



VAS VÁRMEGYEI SZAKKÉPZÉSI CENTRUM
HORVÁTH BOLDIZSÁR
KÖZGAZDASÁGI ÉS INFORMATIKAI
TECHNIKUM

A 5 0613 12 03 számú Szoftverfejlesztő és –tesztelő vizsgaremek

**A MeloGo
dokumentációja**

Készítették:

ÁR JÁNOS DÁNIEL
CZÁLL VIKTÓRIA
CSORBA BÁLINT

SZOMBATHELY

2025

Tartalomjegyzék

Tartalomjegyzék	1
1. Bevezetés	3
1.1. A projekt célja	3
1.2. Felhasználási terület – célközönség.....	3
2. Fejlesztési környezet és technológiák	4
2.1. Használt eszközök és környezet	4
2.2. Backend technológiák.....	4
2.3. Frontend technológiák	4
2.4. Verziókezelés.....	4
3. Rendszerkövetelmények	5
3.1. Általános követelményleírás, hardver és szoftver	5
Szerveroldali követelmények	5
Kliensoldali követelmények	5
3.2. Függőségek, API-k	5
Backend függőségek.....	5
Frontend függőségek	5
4. Rendszerterv és adatmodell	6
4.1. A rendszerről általánosságban	6
Funkciók	6
Főbb entitások	6
4.2. Feladatok felosztása.....	6
4.3. ER-modell.....	6
4.4. Relációs adatmodell, kapcsolatok, szabályok.....	6
Relációs adatmodell.....	6
Táblák részletes leírása	7
5. A program működése	10
5.1. Komponensek, technikai és műszaki leírásuk, kapcsolatuk	10
Adatbázis	10
Backend	11
Frontend.....	20
PHP	36
5.2. A program elindítása	36
5.3. A program használata, fő elemei	37
6. A program tesztelése	39
6.1. Unit tesztek	39

7. Összefoglaló és továbbfejlesztési lehetőségek	42
7.1. Összefoglalás	42
7.2. Továbbfejlesztési lehetőségek	42
1.számú melléklet – Feladatok felosztása.....	44
2. számú melléklet – Adatbázis séma	45
3. számú melléklet – Programhasználati útmutató	47

1. Bevezetés

1.1. A projekt célja

Célunk egy olyan weboldal létrehozása volt, amely kifejezetten alkalmi munkák hirdetésére szolgál. A platform egyszerű, gyors és átlátható módon köti össze a munkát keresőket a munkaadókkal, lehetőséget teremtve arra, hogy rövid távú munkák könnyen elérhetővé váljanak mindenki számára.

Az alkalmi munkák nemcsak diákok és fiatal felnőttek számára jelentenek kiegészítő jövedelmi forrást, hanem idősebb embereknek is segíthetnek a mindennapi feladatok elvégzésében. A világ számos részén az idősödő populáció folyamatosan növekszik, és sokan nehezen birkóznak meg olyan teendőkkel, mint a kertészkedés, takarítás vagy kisebb javítások. Emellett az egyedüllét és a társas kapcsolatok hiánya is komoly kihívást jelent számukra.

A platformunk nemcsak fizikai segítségnyújtást tesz lehetővé, hanem lehetőséget biztosít társas kapcsolatok építésére is, például amikor egy segítő elkíséri őket az orvoshoz vagy egyéb ügyintézéshez. Az ilyen interakciók hozzájárulhatnak a magányérzet csökkentéséhez és a mentális jólét javításához.

Mivel az idősebb felhasználók számára különösen fontos a biztonság és a megbízhatóság, a weboldal kiemelt figyelmet fordít a szolgáltatók ellenőrzésére, értékelésére és hitelesítésére. Ezzel biztosítjuk, hogy a segítségre szorulóknak valóban megbízható személyektől kapjanak támogatást. Emellett a szoftver ösztönzi az idősebb felhasználókat a digitális technológiák elsajátítására, amely hosszú távon növeli az önállóságukat és magabiztosságukat a modern világban. Az online hirdetések böngészése és a platform használata elősegítheti a digitális írástudás fejlődését is.

1.2. Felhasználási terület – célközönség

A weboldal széles felhasználói kör számára kínál előnyöket, különböző élethelyzetekhez és igényekhez igazodva:

- Diákoknak és fiatal felnőtteknek rugalmas munkalehetőséget biztosít, amely lehetőséget ad kiegészítő jövedelem szerzésére a tanulmányok vagy más elfoglaltságok mellett.
- Családok és elfoglalt dolgozók számára egyszerű megoldást kínál arra, hogy gyorsan és megbízhatóan találjanak segítséget háztartási, karbantartási vagy egyéb alkalmi feladatok elvégzésére.
- Időseknek lehetőséget nyújt arra, hogy könnyen találjanak megbízható segítőket mindennapi teendőikhez, például bevásárláshoz, takarításhoz vagy kisebb javításokhoz. Emellett a társas interakciók révén csökkentheti az elszigeteltség érzését.
- Önkéntesek és közösségi szervezetek részére egy platformot biztosít jótékonyági események, közösségi szolgáltatások és szociális programok népszerűsítésére, ezáltal támogatva a helyi közösségek erősödését.

A weboldal tehát egy sokoldalú, mindenki számára elérhető és könnyen használható megoldást kínál az alkalmi munkák és segítségnyújtás megszervezésére.

2. Fejlesztési környezet és technológiák

2.1. Használt eszközök és környezet

Fejlesztői környezet: Visual Studio 2022, Visual Studio 2019, Visual Studio Code, Canva, Office 365, Office 2019

Szerver: XAMPP (Apache, MySQL, PHPMyAdmin)

Operációs rendszer: Windows 10/Windows 11

Tesztelés: C# NUnit (API teszteléshez), böngésző fejlesztői eszközei (Chrome)

2.2. Backend technológiák

Programozási nyelv: C#

Keret: ASP.NET

Adatbázis-kezelés: MySQL (PHPMyAdmin felületen keresztül, XAMPP szerver segítségével)

Autentikáció: JWT (JSON Web Token) alapú hitelesítés

API: RESTful API-k a frontend és backend közötti kommunikációhoz

Adatkezelés: Entity Framework

2.3. Frontend technológiák

HTML5, CSS3 – az alapvető felépítés és formázás

JavaScript – dinamikus tartalmak és interakciók

Fetch API – a backend kommunikáció gyorsítására

A cél egy egyszerű, gyors és könnyen használható felhasználói felület létrehozása volt, amely különböző eszközökön is megfelelően működik.

2.4. Verziókezelés

A projekt során a **Git** verziókezelő rendszert használtunk a forráskód nyomon követésére és a változások kezelésére. Lehetővé tette a párhuzamos fejlesztést, az előző állapotok visszaállítását, valamint a csapatmunka hatékony támogatását.

A forráskód tárolása és menedzselése a **GitHub** platformon történt, ennek számos előnye volt:

- **Központi tárolás:** a projekt bármikor elérhető bárhol
- **Branch kezelés:** külön ágakon zajlott a fejlesztés és a hibajavítás, majd ezek összeolvasztása *main* ágba történt
- **Verziókövetés:** minden *commit* egyértelműen dokumentálja a változásokat
- **Együttműködés:** A *pull requestek* segítségével a csapatmunka átláthatóbbá vált

3. Rendszerkövetelmények

A rendszer megfelelő működéséhez meghatározott hardver- és szoftverkövetelmények szükségesek, valamint figyelembe kell venni a külső függőségeket és API-kat, amelyek a fejlesztés során beépítésre kerültek.

3.1. Általános követelményleírás, hardver és szoftver

Szerveroldali követelmények

A backend stabil és megbízható futtatásához ajánlott egy megfelelően konfigurált szerver.

Processzor: 1GHz vagy gyorsabb

Memória (RAM): min. 4GB

Tárhely: min. 16GB

Operációs rendszer: Windows 10/Windows 11 (64-bites)

Szükséges szoftverek:

- Visual Studio 2019
- MySQL Server
- PHPMyAdmin
- XAMPP

Kliensoldali követelmények

A felhasználók böngészőből érik el a rendszert, ezért fontos a megfelelő kompatibilitás.

Ajánlott böngésző: Google Chrome

Eszközök: Asztali számítógép, laptop, táblagép, okostelefon

3.2. Függőségek, API-k

Backend függőségek

A rendszer fejlesztése során külső könyvtárakat és keretrendszereket használtunk.

ASP.NET – szerveroldali logika és API-k

Entity Framework – adatbázis-kezelés

JWT (JSON Web Token) – hitelesítés

MySQL.Data – MySQL adatbázis-kezelő könyvtár

Frontend függőségek

Fetch API - aszinkron adat lekérdezések

Ezek az eszközök és függőségek biztosítják, hogy a rendszer stabilan, gyorsan és biztonságosan működjön.

4. Rendszerterv és adatmodell

4.1. A rendszerről általánosságban

Funkciók

- Felhasználók regisztrációja és belépése
- Munkák felvitele, módosítása, törlése
- Jelentkezés állásokra
- Keresési funkció (munkanév, kulcsszavak alapján)

Főbb entitások

- Felhasználó – munkakereső és/vagy munkaadó
- Munka (Feladat) – egy meghirdetett munka
- Jelentkezés – kapcsolat a felhasználó és a munka között (ki mire jelentkezett)

4.2. Feladatok felosztása

A feladatok szétosztása és a projekt határidők követése táblázat segítségével történt.

A táblázat a dokumentáció **1. számú mellékletében** – [1.ábra](#) – található.

4.3. ER-modell

Egy felhasználó egyszerre lehet munkáltató és munkavállaló is. Tehát jelentkezhet úgy állásra, hogy közben ő is adott fel munkahirdetést. Egy felhasználó értékelheti a másikat, ez egy fajta visszajelzés a többi felhasználónak is. A feladatokat kategóriákba soroltuk, egy feladatra több kategória is érvényes lehet. A felhasználó, az adott feladatot el tudja menteni, ha később esetleg valami miatt szüksége lenne rá. Az adatbázis ER modell terve a **2. számú mellékletben** – [2.ábra](#) – található.

4.4. Relációs adatmodell, kapcsolatok, szabályok

Relációs adatmodell

3NF-be helyezés után a relációs adatmodell a következőképpen néz ki:

FELHASZNÁLÓ (user_id, jelszo, email, szuldat, veznev, kernev, profilkep, bio, regdatum, felhtipus)

FELADAT (task_id, statusz, helyszin, cím, posztdatu, hatarido, leiras, idotartam, fizetes, user_id)

FELADATKATEGÓRIA (task_id, kat_id)

JELENTKEZÉSEK (statusz, latta_e, jeldatum, user_id, task_id)

MENTÉS (user_id, task_id)

KATEGÓRIA (kat_id, katnev)

ÉRTÉKELES (ert_id, erdatum, comment, ertekeles, user_id)

Táblák részletes leírása

FELHASZNÁLÓ

Ez a tábla tárolja a regisztrált felhasználók adatait. Egy felhasználó lehet munkakereső és munkaadó is egyszerre.

user_id – A felhasználó egyedi azonosítója (elsődleges kulcs)
jelszo – A felhasználó jelszava
email – A felhasználó email-címe (egyedinek kell lenni)
szuldat – Születési dátum (pl. 2005.09.26)
veznev – Vezetéknév (pl. „Kiss”)
kernev – Keresztnév (pl. „Béla”)
profilkep – Profilkép elérési útja
bio – Rövid bemutatkozás
regdatum – Regisztráció dátuma (pl. 2025.04.07. 9:28:12)
felhtipus – Felhasználó típusa („user”, „admin”)

Kapcsolat

- 1 : N kapcsolat a **FELADAT** táblával (egy felhasználó több feladatot hozhat létre)
- M : N kapcsolat a **JELENTKEZÉSEK** táblán keresztül a **FELADAT** táblához
- M : N kapcsolat a **MENTÉS** táblán keresztül a **FELADAT** táblához
- 1 : N kapcsolat az **ÉRTÉKEKELÉS** táblán keresztül (egy felhasználó több másikat is értékelhet)

FELADAT

Ez a tábla tartalmazza a felvitt munkákat/feladatokat.

task_id – A feladat egyedi azonosítója (elsődleges kulcs)
statusz – A feladat állapota („nyitott”, „folyamatban”, „lezárt”)
helyszin – Munkavégzés helyszíne (pl. „Győr”)
cím – A munkavégzés pontos helyszíne („Liliom utca 3.”)
posztdatum – Közzététel dátuma (pl. 2025.04.07. 15:07:12)
hatarido – Jelentkezési határidő (pl. 2025.04.11)
leiras – A feladat leírása (A feladat általános leírása)
idotartam – A munka időtartama (pl. „2 óra”)
fizetes – Fizetés összege (pl. „2000 Ft.”)
user_id – A feladat létrehozójának azonosítója (idegenkulcs (**FELHASZNÁLÓ**))

Kapcsolat

- Több **JELENTKEZÉS** kapcsolódhat hozzá
- Több **KATEGÓRIA** is kapcsolódhat hozzá a **FELADATKATEGÓRIA** táblán keresztül
- Több felhasználó is elmentheti a **MENTÉS** táblán keresztül

JELENTKEZÉSEK

Ez a tábla rögzíti a jelentkezéseket a feladatokra.

Id – A jelentkezés egyedi azonosítója (elsődleges kulcs)
statusz – Jelentkezés állapota („*függőben*”, „*elfogadva*”, „*elutasítva*”)
latta_e – A feladat kiírója látta-e a jelentkezést (*bool* érték)
jel_datum – A jelentkezés dátuma (pl. 2025.04.07. 15:07:12)
user_id – A feladatra jelentkezett felhasználó azonosítója (idegenkulcs (**FELHASZNÁLÓ**))
task_id – A feladat azonosítója (idegenkulcs (**FELADAT**))

Kapcsolat

M:N kapcsolat a **FELHASZNÁLÓ** és **FELADAT** között, ez egy kapcsolótábla.

MENTÉS

Ez a tábla tárolja, hogy egy felhasználó mely feladatokat mentette el.

user_id – A felhasználó azonosítója (idegenkulcs (**FELHASZNÁLÓ**))
task_id – A mentett feladat azonosítója (idegenkulcs (**FELADAT**))

Kapcsolat

M:N kapcsolat a **FELHASZNÁLÓ** és **FELADAT** között, ez egy kapcsolótábla.

KATEGÓRIA

Ez a tábla tárolja a feladatokhoz tartozó kategóriákat.

kat_id – Kategória azonosító (elsődleges kulcs)
katnev – A kategória neve (pl. „*takarítás*”, „*kertészkedés*”, „*orvoshoz kísérés*”)

FELADATKATEGÓRIA

Kapcsolótábla a feladatok és kategóriák között, mivel egy feladat több kategóriába is tartozhat.

task_id – A feladat azonosítója (idegenkulcs (**FELADAT**))
kat_id – A kategória azonosítója (idegenkulcs (**KATEGÓRIA**))

Kapcsolat

M:N kapcsolat a **FELADAT** és **KATEGÓRIA** között, ez egy kapcsolótábla

ÉRTÉKELÉS

Ez a tábla rögzíti a felhasználók közötti értékeléseket, visszajelzéseket.

ert_id – Az értékelés egyedi azonosítója (elsődleges kulcs)
erdatum – Az értékelés dátuma (pl. 2025.04.07. 15:07:12)
comment – Szöveges vélemény (*A felhasználó elmondhatja a véleményét több másíkról.*)
ertekeles – Értékelés (1–5)
ertekelt_id – Az értékelt személy azonosítója (idegenkulcs (**FELHASZNÁLÓ**))
ertekelo_id – Az értékelő személy azonosítója (idegenkulcs (**FELHASZNÁLÓ**))

Kapcsolat

M : N kapcsolat a **FELHASZNÁLÓ** táblán belül (egy felhasználó több másikat is értékelhet)

Ezek alapján a relációs modell vizualizációja a dokumentáció **2. számú mellékletében** - [3. ábra](#) - található.

5. A program működése

5.1. Komponensek, technikai és műszaki leírásuk, kapcsolatok

A rendszerünket négy főkomponens alkotja: **Adatbázis**, **Backend**, **Frontend**, és **PHP**. Munkánk során minden egyes komponens szorosan kapcsolódik egymáshoz, hogy egy működőképes és hatékony alkalmazást hozzunk létre. Az alábbiakban ezt a négy komponenst részletezzük, beleértve ezek technikai és műszaki leírását, valamint kapcsolatát a többi komponenssel.

Adatbázis

Az adatbázis a rendszer központi tárolója, amely az alkalmazás működéséhez szükséges összes adatot (felhasználók, feladatok, jelentkezések stb.) tárolja. A megfelelő adatbázis-struktúra kulcsfontosságú az adatok gyors és biztonságos eléréséhez, valamint az alkalmazás teljesítményének biztosításához.

Technikai leírás

A rendszer adatbázisa relációs típusú, és **MySQL** adatbázist használunk a táblák és kapcsolatok kezelésére, mert széles körben használt, jól skálázható és teljesítményorientált adatbázis-kezelő rendszer, amely ideális választás a webes alkalmazások számára. A rendszer **CRUD** (*Create, Read, Update, Delete*) műveletek segítségével kezeli az adatokat.

A backend és az adatbázis közötti kommunikációt Web API (*Web Application Programming Interface*) keresztül valósítottuk meg. A Web API lehetővé teszi a frontend és a backend közötti adatcserét **HTTP/HTTPS** protokollon keresztül, biztosítva az adatkezelés és a működés rugalmasságát. Az API RESTful architektúrával működik, így könnyen integrálhatók és skálázhatók, miközben a backend logika és az adatbázis közötti kapcsolatot egyszerűsítik.

Műszaki leírás

Kapcsolódás

Az adatbázishoz való kapcsolódás az alkalmazás backendjén keresztül történik egy **Web API** segítségével, amely az alkalmazás logikáját és az adatbázissal való interakciót a frontend felé szolgáltatja. Az alkalmazás a **RESTful API-t** (*Representational State Transfer Application Programming Interface*) használja az adatok lekérésére és manipulálására, amelyek HTTP-metódusok (*GET, POST, PUT, DELETE*) révén működnek. Az API a HTTP-kérések és válaszok formájában kommunikál a frontend és a backend között.

Az adatbázisok közötti adatkezelés közvetlenül a **SQL adatbázissal** kommunikál, amelyet XAAMP és phpMyAdmin segítségével hoztunk létre. Az API megkönnyíti a backend logikai részének és az adatbázis-interakcióknak az elválasztását, így biztosítva, hogy az adatbázis-kezelés ne zavarja meg a felhasználói interakciókat a frontend oldalon. Mi közvetlenül Web API alapú adatkommunikációval valósítjuk meg az adatkezelési műveleteket.

Adatbiztonság

Az alkalmazás különös figyelmet fordít az adatok biztonságos kezelésére és tárolására. Minden érzékeny adat (például felhasználói jelszavak) titkosítva van, hogy megakadályozzuk az adatok jogosulatlan hozzáférését. A titkosításhoz biztonságos algoritmust alkalmazunk, (**bcrypt**), amely biztosítja a felhasználók hitelesítő adatainak védelmét az adatbázisban.

Ezen kívül a jelszavakat nem tároljuk nyílt szöveggént, hanem csak azokat a titkosított formákat tároljuk, amelyeket az alkalmazás az autentikációs folyamat során visszafejt.

Struktúra

Az adatbázis relációs struktúrája biztosítja az adatok rendszerezését és kapcsolódását a különböző entitások között. A táblák közötti kapcsolatokat **idegen kulcsok** (*foreign key*) biztosítják, amelyek alapvető szerepet játszanak a referenciális integritás fenntartásában. Ezek az idegen kulcsok meghatározzák, hogy mely táblák között kell fenntartani a kapcsolatokat, és biztosítják, hogy az adatbázisban ne legyenek olyan adatbejegyzések, amelyek sértik a relációk szabályait.

Például egy felhasználó adatainak tárolásához használt táblán belül a felhasználói azonosító (`user_id`) idegen kulcsként kapcsolódik a **FELADAT** táblához, amely biztosítja, hogy minden feladathoz hozzárendeljünk a megfelelő felhasználót. Ha egy feladatot törölünk, akkor az azt kapcsolódó rekordokat is törölni vagy frissíteni kell az integritás megőrzése érdekében.

Vizualizációja a **2.számú mellékletben** [4.ábra](#) – található.

A Web API felelős azért, hogy megfelelően kezelje az adatbázisban lévő adatokat a kérések révén. Minden műveletet API végpontok kezelnek, és biztosítják, hogy az adatbázis-táblákban lévő adatok a megfelelő módon módosuljanak a felhasználói műveletek alapján.

Az adatbázis integritását biztosítja, hogy az adatok ne legyenek sérültek, és minden művelet végrehajtása után az adatok érvényesek maradjanak a rendszer minden szintjén. Az adatbázis folyamatos karbantartása és optimalizálása garantálja a gyors adatlekérést és a megfelelő adatkezelési sebességet a rendszer bármely területén.

Példaadatok

Az adatbázisban szereplő példaadatok generálása Mesterséges Intelligencia használatával történt. A ChatGPT kérésünkre mindegyik táblába készített adatokat, köztük létrehozva a kapcsolatokat.

Példa egy adatra:

```
INSERT INTO `felhasznalo` (`user_id`, `jelszo`, `email`, `szul-  
dat`, `veznev`, `kernev`, `profilkep`, `bio`, `regdatum`, `fel-  
htipus`, `telefonszam`) VALUES  
(1, '871b32b5f4e1b9ac25237dc7e4e175954c2dc6098aade48a8abefb585cb  
d53f2', 'felhasznalo1@test.com', '1990-05-10', 'Kovács', 'An-  
na', NULL, 'Szeretem az állatokat', '2024-10-16 19:17:45',  
'user', '+36 30/123-4567');
```

Backend

A backend komponens az alkalmazás központi része, amely a logikát, az adatkezelést, valamint a külső rendszerekkel való kommunikációt kezeli. A backend biztosítja az adatbázis interakciókat, az API-k működését, valamint a felhasználói kérések megfelelő feldolgozását.

Technikai leírás

A backend komponens nyelve **C#**, amelyet az **ASP.NET Web API** keretrendszer segítségével valósítottunk meg. Ez a modern, platformfüggetlen fejlesztői környezet lehetővé teszi RESTful API-k létrehozását, így a frontend és más külső rendszerek HTTP-n keresztül kommunikálhatnak az alkalmazással.

Műszaki leírás

Kapcsolódás

A backend komponens .NET Framework alapú, és az adatbázissal való kapcsolatot WebAPI segítségével valósítja meg. Az adatbázissal való kapcsolódás Entity Framework ORM használatával történik, amely a relációs adatbázis-táblákat objektumként kezeli a kódban.

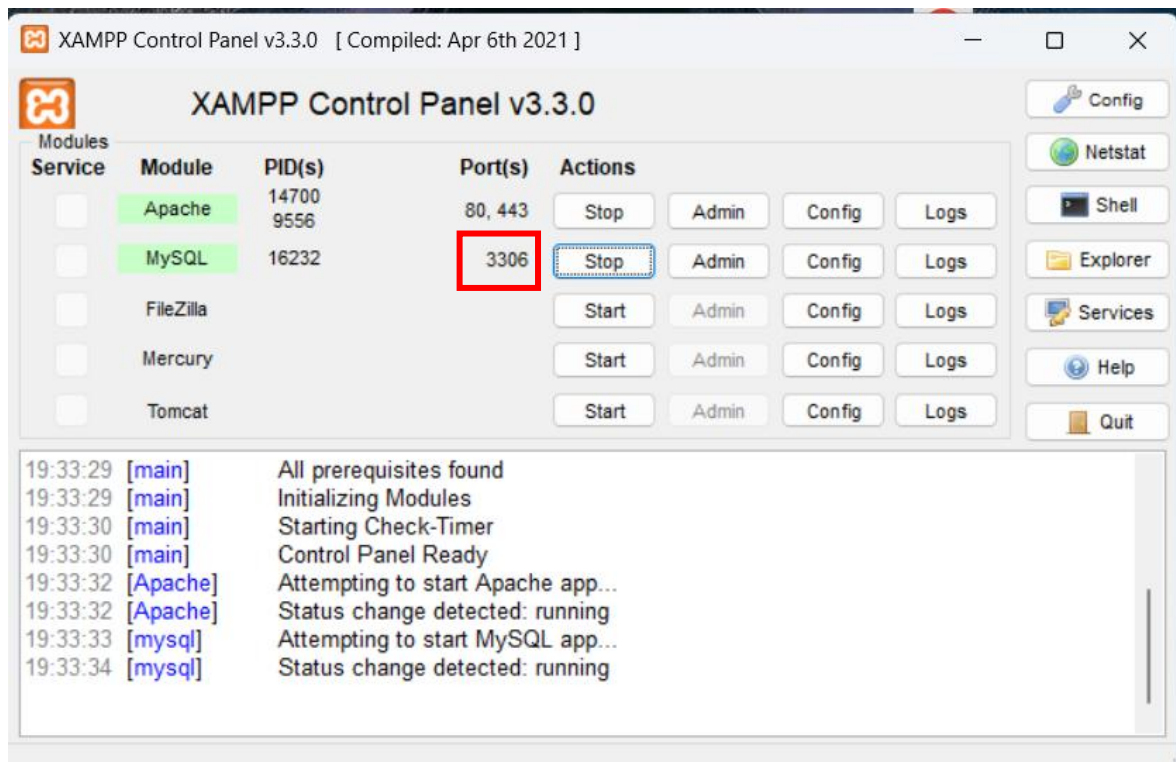
A kapcsolat létrehozása a `Web.config` fájlban definiált **connection string** alapján történik.

```
<connectionStrings>
<add name="WebCon" connectionString="server=jannn1.hu; data-
base=c1604_projekt;user=c1604_projekt;password=projekt;" provi-
derName="MySQL.Data.MySqlClient" />
</connectionStrings>
```

Ha szeretnénk a webservert helyett lokális szerveren futtatni a következőképpen kell átírnunk:

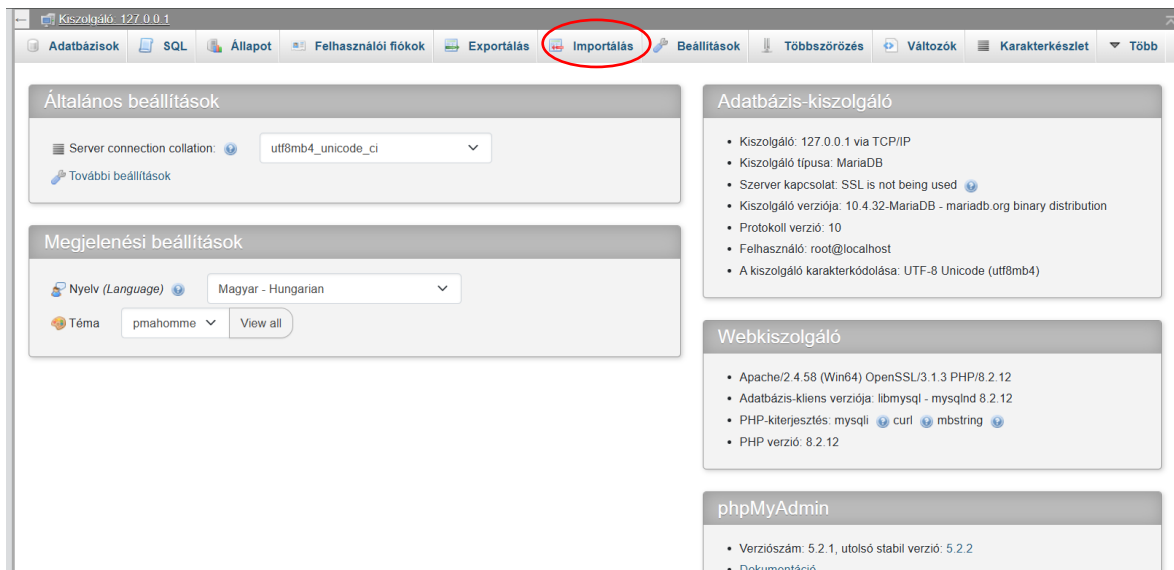
```
<connectionStrings>
<add name="WebCon" connectionString="server=localhost;
port=3306; database=c1604_projekt; user=root; password="; provi-
derName="MySQL.Data.MySqlClient"/>
</connectionStrings>
```

Figyelem! A port változhat, de általában **3306** vagy **3307**. Ezt onnan tudjuk kideríteni, hogy a **XAMPP**-on belül elindítjuk az **Apache**-t és a **MySQL**-t.



Ezek után a MySQL Admin-ra kattintva a következő felület nyílik meg:

Válasszuk az **Importálást**, majd töltsük be az adatbázis export filejét.



A WebContext osztályban konfiguráltuk a modellek és táblák közötti kapcsolatot:

```
public class WebContext : DbContext
{
    public DbSet<Ertekeles> Ertekeles {get; set;}
    public DbSet<Feladat> Feladat {get; set;}
    public DbSet<FeladatKategoria> FeladatKategoria {get; set;}
    public DbSet<Felhasznalo> Felhasznalo {get; set;}
    public DbSet<Jelentkezesek> Jelentkezesek {get; set;}
    public DbSet<Kategoria> Kategoria {get; set;}
    public DbSet<Mentes> Mentes {get; set;}

    public WebContext() : base("name=WebCon") { }
}
```

Adatbiztonság

A backend szigorú adatvédelmi szabályokat alkalmaz az érzékeny információk – például jelszavak – kezelésére. A jelszavakat nem tároljuk nyílt szöveggént, hanem **títkosított formában**, jellemzően **bcrypt** alapú algoritmussal. A hashelt jelszavakat az adatbázisban tároljuk, így külső támadások esetén az eredeti jelszó nem visszafejthető. Ehhez segítségül a WebAPI egy csomagját/függőségét használtuk:

```
using BCrypt.Net;
```

Az ellenőrző metódusunk a következőképpen néz ki:

```
public class Validator
{
    private WebContext _context;

    public Validator(WebContext context)
```

```

{
    _context = context;
}

public static string HashPassword(string password)
{
    // a jelszó hash kódját állítja elő
    return BCrypt.Net.BCrypt.HashPassword(password);
}

public static bool VerifyPassword(string password, string
hashedPassword)
{
    // ellenőrzi a kapott és a hash kódjával tárolt jelszavakat
    return BCrypt.Net.BCrypt.Verify(password, hashedPassword);
}
}

```

Hitelesítés és jogosultságkezelés

A rendszer **JWT** (*JSON Web Token*) alapú autentikációs megoldást használ. A bejelentkezés után a felhasználó kap egy digitálisan aláírt tokenet, amelyet minden API-hívásnál meg kell adnia.

Ehhez a szükséges csomag:

```
using System.IdentityModel.Tokens.Jwt
```

A token tartalmazza a felhasználói azonosítót és jogosultságokat, és a **Headers** részben kerül továbbításra:

Authorization: Bearer <token>

```

public override void OnAuthorization(HttpContext action-
Context)
{
    // Lekérjük a fejléct a kérésből ("Bearer <token>")
    var authHeader = actionContext.Request.Headers.Authorization;
    // Ha nincs fejléc, vagy nem "Bearer" típusú, vagy nincs token érték megadva, akkor visszatér-
    rünk 401-es hibával
    if (authHeader == null || authHeader.Scheme != "Bearer" ||
string.IsNullOrEmpty(authHeader.Parameter))
    {
        // Unauthorized válasz küldése – a kliens nem küldött érvényes hitelesítést
        actionContext.Response = actionContext.Request.CreateRes-
        ponse(HttpStatusCode.Unauthorized, "Hiányzó vagy érvénytelen
        token!");
        return;
    }
}

```

```

        // A token dekódolása – visszaad egy felhasználói objektumot (várhatóan benne van a
        típus is)
        var user = TokenManager.DecodeToken(authHeader.Parameter);

        // Ellenőrizzük, hogy sikerült-e dekódolni a tokent, a típus értelmezhető-e enumként, és
        hogy a típus szerepel-e az engedélyezett típusok listájában
        if (user == null || !Enum.TryParse(user.Felhtipus, out Fel-
        hasznaloTipus userType) || !ut.Contains(userType))
        {
            // Ha nem, akkor 403 Forbidden válasszal megtagadjuk a hozzáférést
            actionContext.Response = actionContext.Request.Cre-
            ateResponse(HttpStatusCode.Forbidden, "Hozzáférés megtagadva");
            return;
        }

        // Ha minden ellenőrzés sikeres, akkor a kérés folytatódik (nem állítjuk be a Response értékét)
    }

```

TokenManager.cs

Egy saját készítésű tokenkezelő a hitelesítéshez, ami email cím és felhasználótípus alapján egy tokenet generál, ezt **base64** kódolással teszi.

GenerateToken(Felhasznalo user)

Összefűzi ezt a kettőt, | karakterrel elválasztva:

```

var bytes = Encoding.UTF8.GetBytes($"{user.Email}|{user.Fel-
htipus}");

```

Majd átalakítja ezt byte tömbbé UTF-8 kódolással:

```

return Convert.ToBase64String(bytes);

```

DecodeToken(string token)

Célja: Egy ilyen base64 formátumú tokenet "visszafejt", és visszaad egy Felhasznalo objektumot az eredeti adatokkal.

```

var bytes = Convert.FromBase64String(token);

```

A base64 tokenből visszanyeri az eredeti byte tömböt.

```

var tokens = Encoding.UTF8.GetString(bytes).Split('|');

```

UTF-8 karakterláncot csinál belőle, majd szétválasztja a | karakter mentén (tehát [email, felhasználótípus] lesz belőle).

```

return new Felhasznalo
{
    Email = tokens[0],
    Felhtipus = ((FelhasznaloTipus)Enum.Parse(typeof(FelhasznaloTipus), tokens[1])).ToString()
};

```

Egy új Felhasznalo objektumot hoz létre a kinyert értékekkel.

Ezek alapján, az alábbi backend kérésre csak az admin jogú felhasználó lesz képes a szerver oldalán.

```
[TokenAuthorize("Admin")]
public IActionResult Delete(int id)
{
    var erkekeles = ctx.Ertekeles.FirstOrDefault(e => e.Ert_Id == id);
    if (erkekeles == null)
    {
        return NotFound();
    }
    try
    {
        ctx.Ertekeles.Remove(erkekeles);
        ctx.SaveChanges();
        return Ok(erkekeles);
    }
    catch (Exception ex)
    {
        return InternalServerError(ex);
    }
}
```

Struktúra

A backend projekt logikailag jól elkülönített rétegekre és mappákra van osztva, hogy átlátható és könnyen karbantartható legyen az alkalmazás.

Models

A *Models* mappában található az adatmodellek, amelyek az adatbázis tábláit reprezentálják osztályok formájában. Minden modell osztály tulajdonságai megfelelnek az adatbázis oszlopainak. Ezeket az osztályokat használjuk az adatbázissal történő interakcióhoz, az adatátvitelhez, valamint az API-n keresztül érkező vagy küldött adatok strukturálásához.

Példa egy modellre:

```
public enum FelhasznaloTipus
{
    Admin,
    User
}
public class Felhasznalo
{
    [Key] //Elsődleges kulcs beállítása
```

```

public int User_Id { get; set; }
public string Jelszo { get; set; }
public string Email { get; set; }
public string SzulDat { get; set; }
public string VezNev { get; set; }
public string KerNev { get; set; }
public string ProfilKep { get; set; }
public string Bio { get; set; }
public string RegDatum { get; set; }
public string Felhtipus { get; set; } = FelhasznaloTi-
pus.User.ToString();
}

```

Az alábbi kód a felhasználó osztályt mutatja.

Controllers

A *Controllers* mappában találhatók azok az osztályok, amelyek a kéréseket kezelik. Ezek az osztályok tartalmazzák az API végpontokat (*GET*, *POST*, *PUT*, *DELETE*, *PATCH*), és felelősek a frontend és a backend közötti kommunikáció lebonyolításáért. A controller metódusok feldolgozzák a kéréseket, kommunikálnak az adatbázissal és válaszokat küldenek vissza a kliensnek.

Példa egy GET metódusra:

```

public IActionResult Get()
{
    var mentettFeladatok = ctx.Mentes
        .Select(m => new
        {
            m.User_Id,
            m.Task_Id,
            Felhasznalo = new
            {
                m.Felhasznalo.User_Id,
                m.Felhasznalo.Email,
                m.Felhasznalo.VezNev,
                m.Felhasznalo.KerNev,
                m.Felhasznalo.ProfilKep
            },
            Feladat = new
            {
                m.Feladat.Task_Id,
                m.Feladat.Cim,

```

```

        m.Feladat.Statusz,
        m.Feladat.Helyszin,
        m.Feladat.PosztDatum,
        m.Feladat.Hatarido
    }
})
.ToList();
return Ok(mentettFeladatok);
}

```

Az alábbi kód az összes mentett feladatot listázza, felhasználókkal összekapcsolva.

WebApiConfig.cs

Ez a fájl a projekt *Startup* mappájában található, és a Web API konfigurációját tartalmazza. Itt állítjuk be az útvonalakat (routing), engedélyezzük a CORS-t, és más beállításokat végzünk, amelyek az API működéséhez szükségesek.

```

public static void Register(HttpConfiguration config)
{
    var cors = new EnableCorsAttribute("http://localhost:3000,http://127.0.0.1:3000", "*", "*");
    config.EnableCors(cors);
    config.MapHttpAttributeRoutes();
    config.Routes.MapHttpRoute(
        name: "DefaultRouteApi",
        routeTemplate: "api/{controller}/{id}",
        defaults: new { id = RouteParameter.Optional }
    );
    config.Formatters.Remove(config.Formatters.XmlFormatter);
    config.Formatters.JsonFormatter.SerializerSettings.ContractResolver = new CamelCasePropertyNamesContractResolver();
    config.Formatters.JsonFormatter.SerializerSettings.ReferenceLoopHandling = Newtonsoft.Json.ReferenceLoopHandling.Ignore;
}

```

Ehhez használt függőségek:

```

using System.Web.Http.Cors;
using Newtonsoft.Json.Serialization;

```

Táblák és Classok kapcsolata

A backend és adatbázis kommunikáció típusproblémáit a következőképpen oldottuk meg:

MySQL adattípus	C# adattípus
int (11)	int

varchar (255)	string
enum	enum
text	string
datetime	string
tinyint (4)	int

Osztályok kapcsolata

A class-ok közötti kapcsolatokat [Key] annotációval, és a modelBuilder metódusok segítségével hidaltuk át. Ehhez a következő csomagot alkalmaztuk:

```
using System.ComponentModel.DataAnnotations;
```

A WebContext.cs nevű osztályban kötöttük össze a classokat.

```
modelBuilder.Entity<Feladat>()
    .HasRequired(f => f.Felhasznalo)
    .WithMany(u => u.Feladatok)
    .HasForeignKey(f => f.User_Id)
    .WillCascadeOnDelete(false);
```

A példa azt mutatja, hogy a Feladat classhoz idegenkulcsként hozzákapcsoltuk a Felhasználó classban lévő User_Id-t.

Ahhoz, hogy ez tényleg gondtalanul működjön, a Felhasználó osztályba a következőt kellett beírni:

```
public ICollection<Feladat> Feladatok {get; set;}
```

Használt válaszok

A backend visszajelzései során a következő kódokat használtuk:

200 - Ok

A kérés sikeresen feldolgozásra került, és az eredmény a válaszban visszaadásra került.

400 - Bad Request

A kérés formátuma vagy tartalma hibás. Ez lehet hiányzó paraméter, érvénytelen adat, vagy nem teljes kérés.

401 - Unauthorized

A kérés nem tartalmaz érvényes hitelesítést (pl. token), vagy a felhasználó nincs bejelentkezve.

404 - Not Found

Az erőforrás nem található. A megadott azonosító alapján nem létezik az adott adat az adatbázisban.

409 - Conflict

Az erőforrás ütközést jelez – általában akkor, ha már létezik az adott adat, amit a felhasználó be akar vinni.

500 - Internal Server Error

A szerveren belül valami hiba történt. Ez általában nem a felhasználó hibája, hanem programozási vagy szerveroldali hiba.

Frontend

A frontend az alkalmazás felhasználói oldala, amely lehetővé teszi a felhasználók számára az interakciót a rendszerrel. A felhasználói felület biztosítja az adatbevitel, adatszolgáltatás és navigáció vizuálisan átlátható és felhasználóbarát módját. A frontend felelős az oldalak megjelenítéséért, az események kezeléséért, valamint a backenddel való kommunikációért.

Technikai leírás

A frontend fejlesztéséhez **HTML**, **CSS** és **JavaScript** nyelveket használtunk. CSS segítségével reszponzív, mobilbarát és modern felületet alakítottunk ki, saját CSS kód mellett.

A frontend *statikus* oldalakkal áll, amelyeket JavaScript segítségével tettünk *dinamikussá*. A felhasználói műveletek – mint például regisztráció, bejelentkezés, hirdetés feladása vagy munkára jelentkezés – során a JavaScript **fetch()** metódus segítségével HTTP-kéréseket küld a C#-ban írt backend felé.

A rendszer **nem használ** frontend frameworköket (pl. React vagy Vue), így a navigáció klasszikus, többoldalas megoldáson alapszik. Az adatok és a funkciók elérése **RESTful API**-kon keresztül történik, amelyek biztosítják a frontend és a backend közötti kapcsolatot.

Műszaki leírás

Technológiai háttér

A frontend fejlesztése során egyszerű, de megbízható webes technológiákat használtunk, amelyek biztosítják az oldal működését minden modern böngészőben.

A weboldal váza és tartalmi struktúrája **HTML5** segítségével készült. Minden oldal külön .html fájlban található, így a fejlesztés átlátható maradt. A HTML elemek szemantikusan lettek használva (pl. `<header>`, `<main>`, `<section>`, `<footer>`), ez megkönnyítette az oldal átláthatóságát és a hibák javítását.

A saját stílusokat külön .css fájlokban kezeltük. A **CSS** lehetővé tette a vizuális testreszabást, például színek, betűtípusok, margók és animációk beállítását. Kisebb egyedi effekteket, árnyékokat, gombstílusokat kézzel írtunk meg.

A JavaScript a weboldal interaktív működéséért felelős. Ennek segítségével oldottuk meg:

- a dinamikus űrlapellenőrzést (pl. *e-mail formátum, jelszó hossz*),
- az API-k hívását (*fetch()* segítségével kommunikál a backenddel),
- eseménykezelést (pl. *gombkattintás, űrlapbeküldés*),
- bejelentkezési állapot tárolását (pl. *localStorage, sessionStorage*),
- oldalelemek megjelenítését vagy elrejtését (pl. *hirdetések betöltése*).

Struktúra

A frontend felépítése logikusan tagolt mappákból és fájlokból áll, amelyek külön kezelik a tartalmat (**HTML**), a megjelenést (**CSS**), az interakciót (**JavaScript**) és az erőforrásokat (képek, ikonok, logó). A cél az volt, hogy átlátható, jól karbantartható legyen a projekt.

HTML

A weboldal felépítése logikusan elkülönített fájlokból áll, ahol minden oldal külön dokumentumban kapott helyet. A szerkezet egyszerű és átlátható, az oldalak közötti navigáció jól strukturált. A HTML oldalak külső CSS fájlokra épülnek.

A következőképpen értük el, hogy az oldal képes legyen használni a CSS-t:

```
<link rel="stylesheet" href="css/login.css">
```

index.html – A kezdőlapot tölti be. Itt található az általános bemutatkozás, navigációs sáv, valamint a fő funkciók bemutatása.

Login.html – A bejelentkezési és regisztrációs oldal. Felhasználóbarát űrlapokat és interaktív visszajelzéseket tartalmaz.

Közös jellemzők

Szerkezeti elemek – A header, main, footer HTML5 tagek egységes szerkezetet biztosítanak az oldalakon.

Ikonhasználat – A *Font Awesome*¹ könyvtár CDN (*Content Delivery Network*) -ről kerül betöltésre, így az oldalak modern, látványos ikonokat tartalmaznak.

Navigáció – Az oldalak között a navigációs sáv segítségével lehet váltani, amely minden oldalon azonos szerkezetű és stílusú.

CSS

A weboldal megjelenését több különálló CSS fájl biztosítja, amelyek logikusan el vannak különítve az oldalak és funkciók szerint, Valamint az alapvető responzív viselkedéshez és stílusokhoz segítség volt.

index.css – Az oldal fő kinézetét szabályozza: menük, főoldal elrendezés, hirdetések megjelenése, általános színek és tipográfia.

login.css – A bejelentkezési és regisztrációs felülethez tartozó stílusokat tartalmazza, külön figyelmet fordítva a felhasználóbarát űrlapokra.

Responzivitás – A saját media query-k segítségével minden oldal jól jelenik meg mobilon és asztali gépen is.

Ikoncsomag beépített használata

Példa egy ikon használatára:

```
<i class="fas fa-star" data-value="4"></i>
```

Javascript

A weboldal dinamikus funkcióit és interaktivitását JavaScript biztosítja. A JavaScript különböző fájlokba van rendezve, hogy az egyes oldalak és funkciók logikáját könnyen kezelhessük. Az alkalmazott JavaScript lehetővé teszi az aszinkron adatkezelést, a felhasználói interakciók kezelését és az oldalon belüli dinamikus tartalom módosítását. Az **eseménykezelés** és a **DOM manipuláció** segítségével biztosítjuk a weboldal interaktivitását és a gördülékeny felhasználói élményt.

index.js

Feladata

Kezeli felhasználói interakciókat, API-hívásokat, dinamikus felületfrissítéseket és a profilkezelést. Ez teszi lehetővé a felhasználók számára, hogy munkákat böngésszenek, mentsenek, jelentkezzenek rájuk, kezeljék saját hirdetéseiket, valamint megtekintsék más felhasználók profiljait és értékeléseit.

Főbb funkciók

¹ <https://cdnjs.cloudflare.com/ajax/libs/font-awesome/6.0.0/css/all.min.css>

Munkaajánlatok:

- Munkaajánlatok „kártyákon” jelennek meg, olyan részletekkel, mint a cím, a fizetés, a helyszín és a státusz.
- A szűrés keresőszavak, helyszínek és kategóriák alapján lehetséges.
- Lehetőség van munkák mentésére/eltávolítására és részletes információk megtekintésére.

Felhasználói panel:

- Lehetőséget biztosít a felhasználói adatok megtekintésére, saját hirdetések kezelésére és jelentkezések áttekintésére.
- A keresési és szűrési funkció a felhasználó saját munkáira is működik.

Profilkezelés:

- Profilképek feltöltése (max. 5 MB, csak képformátum).
- Felhasználói adatok, bio, hirdetett munkák és értékelések megjelenítése.
- Értékelések hozzáadása és megtekintése csillagos rendszerrel.

Főbb függvények

fetchData(url, method, options): Aszinkron API-hívások kezelése GET, POST, PATCH, DELETE metódusokkal, token-alapú hitelesítéssel.

Példa a használatára:

```
async function fetchData(url, method = 'GET', options = {}) {
  try {
    // Változó a fetch kérés törzsének tárolására
    let fetchBody;

    // GET metódus esetén csak fejléceket állítunk be
    if (method === 'GET') {
      fetchBody = {
        method: method, // HTTP metódus (alapértelme-
        zett: GET)
        headers: {
          'Authorization': 'Bearer ' + token, // Bearer token hitelesítéshez
          'Content-Type': 'application/json' // JSON tartalom típus
        }
      };
    } else {
      // POST, PATCH, DELETE stb. esetén a kérés törzsét is hozzáadjuk
      fetchBody = {
        method: method, // HTTP metódus
        headers: {
          'Authorization': 'Bearer ' + token, // Bearer token hitelesítéshez
          'Content-Type': 'application/json' // JSON tartalom típus
        }
      };
    }
  }
}
```

```

    },
    body: JSON.stringify(options) // Kérés törzse JSON formátumban
  };
}

// API-hívás végrehajtása a megadott URL-re és fetchBody-val
const response = await fetch('https://localhost:44300/api/' + url, fetchBody);

// Ellenőrizzük, hogy a válasz sikeres-e (HTTP státusz 200-299)
if (!response.ok) {
  throw new Error(`Hiba: ${response.status} - ${response.statusText}`);
}

// Válasz szöveggént való kiolvasása
const text = await response.text();

// Szöveg JSON-ra konvertálása, ha üres, akkor null
const data = text ? JSON.parse(text) : null;

// Válasz adatainak visszaadása
return data;
} catch (error) {
  // Hiba esetén konzolon való naplózás
  console.error('Fetch hiba:', error.message);

  // Null visszaadása hiba esetén
  return null;
}
}

```

getDate(): Aktuális dátum és idő formázása (YYYY-MM-DD HH:MM).

Példa a használatra:

```

function getDate() {
  // Új Date objektum létrehozása az aktuális időponttal
  const now = new Date();

  // Évszám kinyerése (pl. 2025)
  const year = now.getFullYear();

  // Hónap kinyerése (0-11, ezért +1), két számjegyre formázva (pl. 04)
  const month = String(now.getMonth() + 1).padStart(2, '0');

  // Nap kinyerése, két számjegyre formázva (pl. 23)
  const day = String(now.getDate()).padStart(2, '0');

  // Óra kinyerése, két számjegyre formázva (pl. 09)

```



```

    const hours = String(now.getHours()).padStart(2, '0');
    // Perc kinyerése, két számjegyre formázva (pl. 45)
    const minutes = String(now.getMinutes()).padStart(2, '0');
    // Formázott dátum és idő összeállítása: ÉÉÉÉ-HH-NN ÓÓ:PP
    const formatted = `${year}-${month}-${day} ${hours}:${minutes}`;
    // Formázott dátum visszaadása
    return formatted;
}

```

createjobcard(data, edit): Munkaajánlat kártyák dinamikusan létrehozása, szerkesztési opcióval (pl. jelentkezések kezelése).

Példa a használatra:

```

// Ha szerkesztési mód van engedélyezve (edit = true), egy "Jelentkezések" gombot adunk a kártyához
if (edit) {
    // Gomb létrehozása a jelentkezések megtekintéséhez
    const listbtn = document.createElement('button');
    // Stílusosztály hozzáadása a gombhoz
    listbtn.classList.add('jobcard-btn');
    // Gomb szövegének beállítása
    listbtn.textContent = 'Jelentkezések';
    // Munka azonosítójának (task_Id) hozzáadása attribútumként
    listbtn.setAttribute('jobid', data.task_Id);
    // Eseménykezelő hozzáadása a gombhoz a jelentkezések panel megnyitására
    listbtn.addEventListener('click', function(event) {
        // A kiválasztott munka azonosítójának lekérése
        let selectedjob = event.currentTarget.getAttribute('jobid');
        // Az alkalmazás panel beállítása "received" módra (a felhasználó által kapott jelentkezések)
        document.getElementById('application-panel').setAttribute('data-action', 'received');
        // A kiválasztott munka azonosítójának beállítása a panelhez
        document.getElementById('application-panel').setAttribute('jobid', selectedjob);
        // Jelentkezési panel megnyitása
        applicationsOpen(true);
    });
    // Ha vannak új jelentkezések (ujjelentkezes = true), figyelmeztető jelzés hozzáadása
}

```

```

    if (data.ujjelentkezés) {
        // Figyelmeztető elem létrehozása
        const listalert = document.createElement("div");

        // Stílusosztály hozzáadása a figyelmeztetéshez
        listalert.classList.add('alert');

        // Figyelmeztető szöveg beállítása
        listalert.textContent = "!";

        // Figyelmeztető elem hozzáadása a gombhoz
        listbtn.appendChild(listalert);
    }

    // Gomb hozzáadása a kártya fejlécéhez
    header.appendChild(listbtn);
}

```

Ez a kódrészlet felelős azért, hogy szerkesztési módban (amikor az **edit** paraméter *true*), a munkaaajánlat kártyához egy **"Jelentkezések"** gomb kerüljön, amely lehetővé teszi a felhasználó számára, hogy megtekintse az adott munkára érkezett jelentkezéseket. A gombhoz egy eseménykezelő is társul, amely megnyitja a jelentkezési panelt, és beállítja a megfelelő attribútumokat (pl. **data-action** és **jobid**). Ha új jelentkezések érkeztek (**ujjelentkezés igaz**), egy figyelmeztető jelzés ("!") is megjelenik a gombon. Ez a funkcionalitás kifejezetten a szerkesztési opciók részeként, a jelentkezések kezelésére szolgál.

jobsload(): Munkák betöltése és szűrése keresési feltételek alapján.

Példa a használatra:

```

async function jobsload() {
    // Munkák lekérdezése az API-ról a felhasználó azonosítója alapján
    let jobs = await fetchData("munkakmentette/" + user.id);

    // Kiválasztott helyszínek és kategóriák lekérése a legördülő menükből
    const selectedCheckboxloc = document.querySelector-
    All("#dropdownlocation input:checked");
    const selectedCheckboxgro = document.querySelector-
    All("#dropdownngroup input:checked");

    // Kiválasztott helyszínek és kategóriák értékének kinyerése tömbökként
    const selectedLocations = Array.from(selec-
    tedCheckboxloc).map(checkbox => checkbox.value);
    const selectedGroups = Array.from(selec-
    tedCheckboxgro).map(checkbox => checkbox.value);

    // Keresőmező tartalmának lekérése, kisbetűssé alakítva és whitespace-ek eltávolítása
    const search = document.getElementById("search").va-
    lue.trim().toLowerCase();

    // Törlés gomb megjelenítése, ha van keresési feltétel (szöveg, helyszín vagy kategória

```

```

        showclearbtn((search !== "" || selectedLocations.length !== 0 || selectedGroups.length !== 0));

        // Kiválasztott kategóriák és munkák naplózása hibakeresés céljából
        console.log(selectedGroups);
        console.log(jobs);

        // Munkák szűrése a megadott feltételek alapján
        const szurtMunkak = await jobs.filter((munka) => {
            // Helyszín szűrése: igaz, ha nincs helyszín kiválasztva, vagy a munka helyszíne szerepel a kiválasztottak között
            const helyszinTalalat =
                selectedLocations.length === 0 || selectedLocations.includes(munka.helyszin);

            // Kategória szűrése: igaz, ha nincs kategória kiválasztva, vagy a munka kategóriái tartalmazzák a kiválasztottakat
            const categoriaTalalat =
                selectedGroups.length === 0 ||
                (munka.feladatKategoriak &&
                    selectedGroups.some((group) => munka.feladatKategoriak.includes(group)));

            // Keresőszó szűrése: igaz, ha nincs keresőszó, vagy a munka címe/leírása tartalmazza a keresőszót
            const keresesTalalat =
                search === "" ||
                (munka.cim && munka.cim.toLowerCase().includes(search)) ||
                (munka.leiras && munka.leiras.toLowerCase().includes(search));

            // Összes feltétel együttes teljesülése esetén a munka megfelel a szűrőnek
            return helyszinTalalat && categoriaTalalat && keresesTalalat;
        });

        // Találatok számának megjelenítése a felületen
        document.getElementById("job-count").innerText = `${szurtMunkak.length} találat`;

        // Szűrt munkák betöltése és megjelenítése
        loadjob(szurtMunkak);
    }

```

jobviewopen(event): Munka részleteinek megtekintése vagy új munka létrehozása szerkeszthető mezőkkel.

A **jobviewopen** függvény két fő üzemmódban működik:

Szerkeszthető mód (ha a felhasználói panel látható vagy új munkát hoz létre): Ebben az esetben a függvény szerkeszthető input mezőkkel tölti fel a munka részleteit (cím, leírás, helyszín, fizetés, stb.), és lehetővé teszi új kategóriák hozzáadását, a munka adatainak mentését (**jobactionbutton**) vagy lezárását (**jobdelete**). Új munka esetén üres alapértelmezett adatokkal indul.

Megtekintési mód (ha nem szerkeszthető): A munka részletei csak olvasható formában jelennek meg, és egy jelentkezési gomb áll rendelkezésre. A hirdető nevére kattintva a profilja megtekinthető.

A függvény kezeli a kategóriák dinamikus hozzáadását és törlését szerkeszthető módban, valamint biztosítja, hogy a határidő formátuma megfelelő legyen. A szerkesztési lehetőségek csak akkor érhetők el, ha a munka nem "lezárt" státuszú.

profileviewopen(uid): Felhasználói profil megjelenítése adatokkal, munkákkal és értékelésekkel.

A függvény egy adott felhasználó profiljának megnyitásáért és adatainak megjelenítéséért felelős. Lekéri a felhasználó adatait (név, születési dátum, regisztrációs dátum, bio, profilkép, telefonszám, munkák, értékelések) az API-ról, majd dinamikusan feltölti ezeket a profil kártyára. Az értékelések átlagát kiszámítja, csillagos formátumban jeleníti meg, a felhasználó munkáit kártyákként lemodellezi, az értékeléseket pedig külön-külön elemekként listázza. Végül a **profileview(true)** függvényhívással láthatóvá teszi a profil nézetet.

loadApplications(): Jelentkezések betöltése és kezelése.

A függvény a jelentkezések betöltéséért és megjelenítéséért felelős az aktuális szekció („*függőben*”, „*elfogadva*”, „*elutasítva*”) alapján, attól függően, hogy a felhasználó a saját jelentkezéseit („*sent*”) vagy a munkájára érkezett jelentkezéseket („*received*”) szeretné megtekinteni. „*Received*” módban a megadott munka (**jobid**) jelentkezéseit kéri le az API-ról (**munkarajelentkezések/\${jobid}/\${section}**), míg „*sent*” módban a felhasználó saját jelentkezéseit tölti be (**jelentkezések/\${user.id}**), majd szűri a szekció szerint. Dinamikusan létrehozza a jelentkezési kártyákat (**application-card**), amelyek „*received*” esetén a jelentkező teljes nevét, státuszát, jelentkezési dátumát, telefonszámát és egy „*Profil megtekintése*” gombot, „*Sent*” esetén pedig a munka címét, helyszínét, jelentkezési dátumát és egy „*Munka megtekintése*” gombot tartalmaznak. A kártyák összecuszkható részletekkel rendelkeznek, amelyeket egy nyíl (**application-arrow**) segítségével lehet megjeleníteni vagy elrejteni. „*Received*” és „*függőben*” szekció esetén elfogadás (**acceptApplication**) és elutasítás (**rejectApplication**) gombok jelennek meg. Ha van nem látott jelentkezés (**latta_e === 0**), egy figyelmeztető jelzés (**alert**) látható a kártyán. „*Received*” módban a **jobdepeding** elem, „*sent*” módban pedig a **jobaccept** vagy **jobreject** elemek jelennek meg, ha van nem látott jelentkezés. A nyílra kattintva a részletek megjelennek/eltűnnek (**toggleDetails**), a gombok pedig a megfelelő profil- vagy munka-megtekintési funkciókat indítják el.

API-hívások

Munkák:

példák:

```
/munkakmentette/{userId}
```

```
/sajاتمunkak/{userId}
/feladat/update/{jobId}
/feladat/statusupdate/{jobId}/lezart
```

Kategóriák:

példák:

```
/kategoria/
/kategoria/ujkategoriahozzaadas/{jobId}/{katname}
/kategoria/kapcsolattorles/{jobId}/{katId}
```

Jelentkezések:

példák:

```
/munkarajelentkezesek/{jobId}/{section}
/jelentkezes/action/{userId}/{jobId}
```

Felhasználók:

példák:

```
/felhasznalo/getuser/{uid}/{currentUserId}
/felhasznaloertekeles
```

Képek:

Profilképek feltöltése a https://jannn1.hu/profile_image_upload.php címre.

Eseménykezelés

DOMContentLoaded: Automatikus bejelentkezés inicializálása.

```
document.addEventListener('DOMContentLoaded', () => {});
```

click/input: Navigációs gombok, szűrők, keresőmezők és űrlapok kezelése.

```
document.getElementById("navico").addEventListener("click",
function() {})
```

load: Kategóriák és helyszínek betöltése legördülő menükbe.

```
window.addEventListener("load", function() {})
```

Megjegyzések

- A kód JSON formátumban adja vissza az adatokat.
- A profilképek feltöltése külső PHP végpontra történik.
- A csillagos értékelési rendszer dinamikus, 1-5 közötti pontszámokkal.

login.js

Feladata

A fájl három fő függvényt tartalmaz: egy panelt forgató **segédfüggvényt**, valamint a bejelentkezés és regisztráció kezelésére szolgáló **aszinkron** függvényeket. A kód felelős a felhasználói adatok begyűjtéséért, validálásáért, API-hívásokkal történő adatküldésért, valamint a válaszok kezelésére és a felhasználói munkamenet fenntartására.

Főbb funkciók

Panel forgatás:

- A bejelentkezési és regisztrációs panelek közötti vizuális váltás támogatása.

Bejelentkezés:

- Felhasználói adatok (e-mail, jelszó) begyűjtése, validálása és küldése a backend felé.
- Munkamenet és tartós bejelentkezés kezelése.

Regisztráció:

- Új felhasználó regisztrálása e-mail és jelszó megadásával, jelszó-egyezés ellenőrzésével.
- Sikeres regisztráció esetén munkamenet és opcionális tartós bejelentkezés.

Főbb függvények

togglePanel(): A bejelentkezési és regisztrációs panelek közötti vizuális váltást kezeli.

```
function togglePanel() {  
    const container = document.getElementById('panel-container');  
    container.classList.toggle('rotated');  
}
```

A függvény a panel-container azonosítójú elemhez hozzáadja vagy eltávolítja a rotated osztályt, ezzel váltva a bejelentkezési és regisztrációs panelek között. A felhasználói felületen a panelek közötti váltás vizuális effektusának biztosítására szolgál.

login(): Aszinkron függvény a bejelentkezés kezelésére, API-hívással és hibakezeléssel.

```
async function login() {  
    // Hibák resetelése  
    document.querySelector('#loginerror').style.display = 'none';  
    document.querySelector('#loginallerror').style.display = 'none';  
    // Adatok begyűjtése  
    const email = document.querySelector('#logmail').value;  
    const password = document.querySelector('#logpw').value;  
    const rememberMe = document.querySelector('#logremember').checked;  
    // Hibák ellenőrzése  
    if (!email || !password) {  
        document.querySelector('#loginallerror').style.display = 'block';  
        return;  
    }
```

```

    }
    // Adatok küldése a backendnek
    var resp = await fetch('/api/login', {
      method: 'POST',
      headers: {
        'Content-Type': 'application/json',
      },
      body: JSON.stringify({ email, password }),
    });
    // Backend válaszána elemzése
    if (resp.ok) {
      // Felhasználó elmentése, ha a "bejelentkezve marad" opció be van jelölve
      if (rememberMe) {
        localStorage.setItem('jobsearchuser',
JSON.stringify({email, password}));
      }
      // Felhasználó elmentése a munkamenet idejére
      sessionStorage.setItem('jobsearchuser',
JSON.stringify({email, password}));
      window.location.href = './index.html';
    } else {
      document.querySelector('#loginerror').style.display =
'block';
    }
  }
}

```

Begyűjti az e-mailt, jelszót és a *"bejelentkezve marad"* opciót, ellenőrzi az adatok meglétét, majd POST kéréssel elküldi azokat az `/api/login` végpontnak. Sikeres válasz esetén a felhasználói adatokat elmenti, és átirányít az **index.html** oldalra.

Hibakezelés:

- Ha az e-mail vagy jelszó hiányzik, a **#loginallerror** hibaüzenet jelenik meg.
- Ha a backend válasza nem sikeres, a **#loginerror** hibaüzenet jelenik meg.

Munkamenet kezelés:

- **localStorage**: Tartós bejelentkezéshez, ha **rememberMe** igaz.
- **sessionStorage**: Munkamenet idejére történő tárolás minden bejelentkezésnél.

register(): Aszinkron függvény a regisztráció kezelésére, jelszó-egyeztetés ellenőrzésével és API-hívással. Begyűjti az e-mailt, jelszót, jelszó megerősítést és a *"bejelentkezve marad"* opciót, ellenőrzi az adatok meglétét és a jelszavak egyezését, majd POST kéréssel elküldi az adatokat az `/api/register` végpontnak. Sikeres válasz esetén értesíti a felhasználót, elmenti az adatokat, és átirányít az index.html oldalra.

Hibakezelés:

- Ha az e-mail, jelszó vagy jelszó megerősítés hiányzik, a **#regallerror** hibaüzenet jelenik meg.
- Ha a jelszó és a megerősítés nem egyezik, a **#regpwerror** hibaüzenet jelenik meg.
- Ha a backend válasza nem sikeres, a **#regerror** hibaüzenet jelenik meg.

Munkamenet kezelés:

- **localStorage**: Tartós bejelentkezéshez, ha rememberMe igaz.
- **sessionStorage**: Munkamenet idejére történő tárolás minden regisztrációnál.

Vizuális váltás: A **togglePanel()** függvény forgatja a **panel-container** elemet a panelek közötti váltáshoz.

API-hívások

Bejelentkezés

```
POST /api/login - JSON payload: { email, password }
```

Regisztráció

```
POST /api/register - JSON payload: { email, password }
```

Megjegyzések

- A kód feltételezi egy backend API létezését az **/api/login** és **/api/register** végpontokkal.
- A jelszavak sima szöveggént kerülnek tárolásra a **localStorage** és **sessionStorage** tárolókban, ami biztonsági kockázatot jelent.
- A regisztráció során a jelszó megerősítése kliensoldalon történik.

Routing/ Navigáció

Oldalak közötti navigáció

Az alkalmazás kettő különálló HTML oldalt tartalmaz, amelyek közötti mozgás szerveroldali betöltéssel történik:

Login.html

Bejelentkezés és regisztráció kezelése.

Navigáció innen: Sikeres bejelentkezés vagy regisztráció után a **Login.js** átirányít az **index.html** oldalra:

```
window.location.href = './index.html';
```

A bejelentkezési és regisztrációs panelek közötti váltás kliensoldali, a **togglePanel()** függvénnyel.

Navigáció ide: Az **index.html** navigációs menüjének **"Kijelentkezés"** (**#logout**) opciója ide irányít vissza.

index.html

Az alkalmazás főoldala, amely a kezdőlapot, munkák keresését és különböző paneleket tartalmaz.

Navigáció innen: A felhasználói profil megtekintésekor a **profileviewopen()** függvény meghívhatja a **profil.html** oldalt (pl. **#navuser** vagy **#myprofile** opciók).

Kijelentkezés (**#logout**) a **Login.html** oldalra irányít.

Navigáció ide: A **Login.html** sikeres bejelentkezés/regisztráció után ide irányít.

Oldalon belüli navigáció (index.html)

Az **index.html** oldalon belüli navigáció kliensoldali, és a következő elemek biztosítják:

Navigációs menü (#nav)

A bal oldali navigációs sáv ikonokkal és szövegekkel biztosít gyors hozzáférést az alkalmazás fő szekcióihoz:

Főoldal (#home)

Megjeleníti a **#homepage** szekciót (kezdőlap, hero szekció, friss munkák).

Esemény: A **js/index.js** elrejt a többi szekciót (pl. **#container**) és megjeleníti a **#homepage** elemet show osztállyal.

Munkák (#jobs)

Megjeleníti a **#container** szekciót, amely a munkák keresési felületét és listáját tartalmazza.

Esemény: A **jobsLoad()** függvény betölti a munkákat és szűrőket.

Mentett munkák (#savedjobs)

A felhasználó által mentett munkákat mutatja, a **#container** szekcióban szűrve.

Esemény: A **jobsLoad()** függvény egy specifikus szűrővel (pl. mentett munkák API-hívása).

Új munka létrehozása (#newjob)

Megnyitja a **#jobfullview** panelt szerkeszthető módban új munka létrehozásához.

Esemény: A **jobviewopen(null)** hívása üres munkaadatokat tölt be.

Felhasználói profil (#navuser):

Megnyitja a **#navusermanage** almenüt, amely további opciókat kínál.

Esemény: A **toggle** funkció a **#navusermanage** megjelenítésére/elrejtésére.

Felhasználói kezelőpanel (#navusermanage)

Ez egy almenü, amely a **#navuser** elemre kattintva jelenik meg:

Munkáim (#myjobs)

A felhasználó által meghirdetett munkákat mutatja a **#container** szekcióban szűrve.

Értesítési jelző: **#myjobsalert** (pl. új jelentkezések esetén).

Jelentkezéseim (#jobapplications)

Megnyitja az **#application-panel** panelt, amely a felhasználó által küldött vagy kapott jelentkezéseket listázza.

Esemény: A **loadApplications(section)** függvény betölti a jelentkezéseket.

Értesítési jelző: **#jobvalami**

Adatok (#userdata)

Megnyitja a **#userpanel** panelt a felhasználói adatok szerkesztéséhez (vezeték-név, jelszó stb.).

Értesítési jelző: **#userdataalert**

Profilom (#myprofile)

Megnyitja a saját profil nézetet

Esemény: A **profileviewopen(user.id)** függvény.

Kijelentkezés (#logout)

Visszaállítja a munkamenetet és átirányít a **login.html** oldalra.

Esemény: sessionStorage/localStorage törlése és

```
window.location.href = './login.html'
```

Dinamikus panelek

Az oldalon belüli mozgás jelentős része panelek megjelenítésével és elrejtésével történik, amelyeket a JavaScript vezérel:

Munka részletei (#jobfullview)

Megjelenik egy munka kártyára kattintva (pl. **jobcard-btn**).

Esemény: A **jobviewopen(event)** függvény betölti a munka adatait.

Bezárás: A **#jobviewx** gomb elrejt a panelt (**showjobfullview(false)**).

Felhasználói adatok szerkesztése (#userpanel)

Megjelenik a **#userdata** menüpontból vagy más szerkesztési műveletből.

Esemény: A **#userpanel** tartalmát dinamikusan tölti be az **index.js**.

Bezárás: A **#panelx** gomb elrejt a panelt.

Jelentkezések panel (#application-panel):

Megjelenik a **#jobapplications** menüpontból vagy egy munka jelentkezéseinek megtekintésekor.

Esemény: A **loadApplications(section)** függvény betölti a jelentkezéseket.

Bezárás: A **#applicationclose** gomb elrejt a panelt.

Profil nézet (#profileview):

Megjelenik egy felhasználó profiljának megtekintésekor (pl. hirdető neve, **#myprofile**).

Esemény: A **profileviewopen(uid)** függvény tölti be az adatokat.

Bezárás: A **#profileviewx** gomb elrejt a panelt.

Alternatív profil nézet (#profileview2):

- Egyszerűbb profilnézet
- Bezárás: A **#profileviewx2** gomb

Kezdőlap navigáció

A **#homepage** szekcióban a következő gombok biztosítanak navigációt:

Munkák megtekintése (#mainjobs)

Átvált a **#container** szekcióra, és betölti az összes munkát.

Esemény: Hasonló a **#jobs** menüponthoz, **jobsload()** hívás.

Munka meghirdetése (#mainnewjob)

Megnyitja a **#jobfullview** panelt új munka létrehozására.

Esemény: **jobviewopen(null)**.

Összes munka megtekintése (#viewalljobs)

Átvált a **#container** szekcióra, hasonlóan a **#mainjobs** gombhoz.

Oldalon belüli navigáció (login.html)

A **login.html** oldalon a navigáció egyszerűbb, és a bejelentkezési/regisztrációs panelek közötti váltásra korlátozódik:

Panel forgatás

A **togglePanel()** függvény vált a **#login-panel** és **#register-panel** között:

```
function togglePanel() {  
    const container = document.getElementById('panel-container');  
    container.classList.toggle('rotated');  
}
```

Esemény: A **"Regisztráció"** vagy **"Bejelentkezés"** linkekre kattintva

```
<span class="span" onclick="togglePanel()">
```

Sikeres bejelentkezés/regisztráció

Átirányít az **index.html** oldalra.

Űrlapkezelés

Az alkalmazás két űrlapot kezel:

Bejelentkezés és regisztráció (login.html): Autentikáció és fiók létrehozása.

Felhasználói adatok szerkesztése (index.html): Profiladatok és profilkép módosítása, értékelések hozzáadása.

A validáció a **.NET** backendben történik **Data Annotations** használatával, **JWT** autentikációval, míg a kliensoldal dinamikusan kezeli a hibákat.

Bejelentkezési űrlap (Login.html)

Kliensoldal: Ellenőrzi, hogy az email és jelszó nem üres; hibák a backend válaszok alapján.

Backend: Kötelező email (érvényes formátum), jelszó (min. 8 karakter); hibás autentikáció esetén részletes hibaüzenet; sikeres esetben JWT token.

Megjegyzések: Biztonságos token-alapú tárolás, pontos hibakijelzés.

Regisztrációs űrlap (*login.html*)

Kliensoldal: Ellenőrzi a mezők kitöltöttségét; backend hibák megjelenítése.

Backend: Kötelező email (érvényes, egyedi), jelszó (min. 8 karakter, kis-/nagybetű, szám), jelszó megerősítés (egyezés); sikeres regisztráció esetén JWT token.

Megjegyzések: Szigorú jelszóvalidáció, létező email ellenőrzés.

Felhasználói adatok szerkesztése (*index.html*)

Kliensoldal: Profilkép előnézet (kép típus, max. 5 MB); mező-specifikus hibák a backendtől.

Backend: Kötelező vezetéknev, keresztnév, születési dátum; opcionális jelszó, bio, profilkép (kép, max. 5 MB); JWT autentikáció.

Megjegyzések: Teljes űrlapkezelés, jogosultság-ellenőrzés.

Értékelés hozzáadása (*index.html*)

Kliensoldal: Csillagértékelés (1-5), megjegyzés ellenőrzése; backend hibák megjelenítése.

Backend: Kötelező pontszám (1-5), értékelő/értékelt id, megjegyzés, dátum; JWT autentikáció.

Hitelesítés

A backend JWT (*JSON Web Token*) alapú hitelesítést használ, amelyet a kliens a *localStorage/sessionStorage*-ban tárol (*jwtToken* kulcs alatt).

Kliensoldal

Bejelentkezés/regisztráció során a sikeres API-válasz JWT tokent ad, amit a kliens tárol.

Az *index.html* és *profil.html* űrlapok kérései az **Authorization: Bearer <token>** fejléccet használják.

Backendoldal

Az **[Authorize]** attribútum védi az API végpontokat (pl. */api/user*, */api/felhasznaloertekeles*).

A token validálása ellenőrzi a felhasználó azonosítóját (*ClaimTypes.NameIdentifier*)

A hitelesítés biztosítja, hogy csak jogosult felhasználók módosíthatnak adatokat vagy adhatnak értékelést, növelve a biztonságot.

PHP

Leírás

A weboldalhoz kapcsolódó képfeltöltés és képméret-optimalizálás funkció PHP-ben készült.

A kód fő funkciói

- Képfeltöltés támogatása (*POST* kéréssel)
- Engedélyezett formátumok: *.jpg*, *.jpeg*, *.png*, *.gif*
- Fájlok mentése *biztonságos*, *egyedi* fájlnevvvel
- Automatikus átméretezés max. *300x300 px*-re
- Visszajelzés *JSON* formátumban

Folyamat leírása

1. Fájl feltöltése

A felhasználó a kliensoldalon keresztül képet küld a szerverre, amelyet a script a `$_FILES['image']` változón keresztül fogad.

2. Fájlnev generálása

Egyedi fájlnev készül a fájlütközések elkerülése érdekében:
`uniqid('image_', true) . '.' . $kiterjesztés`

3. Átméretezés

A fájl méretarányos kicsinyítése az alábbi logika szerint történik:

- Ha túl nagy, arányosan csökkenti a méretet
- Ha megfelelő méretű, a méret az eredeti marad

4. Mentés és válasz

Két verzió készül: egy eredeti és egy átméretezett kép. Végül a rendszer visszaküld egy JSON választ az elérési útvonalakkal.

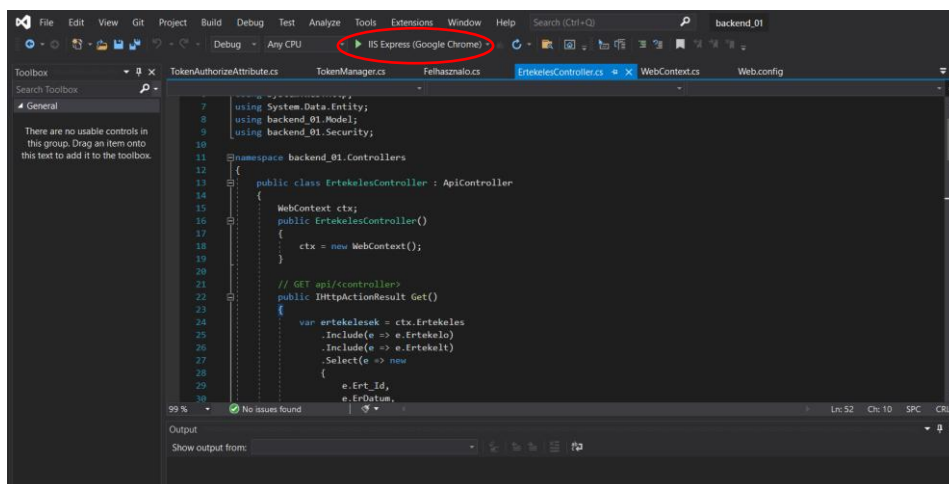
5.2. A program elindítása

Hogyha a **XAMPP**-ot lokálisra írtuk át, akkor fontos, hogy fusson. (ld. [Kapcsolat](#))

Backend elindítása:

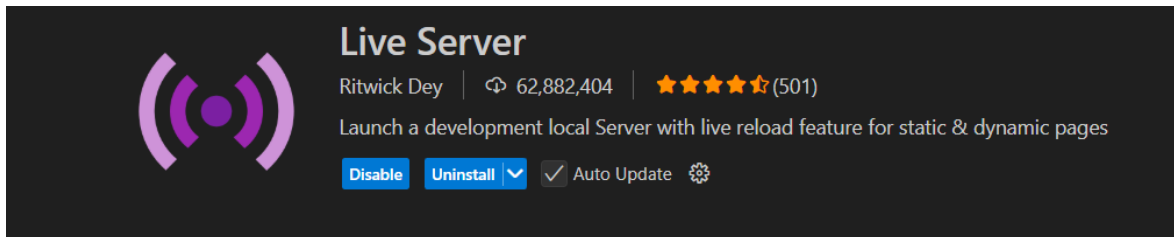
A program elindításához szükségünk lesz a szerveroldalra, illetve a kliens oldalra is.

Indítsuk el a backendet a **.sln** file megnyitása után. Ezt ezen a gombon tehetjük meg:

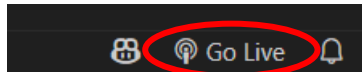


A frontend elindításához szükséges egy bővítmény felrakása:

1. A bal oldali oldalsávon kattints a **Bővítmények** ikonra
2. A felugró keresősávba írd be: **Live Server**.
3. A találatok közül válaszd ki a **Live Server** bővítményt.
4. Kattints a **Live Server** bővítmény melletti **Install** (Telepítés) gombra.
5. A telepítés néhány másodpercet vesz igénybe, és a folyamat végén a gomb **Uninstall** (Eltávolítás) gombra vált.



Ahhoz, hogy a weboldalt lássuk, a jobb alsó sarokban lévő „Go Live”-ra kattintsunk rá, miután betöltöttük a fájlok mappáját.



Az oldalra bejelentkezéshez szükséges adatok:

email cím: felhasználó1@test.com

jelszó: jelszo

De természetesen működik regisztrációval is.

5.3. A program használata, fő elemei

Webalkalmazásunk több különálló, jól elkülöníthető nézetből áll, amelyek a felhasználói szerepkörök és funkciók szerint épülnek fel. A használata néhány lépésben elsajátítható.

Bejelentkezés/Regisztráció

A kezdőlapen lehetőség van új felhasználói fiók létrehozására. A regisztráció során a felhasználónak meg kell adnia:

- e-mail címét,
- jelszavát.

Sikeres regisztráció után a rendszer automatikusan belépteti a felhasználót.

A korábban regisztrált felhasználók a bejelentkezési oldalon adhatják meg adataikat. A bejelentkezés után hozzáférnek a rendszer teljes funkcionalitásához.

Az Bejelentkezés/Regisztráció képernyőképe a **3.számú mellékletben** – *Bejelentkezés* ([5. ábra](#)) / *Regisztráció* ([6. ábra](#)) - található.

Munkahirdetés feladása

A regisztrált felhasználók – függetlenül attól, hogy milyen joguk van - új hirdetést adhatnak fel, ahol megadhatják:

- a munka nevét,
- részletes leírását,
- helyszínt,
- pontos címet,
- határidőt
- várható fizetést,
- elérhetőségeket.

A hirdetés feladás képernyőképe a **3.számú mellékletben** - *Munkahirdetés feladása* ([7.ábra](#)) - található.

Állások böngészése

A felhasználó egy gomb megnyomásával tud csak keresgélni a munkák között. Ezen a felületen tud szűrni is a lehetőség között is, a keresés funkció segítségével. Képernyőképe a **3. számú mellékletben** – *Álláskeresés* ([8.ábra](#)) – található.

Jelentkezés munkára

Ha a felhasználó talál egy számára szimpatikus álláshirdetést, egy gombnyomással jelentkezhet rá. A munkaadó ezután visszajelzést küldhet (elfogadja vagy elutasítja a jelentkezést). Képernyőképe a **4.számú mellékletben** - *Jelentkezés munkára* ([9.ábra](#)) - található.

Értékelés és visszajelzés

A munka elvégzése után a felek értékelhetik egymást, csillagokkal és akár hozzászólásokkal is.

Képernyőképe a **4.számú mellékletben** – *Értékelés leadása* ([10.ábra](#)) - található.

Profilkezelés

Minden felhasználó elérheti a saját profilját, ahol módosíthatja adatait, megnézheti az eddigi jelentkezéseit, illetve korábban feladott hirdetéseit. Képernyőképe a **4.számú mellékletben** – *Profiloldal* ([11.ábra](#)) - található.

6. A program tesztelése

6.1. Unit tesztek

A rendszer minőségének biztosítása érdekében minden fontos backend komponenshez különálló **Unit tesztek** készültek. Ezek célja, hogy az API végpontokat, adatkezelési logikákat és hitelesítési folyamatokat automatikusan, megbízhatóan teszteljük. A tesztek a **Microsoft.VisualStudio.TestTools.UnitTesting** keretrendszert használják, és memória-alapú WebContext környezetben futnak, így nem befolyásolják az éles adatokat. A tesztek célja az volt, hogy a backend és a frontend komponensek legfontosabb részeinek működését automatikusan validáljuk.

Tesztelési eredmények összefoglalása

A tesztelések során néhány esetben korlátozott számú sikeres (zöld) teszt eredményt kaptunk. Ennek oka elsősorban az volt, hogy:

- A projekt fejlesztése során elsődleges szempont az éles működés biztosítása volt.
- Több összetettebb funkció (pl. adatbázis-interakciók, autentikáció, álláshirdetések kezelése) olyan mértékben dinamikus, hogy teljes automatizált teszteléssel nem minden részlet volt lefedhető a projekt ütemezése mellett.
- Egyes funkciók manuális teszteléssel gyorsabban és hatékonyabban voltak validálhatóak.

Fontos azonban kiemelni:

A fejlesztés során minden fő funkció manuálisan lett ellenőrizve.

A rendszer éles működés közben hibamentesen működik.

A felhasználók által végzett alapvető műveletek (regisztráció, bejelentkezés, hirdetés feladása, jelentkezés állásra, értékelés) sikeresen végrehajthatók.

Az alábbi tesztкод egy feladat hozzáadását teszteli:

```
[TestInitialize]
    public void Setup()
    {
        context = new WebContext(); // Az adatbázis kapcsolata
        controller = new FeladatController(); // A controller példányosítása
        // Tesztadatok hozzáadása, ha még nem léteznek
        if (!context.Felhasznalo.Any())
        {
            context.Felhasznalo.Add(new Felhasznalo {
                User_Id = 1, Email = "tesztuser@example.com", Jelszo =
                "Teszt1234", SzulDat = "1990-05-10", VezNev = "Teszt", KerNev =
                "TTeszt", ProfilKep = null, Bio = "asdasdtestasd", RegDatum =
                "2000-01-01", Felhtipus = "user" });
        }
    }
```



```

    }

    if (!context.Feladat.Any())
    {
        context.Feladat.Add(new Feladat { Task_Id = 1,
Cim = "Teszt Feladat", Statusz = "nyitott", Helyszin = "Büfé",
PosztDatum = "2023-04-26", Hatarido = "2023-04-27", User_Id = 1
});

    }

    context.SaveChanges(); // Az adatok mentése a lis-
tában
}

```

A következő kód pedig egy token generálását teszteli:

```

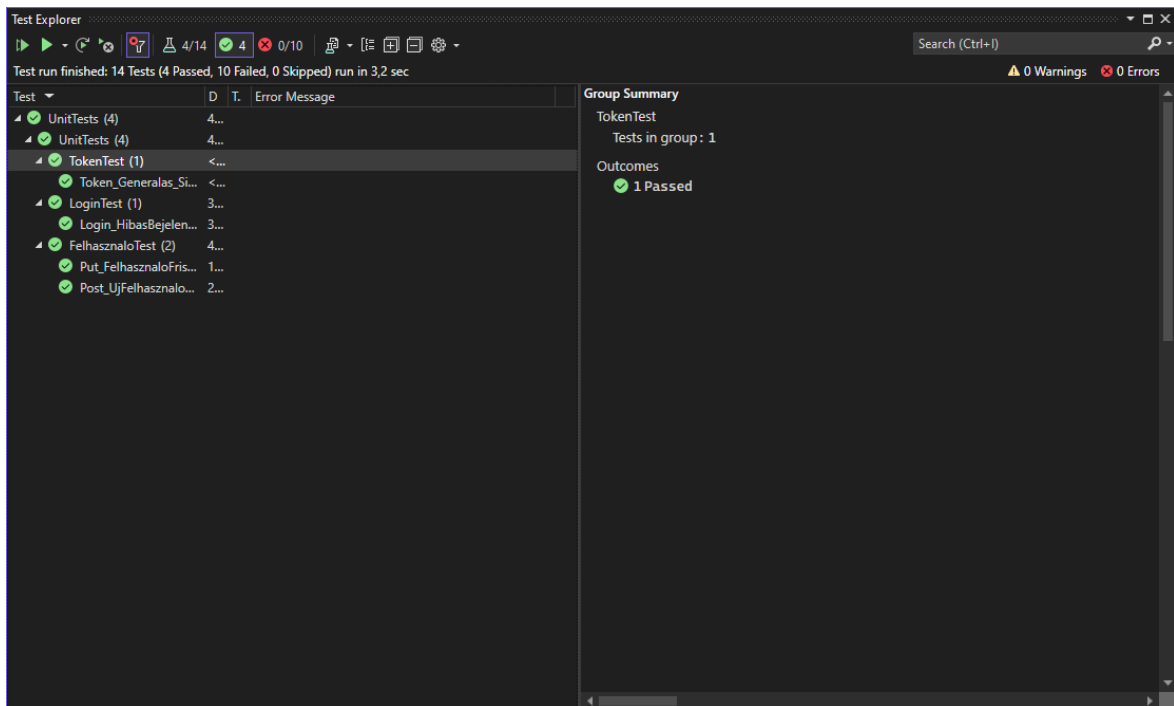
[TestMethod]
public void Token_Generalas_Sikeres()
{
    // Létrehozunk egy tesztfelhasználót
    var user = new Felhasznalo
    {
        Email = "tesztuser@example.com",
        Jelszo = "Teszt1234"
    };

    // Ellenőrizzük, hogy létezik a tesztfelhasználó
    var existingUser = context.Felhasznalo.FirstOrDefault(f => f.Email == user.Email);
    if (existingUser == null)
    {
        context.Felhasznalo.Add(user);
        context.SaveChanges();
    }

    // Token generálása
    var token = TokenManager.GenerateToken(user);
    // Ellenőrizzük, hogy a token nem null és van ér-
téke
    Assert.IsNotNull(token, "A generált token nem lehet
null.");
    Assert.IsTrue(token.Length > 0, "A token nem tar-
talmaz adatokat.");
}

```

Bár az automatikus unit tesztek száma korlátozott maradt, a projekt fő célját – egy működőképes, stabil, használható webalkalmazás létrehozását – sikeresen elértük. A rendszer megbízhatóan üzemel, és minden főbb funkció éles környezetben is hibamentesen teljesít.



7. Összefoglaló és továbbfejlesztési lehetőségek

7.1. Összefoglalás

A projekt során, egy éven keresztül dolgoztunk együtt egy olyan webes alkalmazáson, amely alkalmi munkák hirdetését és keresését segíti elő. A közös munka nemcsak technikai tudásunkat bővítette, hanem rengeteget tanultunk a csapatmunkáról, a feladatok hatékony megosztásáról és az időbeosztás fontosságáról is.

A fejlesztés során lehetőségünk nyílt arra, hogy az eddig elsajátított elméleti ismereteinket gyakorlatban is alkalmazzuk – legyen szó adatbázistervezésről, backend logikáról vagy a felhasználói felület kialakításáról. Közben megtapasztaltuk, hogy hogyan lehet egy ötletből működő rendszert építeni, valamint azt is, hogy milyen kihívásokkal jár az együttműködés. Megértettük, hogy a szoftverfejlesztés nem csak kódolásból áll, hanem folyamatos kommunikációból, rugalmasságból és kreativitásból is.

Büszkék vagyunk arra, hogy egy működő, hasznosnak vélt weboldalt hoztunk létre, amely akár valódi felhasználók számára is értéket teremthet.

7.2. Továbbfejlesztési lehetőségek

Az alkalmazás stabil működése lehetővé teszi a következő szintre lépést, ahol a felhasználói élmény és funkcionalitás bővítése kerül előtérbe. A következő fejlesztések beépítésével a rendszer még teljesebb szolgáltatást nyújthat:

Admin jogosultsággal rendelkező felhasználó szerepe → Bevezetése lehetőséget ad a rendszer karbantartására, a tartalomkezelésre és a felhasználók kezelésére.

Google/Facebook alapú bejelentkezés

A közösségi média alapú hitelesítés lehetővé teszi, hogy a felhasználók gyorsan és biztonságosan jelentkezzenek be. → Könnyebb regisztráció, kevesebb probléma az autentikációval, hitelesítéssel.

Üzenetküldési lehetőség felhasználók között

Egy beépített chatmodul segítségével a felhasználók közvetlenül kommunikálhatnak egymással. → Megnövelt interaktivitás, közösségépítés.

Google Maps integráció

Feladatok, események vagy felhasználók földrajzi elhelyezésének megjelenítése Google Maps API segítségével. → Térképes keresés, helyhez kötött funkciók.

Mobilalkalmazás fejlesztése

A rendszer funkcionalitása mobilra is kiterjeszthető egy natív alkalmazás (pl. React Native vagy MAUI .NET) formájában, ami kényelmesebb hozzáférést biztosít. → Értesítések, offline mód.

Fizetési rendszerek integrációja (PayPal / OTP Simple / Revolut)

Elektronikus fizetések lehetővé tétele prémium funkciók vagy szolgáltatások esetén. → Kifizetések intézése.

Naptárintegráció (Google Calendar, Outlook)

Lehetővé teszi a felhasználók számára, hogy eseményeket automatikusan szinkronizáljanak saját naptárjaikkal. → Emlékeztetők, hatékonyabb időkezelés.

Prémium verzió funkciók

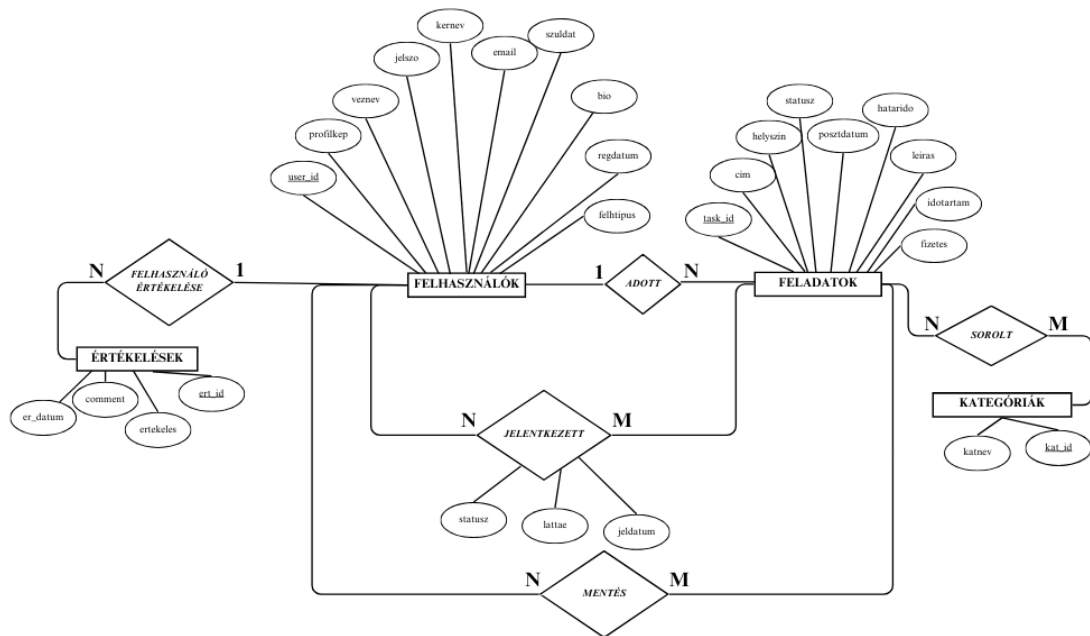
Különböző előfizetési szintek bevezetése extra funkciókkal (pl. kiemelt megjelenés). →
Bevételszerzési lehetőség az működtetőnek és felhasználóbarátabb élmény.

1.számú melléklet – Feladatok felosztása

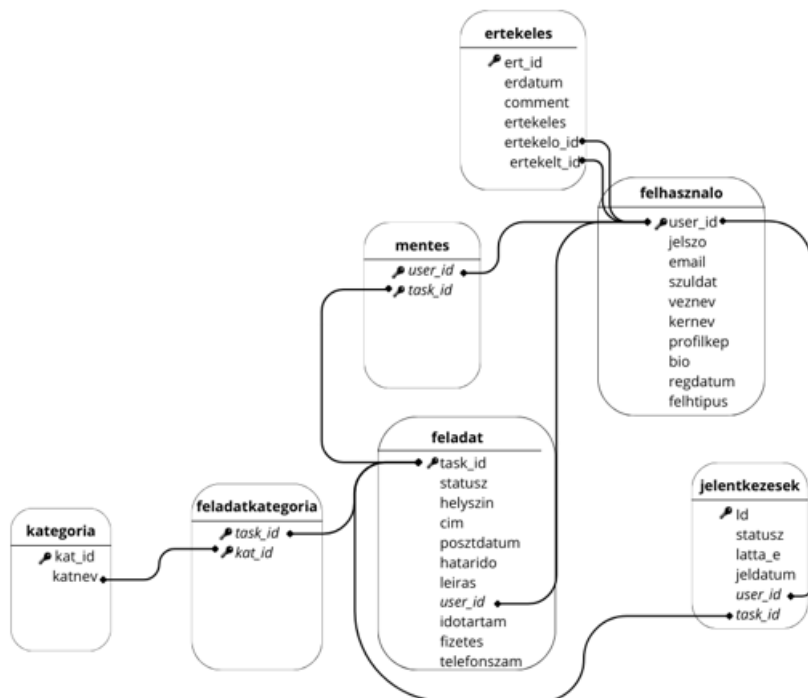
<i>TEVÉKENYSÉG</i>	<i>TERV KÉSZÍTŐJE</i>
Adatbázis ER modell terve	Ár János Dániel, Czall Viktória, Csorba Bálint
Adatbázis relációs terve	Czall Viktória
Adatbázis létrehozása	Csorba Bálint
Adatbázis feltöltése példaadatokkal	Csorba Bálint
Backend modellek felépítése	Czall Viktória
Backend controllerek felépítése	Czall Viktória
Backend titkosítás megvalósítása JWTs token segítségével	Czall Viktória
Backend fejlesztés	Czall Viktória
Frontend dizájnjának és szerkezetének felépítése	Ár János Dániel, Czall Viktória, Csorba Bálint
Frontend alapjainak létrehozása	Ár János Dániel
Frontend fejlesztése, dizájnolása	Ár János Dániel, Csorba Bálint
Frontend összekapcsolása Backenddel	Ár János Dániel
Unit tesztek írása	Csorba Bálint
Rendszertesztek írása	Ár János Dániel
Dokumentáció írása	Ár János Dániel, Czall Viktória, Csorba Bálint
Logótervezés és készítés	Csorba Bálint

1. ábra - Feladatok felosztása

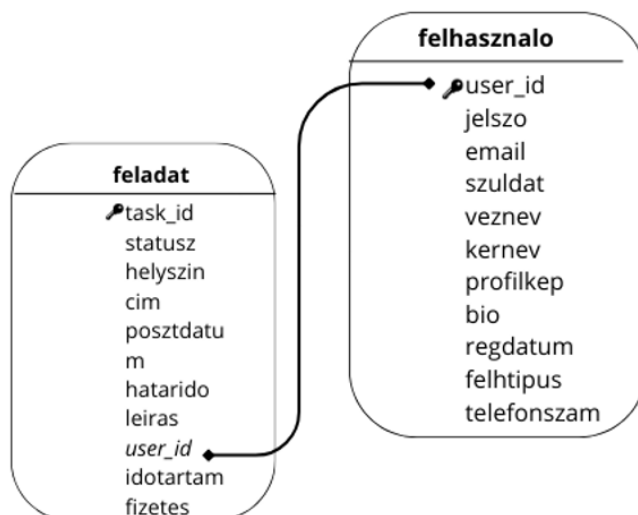
2. számú melléklet – Adatbázis séma



2. ábra - ER – modell

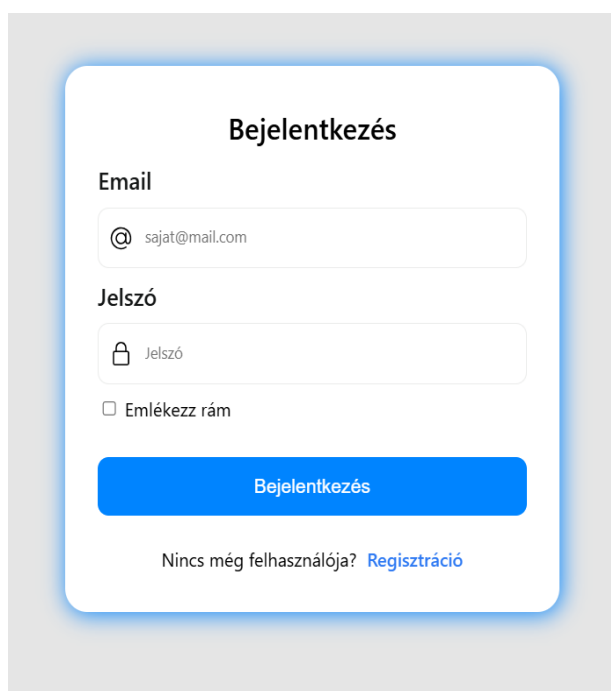


3. ábra - Relációs modell



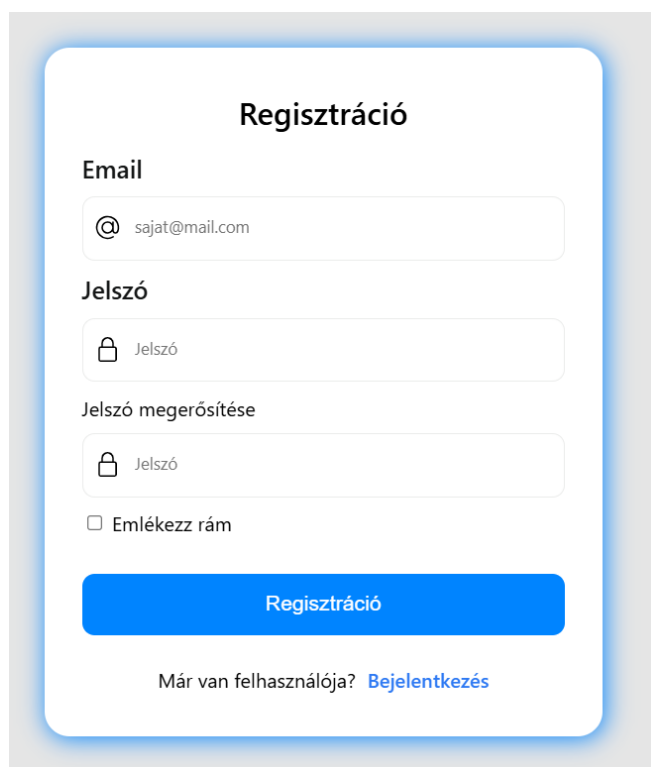
4. ábra – A feladat és a felhasználó kapcsolata

3. számú melléklet – Programhasználati útmutató



The image shows a login form titled "Bejelentkezés". It contains two input fields: "Email" with the placeholder "sajat@mail.com" and "Jelszó" (Password) with a lock icon. Below the password field is a checkbox labeled "Emlékezz rám". A blue button labeled "Bejelentkezés" is positioned below the checkbox. At the bottom, there is a link: "Nincs még felhasználója? [Regisztráció](#)".

5. ábra - Bejelentkezés



The image shows a registration form titled "Regisztráció". It contains three input fields: "Email" with the placeholder "sajat@mail.com", "Jelszó" (Password) with a lock icon, and "Jelszó megerősítése" (Confirm Password) with a lock icon. Below the password fields is a checkbox labeled "Emlékezz rám". A blue button labeled "Regisztráció" is positioned below the checkbox. At the bottom, there is a link: "Már van felhasználója? [Bejelentkezés](#)".

6. ábra – Regisztráció

The form is divided into several sections:

- Név:** A text input field for the name.
- Új kategória hozzáadása:** A text input field with a blue 'Hozzáad' button.
- Munka leírása:** A large text area for the job description.
- Munka részletei:**
 - Meghirdetés dátuma:** 2025-04-25 17:10
 - Helyszín:** A text input field.
 - Fizetés:** A text input field followed by 'Ft'.
 - Időtartam:** A text input field.
 - Határidő:** A date picker showing 'éééé. hh. nn.' and a calendar icon.
 - Hirdető:** A text input field.
- Változtatások mentése:** Two blue buttons: 'Mentés most →' and 'Zárja le most →'.

7. ábra – Munkahirdetés feladása

The interface features a blue header with the title 'Munkák'. Below the header is a search bar with the placeholder text 'Job title, skills, or keywords'. Under the search bar are two dropdown menus: 'Kategória' and 'Helyszínek', both with 'Válassz...' as their selected option. Below these is a text '7 találat'. The main content area displays four job cards, each with a salary, a title, a location, and a 'Több információ' button:

- 8000 Ft**: Kert rendezés, Budapest
- 10000 Ft**: Takarítás albérletben, Debrecen
- 15000 Ft**: Laptop javítás, Pécs
- 4000 Ft**: Fordítás németről, Győr

8. ábra - Álláskeresés

Laptop javítás

Számítástechnika

Nyitva

Munka leírása:

Nem kapcsol be a gép

Munka részletei

Meghirdetés dátuma: 2025. 04. 08. 17:28:10
Helyszín: Pécs
Fizetés: 15000 Ft
Időtartam: 1.5 óra
Határidő: 2025. 04. 20. 0:00:00
Hirdető: Kiss Dani


Jelentkezés a munkára

Jelentkezés most →

Jelentkezés a munkára

Jelentkezett

9. ábra - Jelentkezés munkára



Kiss Dani
Születési dátum: 1994. 11. 02. 0:00:00
Regisztráció dátuma: 2024. 11. 08. 18:22:20
Telefonszám:
★★★★★ 3 (1 értékelés)

ProfilMeghirdetett munkáiÉrtékelések

Írjon értékelést

Értékelés:

★ ★ ★ ★ ★

Írja le gondolatait:

Értékelés jóváhagyása


Értékelések

Lakatos Hajni

Egy kis késés volt, de összességében jó.

★★★★★

10. ábra - Értékelés leadása

Vezetéknév:	Születési dátum:	Profilkép
<input type="text" value="Nagy"/>	<input type="text" value="éééé. hh. nn."/> 	<div>Nincs kép kiválasztva</div>
Keresztnév:	Bemutakozó:	
<input type="text" value="Pista"/>	<div>Írj magadról bemutatkozót</div>	
Új jelszó létrehozása:		
<input type="text" value="Új jelszó"/>		<div>Fájl kiválasztása Nincs fájl kiválasztva</div>
Erősítse meg a jelszavát:		
<input type="text" value="Új jelszó"/>		

11. ábra - Profilkezelés

1.ⁱ Forrásmegjelölés:

2. Logó: Generálva a ChatGPT DALL-E integrációjával (OpenAI, 2023). <https://chatgpt.com/>

3. Adatbázis adatok: Generálva a ChatGPT segítségével (OpenAI, 2023). <https://chatgpt.com/>

4. Ikonok:

Font Awesome ikonok (Font Awesome, 2022). <https://cdn.jsdelivr.net/npm/font-awesome@6.0.0/css/all.min.css>

SVG ikonok (SVG Repo, n.d.). <https://www.svgrepo.com/>

Kulcs <https://www.onlinewebfonts.com/icon/332>