A relációs sémából létrehoz nézeteket, amely megfelel az elméleti modellnek. Összegyűjti az információkat különböző táblákból entitásokat kialakítva belőlük, és szétosztja, illetve frissíti az entitás adatait a különböző táblákban, függetlenül attól, mely táblák tartoznak az entitáshoz.
 - Lekérdezés és frissítés csővezeték, azaz pipeline, folyamat-lekérdezések, filterek és frissítési kérelmek küldése, hogy átalakítsa őket kanonikus utasításokká, amelyeket tároló specifikus lekérdezésekkel alakít a leképzés szolgáltató segítségével.
 - Metaadat szolgáltatók, ez kezeli az összes entitáshoz rendelt metaadatot, kapcsolatokat és leképezéseket.

- Tranzakciók, elérhetővé teszik a tár tranzakciós szolgáltatásait. Amennyiben a felhasznált tároló nem támogatja a tranzakciókat, a szükségeknek megfelelően ez a réteg valósítja meg ezeket.
- Koncepcióos réteg API, egy olyan környezet, amely lehetővé teszi a koncepcióos séma programozását. Követi az ADO.NET mintákat Connection objektumok használatával, hogy leképezés szolgáltatókra hivatkozzon, használja a Command objektumokat hogy leképezések küldjön el, és EntityResultSets vagy EntitySets-ként adja vissza a végeredményt.
- Kapcsolat nélküli/leválasztott komponensek, helyileg tárolják az adathalmazt és az entitás halmazokat, hogy kihasználják az ADO.NET Entitás Keretrendszer lehetőségeit egy alkalmanként csatlakozó környezetben.
- Beágyazott adatbázis: az ADO.NET Entitás Keretrendszer magában foglal egy könnyű súlyú adatbázist a relációs adatok felhasználó oldali gyorsítótárazására.
- Tervező eszközök, a Leképezés Tervezőhöz hasonlóan ezek szintén beépítettek az ADO.NET Entitás Keretrendszerbe. Egyszerűsíti a leképezéseken végzett feladatokat egy koncepcióos sémből relációs sémába és meghatározza, hogy az entitás típus adott tulajdonságai mely táblához tartoznak az adatbázisban.
- Programozási réteg, rajta keresztül férünk hozzá az EDM-hez mint programozási konstrukcióhoz, amely felhasználható a programozási nyelveken.
- Objektum szolgáltatások, automatikusan generálják a kódot CLR osztályokhoz, amelyek az entitásnak megfelelő tulajdonságokkal rendelkeznek, ezzel lehetővé téve az entitások .NET objektumokként való példányosítását.
- Web szolgáltatások, az entitások segítségükkel web szolgáltatásként elérhetőek.
- Magas szintű szolgáltatások, mint például a jelentés szolgáltató, amely entitásokkal dolgozik relációs adatok helyett.

2.7.2. CRUD:

- CRUD a Create (Létrehozás), Read (Olvasás), Update (Módosítás) és Delete (Törlés) rövidítése. Ezek a négy alapvető művelet végezhetők el adatokkal adatbázisban vagy bármely más tartós tárolóban. A CRUD műveletek elengedhetetlenek az alkalmazásokban található adatok kezeléséhez és manipulálásához.

Összegzés

A banki alkalmazásunk egy modern és biztonságos pénzügyi platform, amely különféle technológiák és szolgáltatások integrációjával biztosítja a felhasználók számára a gör-dülékeny és hatékony banki ügyintézést.

Adatbáziskezelés

A rendszerünk MySQL és MariaDB adatbázisokat használ, amelyek biztosítják az adatok gyors és megbízható tárolását. Az adminisztrációs feladatokhoz a PHPMyAdmin felületet alkalmazzuk.

Frontend és Backend Technológiák A frontend fejlesztéséhez React Native technológiát alkalmazunk, amely platformfüggetlen mobilalkalmazásokat tesz lehetővé. A backend oldalon a .NET Web API keretrendszeret használjuk, biztosítva a stabil és biztonságos adatkezelést. A Postman tesztelési eszközzel ellenőrizzük az API működését.

Swagger API Dokumentáció

A rendszerünk Swagger API dokumentációval rendelkezik, amely részletes információkat nyújt a végpontok működéséről, beleértve a biztonsági mentéseket, fájlok feltöltését, regisztrációt, számlák és tranzakciók kezelését, valamint az ügyfelek adatait.

Megjelenés és Funkcionalitás

A weboldalunk modern és letisztult dizájnnal rendelkezik, amely könnyen navigálható és biztosítja az összes szükséges banki funkciót. A felhasználói élmény és a biztonság kiemelt szerepet kap, a személyes adatok védelme érdekében.

Köszönetnyilvánítás

Ezúton szeretnénk kifejezni hálánkat és köszönetünket mindenazonknak, akik hozzájárultak a vizsgaremekünk elkészítéséhez és sikeres megvalósításához.

Különösen szeretnénk megköszönni Deák Csabának, Kerényi Róbertnek és Németh Bencének az értékes segítséget, szakmai tanácsokat és a projekt során tanúsított támogatást. Nélkülük nem sikerült volna ilyen szintű eredményt elérni a banki alkalmazás program megvalósításában.

Köszönjük az együttműködést, az ötleteket, és minden egyes pillanatot, amelyet a közös munka során együtt tölthettünk!



MAMIKBANK
Easy money management

Irodalomjegyzék

- [1] PHPMYADMIN: *Az információkat a wikipédiáról szereztük az adatbázis alapjairól*, 2024.10.22-én, <https://en.wikipedia.org/wiki/PhpMyAdmin>
- [2] REACT NATIVE: *Az információkat a React oldaláról és a wikipédiáról gyűjtöttük össze a react alapjairól*, 2024.11.5-én, <https://hu.legacy.reactjs.org/tutorial/tutorial.html>
- [3] .NET WEB API: *Az információkat a Loadfocus oldaláról gyűjtöttük össze a .NET Web API alapjairól*, 2024.11.18-án, <https://loadfocus.com/hu-hu/glossary/crud-operations-what-is-crud>
- [4] ENTITY FRAMEWORK: *Az információkat a Wikipédia oldaláról gyűjtöttük össze a Entity Framework alapjairól*, 2024.11.21-én, https://en.wikipedia.org/wiki/Entity_Framework
- [5] JWT: *Az információkat a JWT oldaláról gyűjtöttük össze a JWT(Json Web Token) alapjairól*, 2024.11.21-én, <https://jwt.io/>

