**StudentHive** Dokumentáció

A diákmunka jövője🚀

Tartalomjegyzék

[Bevezetés 3](#_Toc193731244)

[Honnan jött az ötlet? 3](#_Toc193731245)

[A projekt bemutatása 3](#_Toc193731246)

[Az oldal funkciói 3](#_Toc193731247)

[Fejlesztői dokumentáció 4](#_Toc193731248)

[A fejlesztői környezetről 4](#_Toc193731249)

[Miért React a frontendhez? 5](#_Toc193731250)

[Miért ASP.NET Core a backendhez? 6](#_Toc193731251)

[Backend architektúra és API végpontok 7](#_Toc193731252)

[Általános backend felépítés 7](#_Toc193731253)

[API végpontok listája 8](#_Toc193731254)

[Példák API-hívásokra 8](#_Toc193731255)

[Kialakított adatszerkezet bemutatása 9](#_Toc193731256)

[Az adatbázis modell 9](#_Toc193731257)

[Főbb jellemzők 10](#_Toc193731258)

[Kapcsolatok és idegenkulcsok 11](#_Toc193731259)

[Biztonsági szempontok 13](#_Toc193731260)

[Program főbb algoritmusai 13](#_Toc193731261)

[Autentikáció 13](#_Toc193731262)

[Iskolaszövetkezet létrehozása 14](#_Toc193731263)

[Műszak hozzáadása 15](#_Toc193731264)

[Műszak törlése és email értesítés 16](#_Toc193731265)

[Tesztdokumentáció 16](#_Toc193731266)

[Továbbfejlesztési lehetőségek 17](#_Toc193731267)

[Felhasználói dokumentáció 19](#_Toc193731268)

[A program célja és funkciói 19](#_Toc193731269)

[Szükséges hardver- és szoftverkövetelmények 19](#_Toc193731270)

[Telepítési és indítási útmutató 19](#_Toc193731271)

[A programról részletesen 19](#_Toc193731272)

[Hibakezelés és hibaüzenetek magyarázata 19](#_Toc193731273)

[Összefoglalás 19](#_Toc193731274)

[Fejlesztési tapasztalatok 19](#_Toc193731275)

[Hogyan tovább? 19](#_Toc193731276)

[Eredmények értékelése 19](#_Toc193731277)

# Bevezetés

## Honnan jött az ötlet?

Az ötlet éppen munka közben született meg, mellyel saját magunk, illetve rengeteg diáktársunk életét könnyíthetnénk meg a diákmunkaszerzés területén. Az egész koncepciót arra építettük fel, hogy mi diákok minek tudnánk a legnagyobb hasznát venni, illetve mivel lehetne forradalmasítani ezt az egyre növekvő piacot.

## A projekt bemutatása

Egy olyan platformot szerettünk volna létrehozni, amely összeköti az ország valamennyi iskolaszövetkezetét a dolgozni vágyó diákokkal. A tanuló a regisztrációit követően láthatja az összes csatlakozott munkaadó álláshirdetéseit, jelentkezhet azokra és vállalhat műszakokat, mindezt egy helyen. A rendszer különböző megkötéseket tesz a szövetkezetek számára az állások közzétételénél, ezzel is növelve az átláthatóságot. Nem egy egyszerű álláskereső portált akartunk, hanem annál jóval többet. Miután felvételt nyert a diák az adott munkára hozzáfér egy foglalórendszerhez, ahol kedvére válogathat az elérhető műszakok között. Mindezek mellett a diákok értékelhetik a meghirdetett munkákat ezzel is növelve a tökéletes állás megtalálását.

## Az oldal funkciói

Négy szerepkört különböztetünk meg, illetve, ha a felhasználó nincs bejelentkezve, ilyenkor tud böngészni a munkák között, megtekinteni azok értékeléseit. Szerepköreink a következők: diák, közvetítő, iskolaszövetkezet, rendszergazda. A diák tud jelentkezni munkákra, műszakokra, adott esetben (12 órás határon kívül) visszamondani azokat. Ezzel egy átlátható környezet teremtünk egyszerűsítve a beosztások kezelését. A közvetítőt a szövetkezet jelöli egy munkára ezt követően a beérkező állásjelentkezéseket tudja kezelni, hozzáadni új műszakokat, illetve elfogadni, elutasítani a rá érkező jelentkezéseket. Szövetkezetként új munkákat lehet hozzáadni, közvetítőket felvenni, kezelni azokat. Rendszergazda fő feladat a szövetkezetek felvétele, asztali alkalmazásunkban a felhasználók kezelése.

# Fejlesztői dokumentáció

## A fejlesztői környezetről

A fejlesztői környezet több technológiát ötvöz annak érdekében, hogy hatékony, modern és skálázható alkalmazásokat lehessen létrehozni mind webes, mind asztali környezetben.

A **weboldal** fejlesztése **React** és **Vite v6.0.6** alapokon történik, biztosítva a gyors fejlesztési folyamatokat és optimalizált teljesítményt. A frontend több népszerű csomagot használ, például az **ag-grid** táblázatokhoz, az **axios** API-hívásokhoz, a **react-router-dom** az útvonalkezeléshez, és a **react-toastify** értesítésekhez. Az alkalmazás a **TypeScript** segítségével típusbiztos és könnyen karbantartható kódbázist biztosít.

A **backend** ASP.NET alapú, a **.NET 6** verziót használva. Az alkalmazás **JWT-alapú hitelesítést** valósít meg az **AspNetCore.Authentication.JwtBearer** csomaggal, az adatok tárolásához **MySQL** adatbázist használ, amelyhez az **EntityFrameworkCore.Tools** és a **MySql.Data** csomagok biztosítanak ORM támogatást. A dokumentáció generálására a **Swashbuckle.AspNetCore** (Swagger) kerül alkalmazásra.

Az **asztali alkalmazás** fejlesztése **C# és WPF .NET 8** környezetben történik, amely modern és reszponzív felhasználói élményt biztosít. A felület kialakításához **Material Design** alapú csomagok kerülnek felhasználásra, mint például a **MaterialDesignColors**, **MaterialDesignThemes**, és **MaterialDesignThemes.MahApps**. Az adatok kezeléséhez a **Newtonsoft.Json** csomag nyújt támogatást.

A fejlesztői eszközök között a **Visual Studio 2022** és a **Visual Studio Code** biztosítják az optimális munkakörnyezetet. Az alkalmazásstruktúra és a csomagválasztás célja egy jól optimalizált, biztonságos és könnyen bővíthető rendszer kialakítása mind webes, mind asztali környezetben.

A fejlesztési folyamat során **Jira** és **GitHub** eszközök segítik a projektmenedzsmentet és a verziókezelést. A **Jira** feladata a feladatok nyomon követése, sprinttervezés és a fejlesztési folyamat strukturált menedzselése. A backlog kezelésére, a feature fejlesztések nyomon követésére és a hibajavítások priorizálására használjuk, biztosítva ezzel az átlátható és hatékony munkavégzést.

A **GitHub** a verziókezelést biztosítja, lehetővé téve a csapat számára a hatékony együttműködést, a kódváltozások követését és a különböző fejlesztési ágak kezelését.

### Miért React a frontendhez?

#### Komponens alapú architektúra

A React lehetővé teszi az újrahasználható UI-komponensek kialakítását, amelyek csökkentik a kódbázis redundanciáját és megkönnyítik a karbantartást. Ezáltal a felhasználói felület könnyen bővíthető és testre szabható.

#### Gyors és reszponzív felület

A React a Virtual DOM segítségével optimalizálja a felhasználói felület frissítését, így az alkalmazás gördülékenyen és hatékonyan kezeli a dinamikus tartalmat. Ez különösen fontos egy olyan platformon, ahol a felhasználók folyamatosan böngészik az álláshirdetéseket és interakcióba lépnek az elemekkel.

#### TypeScript támogatás

A TypeScript használata növeli a típusbiztonságot, csökkenti a hibalehetőségeket és segít a fejlesztőknek jobban átlátni a kódbázist. Ez hosszú távon stabilabb és megbízhatóbb fejlesztési környezetet biztosít.

#### Gazdag ökoszisztéma

A React széleskörű könyvtár- és eszköztámogatása révén könnyen integrálható külső komponensekkel (pl. React Router, Recharts, React-Toastify, Axios, AG-Grid), amelyek segítenek az alkalmazás gyors fejlesztésében és skálázhatóságában.

#### Gyors fejlesztés Vite segítségével

A Vite fejlesztői környezetként való használata rendkívül gyors hot reloadot és fejlesztési élményt biztosít, amely különösen hasznos nagyobb alkalmazások esetén.

### Miért ASP.NET Core a backendhez?

Magas teljesítmény és skálázhatóság

Az ASP.NET Core kiváló teljesítményt nyújt az asynchronous I/O és a lightweight, modularis architektúrája révén. Az optimalizált Kestrel webkiszolgáló lehetővé teszi, hogy az alkalmazás magas terhelés mellett is gyorsan és megbízhatóan működjön.

#### Többrétegű architektúra és tiszta kódszerkezet

Az ASP.NET Core támogatja az MVC (Model-View-Controller) architektúrát, ami segíti a kód szervezését és könnyebbé teszi az üzleti logika, az adatkezelés és a megjelenítés szétválasztását.

#### Beépített autentikáció és jogosultságkezelés

A rendszer JWT (JSON Web Token) alapú hitelesítést használ, amely biztonságos és skálázható megoldást kínál a felhasználói jogosultságok kezelésére. Az ASP.NET Core beépített Identity és Authentication Middleware támogatása egyszerűsíti az authentikáció és az engedélyezési folyamatokat.

#### Kiváló adatbázis támogatás és ORM

Az Entity Framework Core segítségével hatékonyan lehet kezelni az adatbázist, miközben lehetőség van MySQL, PostgreSQL vagy akár Microsoft SQL Server használatára is. Az ORM (Object-Relational Mapping) támogatás megkönnyíti a fejlesztést és csökkenti az SQL hibák esélyét.

#### REST API és könnyű integráció

Az ASP.NET Core Web API támogatásával RESTful szolgáltatásokat lehet létrehozni, amelyek gyorsan és biztonságosan kommunikálnak a frontenddel. A Swagger integráció lehetővé teszi az API dokumentáció automatikus generálását, megkönnyítve ezzel a fejlesztői munkát.

## ****Backend architektúra és API végpontok****

### **Általános backend felépítés**

A backend egy **ASP.NET Core** alapú **REST API**, amely **MySQL** adatbázist használ az adatok tárolására és kezelésére.

#### Az API több rétegből épül fel

* **Controllers**: A vezérlők felelősek az API végpontok kezeléséért és a kérések feldolgozásáért.
* **DatabaseHelper**: Egy segédosztály, amely az adatbázis-műveletek végrehajtásáért felelős.
* **Schema.sql**: Első indításkor egy üres induló adatbázisért létrehozásáért felelős.
* **Hitelesítés és jogosultságkezelés**: Az API **JWT tokeneket** használ a felhasználók hitelesítésére és jogosultságok ellenőrzésére.
* **Felhasználói szerepkörök**: A rendszerben négy szerepkör található (Admin, Organization, Agent, User), amelyek különböző hozzáférési jogosultságokkal rendelkeznek.

#### A backend struktúrája

* **AuthController**: Felelős a regisztrációért, bejelentkezésért, és token generálásért.
* **UserController**: A felhasználói adatok és jelentkezések kezelését végzi.
* **AgentController**: A közvetítők által kezelt munkák, műszakok és jelentkezések kezeléséért felel.
* **OrganizationController**: Az iskolaszövetkezetek általános műveleteit kezeli, ezek a munka létrehozás, módosítás, illetve a közvetítők felvétele és azok munkához rendelése.
* **AdminController**: Rendszergazdai feladat elvégzésére szolgál, főként a szövetkezetek létrehozására szolgál.
* **GeneralController**: Általános folyamatokat tartalmaz, nem feltétlenül szerepkörhöz kötött.

#### Általános jellemzők

* Token alapú hitelesítés (JWT)
* Felhasználói azonosítás és jogosultságellenőrzés
* Adatbázis-lekérdezések paraméterezett SQL utasításokkal
* Hibakezelés és megfelelő státuszkódok visszaadása

### **API végpontok listája**

#### Admin végpontok (/api/admin)

* **GET /users-by-month** – Visszaadja a regisztrált felhasználók számát havonta.
* **GET /total-organizations-and-users** – Lekéri az összes regisztrált szervezet és felhasználó számát.
* **GET /organizations** – Visszaadja az összes szervezet listáját.
* **PUT/organization/{organizationId}/password** – Egy szervezet adminisztrátorának jelszavát módosítja.
* **POST /new-organization** – Új szervezetet hoz létre.
* **PUT /settings** – Adminisztrációs beállításokat frissít.
* **PUT /organization/{organizationId}** – Frissíti egy szervezet adatait.
* **GET /admin-details** – Lekéri az adminisztrátor részleteit.

#### Agent végpontok (/api/agent)

* **GET /work-titles** – Visszaadja az elérhető munkák címét.
* **GET /agent-work-cards** – Lekéri az ügynökhöz rendelt munkák listáját.
* **GET /work-details/{id}** – Lekéri egy adott munka részletes adatait.
* **GET /manage-shifts/{jobId}** – Lekéri az adott munkához tartozó műszakokat.
* **GET /job-title/{jobId}** – Lekéri egy adott munka címét.
* **GET /shift-starts/{jobId}** – Lekéri egy adott munkához tartozó műszakkezdéseket.
* **GET /shift-applications/{jobId}** – Lekéri a munkára érkezett jelentkezéseket.
* **PATCH /shift-applications/{id}/accept** – Elfogad egy műszakra való jelentkezést.
* **PATCH /shift-applications/{id}/decline** – Elutasít egy műszakra való jelentkezést.
* **GET /applications** – Visszaadja az összes jelentkezést.
* **PATCH /applications/{id}/accept** – Elfogad egy jelentkezést.
* **GET /student-list** – Lekéri az ügynökhöz tartozó diákokat.
* **POST /add-shift** – Új műszakot ad hozzá egy munkához.
* **DELETE /delete-shift/{id}** – Töröl egy műszakot.
* **PATCH /applications/{id}/decline** – Elutasít egy jelentkezést.
* **PUT /profilesettings** – Frissíti a közvetítő profilbeállításait.

#### Auth végpontok (/api/auth)

* **POST /register** – Regisztrál egy új felhasználót.
* **POST /register-admin** – Regisztrál egy új adminisztrátort (csak hitelesített kliens használhatja).
* **POST /login** – Bejelentkezteti a felhasználót és visszaad egy JWT tokent.
* **POST /logout** – Kijelentkezteti a felhasználót.

#### General végpontok (/api/general)

* **GET /workcards** – Visszaadja az elérhető munkák listáját.
* **GET /workcards/{id}** – Lekéri egy adott munka részletes adatait.
* **GET /cities** – Lekéri az elérhető városokat.
* **GET /alluser** – Visszaadja az összes felhasználót.
* **PATCH /update-user-status/{id}** – Frissíti egy felhasználó státuszát.
* **PATCH /update-user-password/{id}** – Frissíti egy felhasználó jelszavát.
* **PATCH /update-user-profile/{id}** – Frissíti egy felhasználó profilját.
* **GET /jobreviews/{jobId}** – Lekéri az adott munkához tartozó értékeléseket.
* **POST /jobreviews** – Új értékelést ad hozzá egy munkához.

#### Organization végpontok (/api/organization)

* **GET /total-students-and-jobs/{orgId}** – Lekéri az adott szervezethez tartozó diákok és munkák számát.
* **GET /jobs-created-by-month/{orgId}** – Visszaadja az adott szervezet által havonta létrehozott munkákat.
* **POST /new-agent** – Új ügynököt hoz létre.
* **POST /new-job** – Új munkát hoz létre.
* **GET /jobs** – Visszaadja a szervezet által létrehozott munkák listáját.
* **GET /job/{id}** – Lekéri egy adott munka részleteit.
* **GET /categories** – Visszaadja az elérhető munkakategóriákat.
* **PATCH /assign-agent/{agentId}/{JobId}** – Hozzárendel egy ügynököt egy munkához.
* **GET /agents** – Lekéri a szervezethez tartozó közvetítők listáját.
* **PATCH /toggle-job-status/{jobId}** – Módosítja egy munka állapotát (aktív/inaktív).
* **PATCH /toggle-agent-status/{Id}** – Módosítja egy ügynök állapotát (aktív/inaktív).
* **PUT /update-job/{jobId}** – Frissíti egy adott munka adatait.
* **PUT /orgsettings** – Frissíti a szervezet beállításait.
* **GET /organization-details** – Lekéri a szervezet részletes adatait.

#### User végpontok (/api/user)

* **GET /user-jobs** – Lekéri a felhasználóhoz rendelt munkákat.
* **GET /user-applications** – Lekéri a felhasználó munkajelentkezéseit.
* **DELETE /delete-application/{applicationId}** – Töröl egy jelentkezést.
* **GET /profile** – Lekéri a felhasználó profilját.
* **POST /apply** – Jelentkezik egy munkára.
* **GET /list-user-shifts/date/{date}** – Lekéri a felhasználó műszakjait egy adott napon.
* **DELETE /delete-shift/{shiftId}** – Töröl egy műszakot.
* **GET /list-shifts/{jobId}/date/{date}** – Lekéri az adott munkához tartozó műszakokat egy adott napon.
* **POST /apply-shift** – Jelentkezik egy műszakra.
* **GET /student-details-datas** – Lekéri a felhasználó személyes adatait.
* **PUT /student-details** – Frissíti a felhasználó személyes adatait.

### **Példák API-hívásokra**

#### ****PUT**** /api/admin/organization/{organizationId}/password

* **Leírás**: Egy szervezet adminisztrátorának jelszavát módosítja.
* **Bemenet**: PasswordUpdateRequest

{

"newPassword": "NewStrongPass123”

}

* Válaszok:
  + **Sikeres jelszóváltoztatás** **(200 OK)**

{"message": "Jelszó sikeresen módosítva!"}

* + **Felhasználó nincs hitelesítve (401 Unauthorized)**

**{ "message": "Felhasználó azonosítása sikertelen." }**

* + **Érvénytelen bemenet - üres jelszó (400 Bad Request)**

**{ "message": "A jelszó nem lehet üres!" }**

* + **Érvénytelen bemenet - nem megfelelő jelszó (400 Bad Request)**

**{ "message": "Legalább egy mezőnek (email vagy jelszó) meg kell változnia." }**

* + **Nem létező szervezet (404 Not Found)**

**{ "message": "Szervezet nem található!" }**

* + **Adatbázis hiba (500 Internal Server Error)**

**{ "message": "Hiba történt a jelszó módosítása közben.", "details": "SQL error details here..."}**

#### ****POST**** /api/agent/add-shift

* **Leírás**: Új műszakot ad hozzá egy munkához.
* **Bemenet**: ShiftRequest

{

"jobId": 12,

"shiftStart": "2024-04-15T08:00:00",

"shiftEnd": "2024-04-15T16:00:00"

}

* Válaszok:
* **Sikeres műszak hozzáadás** **(200 OK)**

{"message": "Műszak sikeresen hozzáadva!"}

* **Felhasználó nincs hitelesítve (401 Unauthorized)**

{ "message": "Felhasználó azonosítása sikertelen." }

* **Érvénytelen bemenet (400 Bad Request)**

{ "message": "Érvénytelen adatok!" }

* **A műszak időpontja a múltban van (400 Bad Request)**

{ "message": "A műszak kezdő- vagy befejező időpontja nem lehet a jelenlegi dátum előtt!" }

* **A műszak túl későn kezdődik (400 Bad Request)**

{ "message": "A műszak kezdő időpontja nem lehet több, mint egy héttel a jelenlegi dátum után!" }

* **A műszak vége korábban van, mint a kezdete (400 Bad Request)**

{ "message": "A műszak befejező időpontja nem lehet korábbi, mint a kezdő időpont!" }

* **A műszak időtartama túl hosszú (400 Bad Request)**

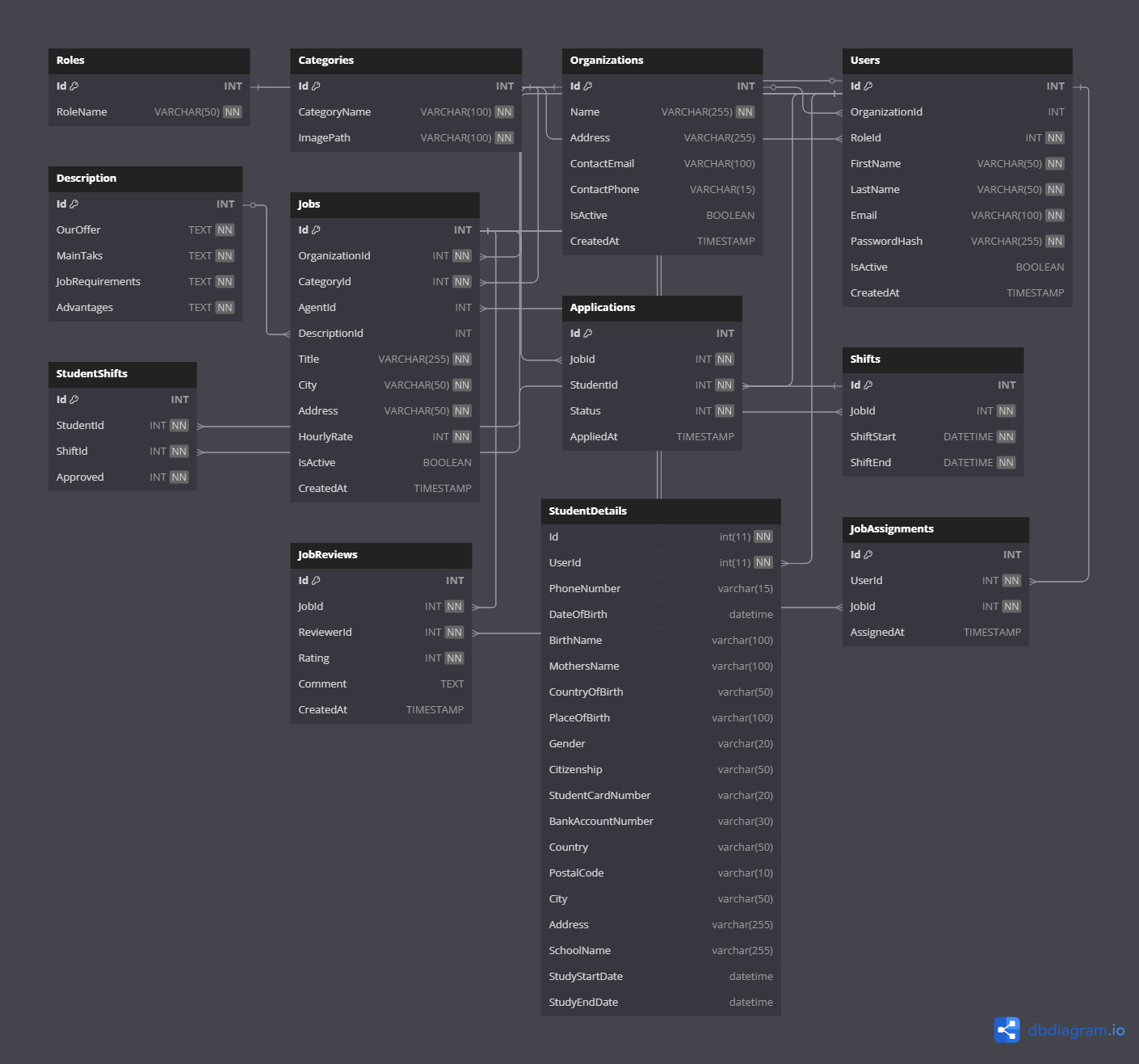
{ "message": "A műszak hossza nem haladhatja meg a 24 órát!" }

* **Adatbázis hiba (500 Internal Server Error)**

{ "message": "Hiba a műszak hozzáadása során!", "details": "SQL error details here..." }

## Kialakított adatszerkezet bemutatása

### Az adatbázis modell



### Főbb jellemzők

#### Felhasználói szerepkörök

A felhasználók különböző szerepkörökben működhetnek a rendszerben, amelyet a **Roles** tábla határoz meg.

* **Admin** (Adminisztrátor) – Teljes körű jogosultságokkal rendelkezik, menedzselheti a rendszert.
* **Organization** (Munkáltató, Iskolaszövetkezet) – Állásokat hirdethet és diákokat alkalmazhat.
* **Agent** (Közvetítő) – Az iskolaszövetkezet nevében koordinálja a jelentkezéseket és a diákokat.
* **User** (Diák) – Állásokra jelentkezhet, műszakokat vállalhat.

A **Users** tábla tárolja a felhasználók személyes adatait, e-mail címüket, jelszóhash-üket és szervezetük azonosítóját (ha van ilyen).

#### Szervezetek és állások kezelése

A **Organizations** tábla tartalmazza a munkáltató szervezeteket, amelyek állásokat hirdethetnek meg. Az **Jobs** tábla az állásokat tárolja, kategóriákba sorolva (CategoryId), és

szervezetekhez kapcsolódva (OrganizationId). Az AgentId mező egy adott User-re hivatkozik, aki az állásért felelős.

#### Kategóriák és leírások

Az **Categories** tábla az állások kategorizálására szolgál. Az egyes állások mindig egy adott kategóriába sorolhatók, például:

* Informatikai, mérnöki munkák
* Vendéglátás, gyorsétterem
* Áruházi, bolti, eladói

Az állások leírása külön táblában (**Description**) kerül tárolásra, amely tartalmazza:

* Feladatokat (MainTasks)
* Követelményeket (JobRequirements)
* Előnyöket (Advantages)
* Juttatások, mit ajánl a cég (OurOffer)

#### Jelentkezések és értékelések

A diákok jelentkezéseit az **Applications** tábla tárolja, amely rögzíti:

* A jelentkező diák azonosítóját (StudentId)
* A kiválasztott állás azonosítóját (JobId)
* A jelentkezés státuszát (Status)

A **JobReviews** tábla biztosítja, hogy a diákok értékelhessék a munkáltatókat:

* **Értékelés** (1-5 csillag)
* **Vélemény szövegesen**
* **Dátum**

Ez segíti a jövőbeli jelentkezőket a megfelelő munkahely kiválasztásában.

#### Munkarend és műszakok

A **Shifts** tábla az állásokhoz kapcsolódó műszakokat tárolja (ShiftStart, ShiftEnd).

A **StudentShifts** táblában tárolódik, hogy egy diák (StudentId) milyen műszakokat vállalt el.

#### Diákok adatai

A **StudentDetails** tábla a diákok részletes személyes adatait tartalmazza (pl. születési hely, anyja neve, bankszámlaszám, iskola).

#### Állásmegbízások

A **JobAssignments** tábla tartalmazza, hogy egy adott felhasználó (UserId) melyik álláshoz lett hozzárendelve.

### Kapcsolatok és idegenkulcsok

#### Felhasználók és szerepkörök kapcsolata

A **Users** (felhasználók) tábla egy idegen kulccsal (RoleId) kapcsolódik a **Roles** (szerepkörök) táblához. Egy felhasználónak mindig van egy szerepe, de egy szerepkörhöz több felhasználó is tartozhat.

* Kapcsolat típusa: Egy-sok (1:N)
* Idegen kulcs: **Users**.RoleId → Roles.Id
* Megkötések: **Users**.RoleId értéke csak olyan Id lehet, amely létezik a **Roles** táblában.

#### Felhasználók és szervezetek kapcsolata

A **Users** tábla kapcsolódik az **Organizations** (szervezetek) táblához, mivel egy felhasználó egy szervezethez tartozhat.

* Kapcsolat típusa: Egy-sok (1:N)
* Idegen kulcs: **Users**.OrganizationId → **Organizations**.Id
* Megkötések: Egy felhasználónak lehet szervezete, de nem kötelező (NULL megengedett).

#### Szervezetek és álláshirdetések kapcsolata

A **Jobs** (állások) tábla egy szervezethez kapcsolódik, mivel egy szervezet több állást is közzétehet.

* Kapcsolat típusa: Egy-sok (1:N)
* Idegen kulcs: **Jobs**.OrganizationId → **Organizations**.Id
* Megkötések: Egy álláshirdetés mindig egy szervezethez tartozik.

#### Diákok és állások kapcsolata (jelentkezések)

A **Users** tábla és a **Jobs** tábla közötti kapcsolatot a **Applications** tábla valósítja meg, mivel egy diák több állásra is jelentkezhet, és egy állásra több diák is jelentkezhet.

* Kapcsolat típusa: Sok-sok (M:N) köztes táblával
* Idegen kulcsok: **Applications**.StudentId → **Users**.Id

**Applications**.JobId → **Jobs**.Id

* Megkötések: Egy jelentkezés csak létező diákkal és állással történhet.

#### Állások és álláskategóriák kapcsolata

A **Jobs** tábla egy kategóriához kapcsolódik a **Categories** táblán keresztül, amely az elérhető álláskategóriákat tartalmazza.

* Kapcsolat típusa: Egy-sok (1:N)
* Idegen kulcs: Jobs.CategoryId → **JobCategories**.Id
* Megkötések: Egy álláshirdetés mindig egy kategóriába tartozik.

### Biztonsági szempontok

#### Jelszavak titkosítása

A felhasználók jelszavai bcrypt hash-eléssel kerülnek tárolásra.

#### Adatvédelem

Az érzékeny adatokat (pl. diák személyes adatai) külön táblában (StudentDetails) tároljuk.

#### Tranzakciókezelés

A jelentkezések és állások módosításai tranzakciókba foglalhatók, így elkerülhetők az inkonzisztens állapotok.

## Program főbb algoritmusai

### Autentikáció

#### Regisztrációs algoritmus

* Az alkalmazás ellenőrzi az e-mail cím formátumát (**IsValidEmail** függvény, regex validációval).
* Az alkalmazás biztosítja, hogy a jelszó megfeleljen a minimális biztonsági követelményeknek (**IsValidPassword** függvény).
* Az adatbázisban lekérdezi, hogy az adott e-mail cím már létezik-e.
* Ha az e-mail cím egyedi, a jelszó egy **bcrypt hash**-el titkosítva kerül tárolásra.
* Az új felhasználó bekerül az adatbázisba alapértelmezett szerepkörrel (**User – RoleId = 4**).
* A felhasználóhoz egy üres **StudentDetails** rekord is létrejön.

#### Bejelentkezési algoritmus

* Az alkalmazás lekéri az adott e-mail címhez tartozó felhasználói adatokat (ID, jelszó hash, szerepkör, aktív státusz).
* Ellenőrzi, hogy a fiók aktív-e.
* Az alkalmazás **bcrypt** segítségével összehasonlítja a megadott jelszót az adatbázisban tárolt hash értékkel.
* Sikeres bejelentkezés esetén az alkalmazás **JWT tokent** generál, amely a felhasználó azonosítóját és szerepkörét tartalmazza.

#### JWT token generálása

* A rendszer egy **JSON Web Token** (JWT) alapú hitelesítési mechanizmust használ.
* A **token** egyedi azonosítókat tartalmaz (userId, role), és egy titkos kulccsal van aláírva (GenerateJwtToken függvény).
* Ha a felhasználó a "Maradjak bejelentkezve" opciót választja, akkor a token érvényessége **2 év**, egyébként **1 nap**.

### Iskolaszövetkezet létrehozása

#### ****Adatok validálása****

Az API először ellenőrzi a bejövő kérést (**NewOrganizationRequest** objektumot), hogy minden kötelező mező ki legyen töltve:

* **OrgName** – az iskolaszövetkezet neve
* **Email** – az adminisztrátor e-mail címe
* **PhoneNumber** – az iskolaszövetkezet telefonszáma
* **Address** – az iskolaszövetkezet címe

Ha bármelyik mező üres, a rendszer hibát dob és megszakítja a feldolgozást.

Jelszó generálása

Mivel az új iskolaszövetkezet adminisztrátora kezdetben nem állít be saját jelszót, a rendszer egy véletlenszerű jelszót generál a **GenerateRandomPassword** metódussal.

Ez a metódus:

* Egy 10 karakter hosszú jelszót hoz létre, amely tartalmaz nagybetűket, kisbetűket és számokat.
* A jelszót bcrypt algoritmussal titkosítja, mielőtt adatbázisba kerülne.

#### E-mail küldése a felhasználónak

A rendszer automatikusan küld egy e-mailt az új adminisztrátornak a belépési adatokkal (**SendEmail** metódus).

Ez az e-mail tartalmazza:

* A regisztrált e-mail címet
* A generált jelszót
* Figyelmeztetést, hogy az első bejelentkezés után cserélje le a jelszót

### Műszak hozzáadása

#### Felhasználó azonosítás

Először ellenőrzi, hogy a felhasználó be van-e jelentkezve, ClaimTypes.NameIdentifier használatával.

#### Bemeneti adatok validálása

Ellenőrzi, hogy a JobId, ShiftStart és ShiftEnd érvényes-e

Budapesti időzónára konvertálja az időpontokat

Validálja, hogy:

* A műszak nem kezdődhet a múltban
* A műszak nem kezdődhet egy hétnél később
* A befejezés nem lehet a kezdés előtt
* A műszak hossza nem haladhatja meg a 12 órát

#### Adatbázis művelet

SQL INSERT parancsot hajt végre a Shifts táblába

Paraméterezetten adja át az értékeket az SQL injekcióval szembeni védelem érdekében

#### Visszatérési érték kezelése

Siker esetén 200 OK státuszkóddal tér vissza

Hiba esetén 500-as hibakóddal és részletes hibaüzenettel

### Műszak törlése és email értesítés

#### Műszak adatok lekérése

* Ellenőrzi, hogy a törlendő műszak létezik-e a **Shifts** táblában
* Lekéri a műszak részleteit (jobId, kezdési és befejezési idő)

#### Érintett diákok azonosítása

* Lekérdezi az összes diákot, aki jelentkezett a műszakra a **StudentShifts** táblából

#### Adatbázis műveletek

* Először törli a jelentkezéseket a **StudentShifts** táblából
* Ezután törli magát a műszakot a **Shifts** táblából

#### Email értesítések küldése

Minden érintett diáknak egyenként:

* + Lekéri a diák elérhetőségét és nevét a **Users** táblából
  + Személyre szabott üzenetszöveget generál, amely tartalmazza:
    - A diák teljes nevét
    - A törölt műszak időpontját (kezdés és befejezés)
    - Általános tájékoztatást és iránymutatást
  + Külső **SMTP** szolgáltatást használ (Gmail) a levelek kiküldésére
  + Az email küldés hibakezelését a metóduson belül kezeli le

#### Az email küldés algoritmusa (SendEmail privát metódus)

* **SMTP** kliens létrehozása megfelelő konfigurációval (host, port, SSL, hitelesítés)
* **MailMessage** objektum összeállítása (feladó, címzett, tárgy, szövegtörzs)
* Az email elküldése és erőforrások felszabadítása
* Hibakezelés és naplózás kivételek esetén

## Tesztdokumentáció

Tesztkörnyezet és tesztelési stratégiák

* Fejlesztői környezet (Windows, böngészők, adatbázis szerverek)
* Automata tesztelés (unit és integrációs tesztek)

#### Funkcionális tesztelés

* Regisztráció és bejelentkezés ellenőrzése
* Jogosultságok tesztelése (pl. admin funkciók)

#### Hibakezelés és validáció tesztelése

* Rossz formátumú adatok tesztelése
* Edge case-ek és hibás bemenetek kezelése

## Továbbfejlesztési lehetőségek

#### Jogosultságkezelés és szerepkörök bővítése

Jelenleg az adminisztrátorok manuálisan hozzák létre az iskolaszövetkezeteket. Bevezethető lenne egy önregisztrációs folyamat az iskolaszövetkezetek számára, amelyet az admin jóváhagyhat.

A jelenlegi szerepkörök mellé új jogosultsági szintek is definiálhatók, például moderátorok vagy részleges adminisztrátorok, akik bizonyos műveleteket elvégezhetnek.

#### Fejlettebb keresési és szűrési lehetőségek

Az állások keresése jelenleg alapvető paraméterek szerint történik. Bővíteni lehetne többdimenziós szűrési lehetőségekkel, például órabérsáv, elérhetőség vagy munkáltatói értékelések alapján.

#### Értesítési rendszer bevezetése

Jelenleg a felhasználók e-mailben kapnak visszajelzéseket bizonyos eseményekről (pl. regisztráció, jelszómódosítás).

Hasznos lenne egy push értesítési rendszer vagy beépített üzenetközpont, amely valós időben tájékoztatná a felhasználókat az álláslehetőségekről, elfogadott jelentkezésekről vagy új üzenetekről.

#### Automatizált munkaszerződés-generálás

Jelenleg a diákok jelentkezés után adminisztratív módon kerülnek be az iskolaszövetkezetek adatbázisába.

Egy automatikus munkaszerződés-generáló rendszer bevezetése lehetővé tenné a digitális szerződéskötést a platformon belül, amely jogilag hiteles dokumentumokat generál PDF formátumban.

#### Fejlettebb statisztikai modul

Az adminisztrátorok számára egy részletesebb analitika beépítése hasznos lenne, például a munkák sikerességi arányáról, a diákok aktivitásáról vagy az egyes állásajánlatok népszerűségéről.

Interaktív grafikonok segítségével az adminisztrátorok jobban átláthatnák a rendszer működését.

Mobilalkalmazás fejlesztése

A jelenlegi rendszer webalkalmazásként működik, de egy natív mobilalkalmazás (Android/iOS) fejlesztése jelentősen növelné a felhasználói élményt.

A mobilalkalmazás push értesítésekkel, gyors regisztrációval és térképes keresővel egészíthetné ki a jelenlegi funkciókat.

#### Összegzés

A rendszer számos módon továbbfejleszthető, beleértve az automatizálást, az élő értesítéseket, a statisztikai elemzéseket és a mobilalkalmazás integrálását. A jövőbeli fejlesztések célja, hogy a platform még felhasználóbarátabb, hatékonyabb és skálázhatóbb legyen a diákmunka piacán.

# Felhasználói dokumentáció

## A program célja és funkciói

## Szükséges hardver- és szoftverkövetelmények

## Telepítési és indítási útmutató

## A programról részletesen

## Hibakezelés és hibaüzenetek magyarázata

# Összefoglalás

## Fejlesztési tapasztalatok

## Hogyan tovább?

## Eredmények értékelése