

SOPRONI SZAKKÉPZÉSI CENTRUM
VAS- ÉS VILLAMOSIPARI
TECHNIKUM



Soproni Szakképzési Centrum

Vas- és Villamosipari Technikum

PitchforkForms

Szerzők:

Ragats Bálint

Németh Mirkó Máté

Németh Bence

Témavezető:

Németh Noel

Sopron, 2025

Tartalomjegyzék

1. Bevezetés.....	4
1.1. Téma ismertetése.....	4
1.2. Témaválasztás indoklása	4
1.3. Célkitűzések.....	4
2. Fejlesztői dokumentáció	6
2.1. Hardver- és szoftverkövetelmények a helyi futtatáshoz.....	6
2.1.1. Hardverkövetelmények	6
2.1.2. Szoftverkövetelmények.....	6
2.1.3. Telepítés és program indítása	7
2.2. Fejlesztőkörnyezet.....	9
2.2.1. Fejlesztéshez használt egyéb szoftverek	9
2.3. Adatszerkezet	11
2.3.1. Adatbázis tervezés/dbdiagram.io használata	11
2.3.2. Az adatbázis szerkezete.....	13
2.4. Fájl szerkezet	15
2.4.1. Frontend fájl szerkezete részekre bontva	15
2.4.2. Backend fájl szerkezete részekre bontva	16
2.5. Programozási kihívások	18
2.5.1. Problémák	18
2.5.2. Megoldásuk	18
2.5.3. Algoritmusok	41
2.6. Tesztelés	44
2.6.1. Backend tesztelés	44
2.6.2. Frontend tesztelés.....	46
2.6.3. Felhasználói esetek.....	48
2.6.4. Felhasználói történetek: Given-When-Then	49
2.7. Továbbfejlesztési lehetőségek	52
3. Felhasználói dokumentáció	53
3.1. Program funkcionalitásainak bemutatása.....	53
3.1.1. Tanárok és adminisztrátorok számára	53

3.1.2.	Diákok számára.....	55
3.1.3.	Általános funkciók	57
3.1.4.	Hibajelzések, magyarázatuk, megoldásuk.....	59
3.2.	Információkérés lehetőségei	60
4.	Összefoglalás.....	61
5.	Irodalomjegyzék	62
6.	Melléklet.....	63

1. Bevezetés

Ebben a fejezetben ismertetjük a témát részletesen. Megindokoljuk, hogy miért ezt a témát választottuk a Vizsgaremekünk alapjául, valamint további célkitűzéseket fogalmazunk meg a projekt elkészítéséhez.

1.1. Téma ismertetése

A projekt célja egy olyan web alkalmazás létrehozása, amely a tanárok és diákok számára nyújt interaktív feladatlap-készítési és kitöltési lehetőséget. Az alkalmazás lehetővé teszi a tanárok számára, hogy különböző típusú kérdéseket tartalmazó feladatlapokat hozzanak létre, amelyeket a diákok online kitölthetnek, majd az eredményeket a rendszer automatikusan kiértékeli.

A fejlesztés során egy olyan struktúrált és felhasználóbarát megoldás megvalósítására törekszünk, amely egyszerűsíti a tanárok munkáját, miközben a diákok számára egy könnyen elérhető és átlátható felületet biztosít a feladatok megoldására.

1.2. Témaválasztás indoklása

A digitális oktatás egyre nagyobb szerepet kap a modern tanítási és tanulási folyamatokban, ezért egy olyan webes alkalmazás fejlesztése, amely támogatja a tanárok és diákok közötti interaktív feladatlapkezelést, releváns és hasznos megoldás és megkönnyíti mind a tanárok, mind a diákok életét is.

A választott téma lehetőséget ad arra, hogy egy korszerű, könnyen használható és testre szabható megoldást fejlesszünk, amely a hagyományos papíralapú feladatlapokat egy hatékonyabb, online formátumra cseréli.

1.3. Célkitűzések

A projekt célja egy webes alkalmazás létrehozása, amely lehetőséget biztosít a tanároknak interaktív feladatlapok létrehozására és a diákok számára azok kitöltésére. A fejlesztés során az alábbi konkrét célokat tűzzük ki:

- **Felhasználóbarát kezelőfelület kialakítása:** Az alkalmazás intuitív és könnyen kezelhető legyen mind a tanárok, mind a diákok számára.
- **Feladatlapok létrehozásának és szerkesztésének lehetősége:** A tanárok különböző típusú kérdéseket tartalmazó feladatlapokat hozhassanak létre és szerkeszthessenek.
- **Feladatlapok kitöltése és beküldése:** A diákok számára egyszerű és gyors kitöltési lehetőség biztosítása bármilyen eszközről (asztali gép, laptop, mobiltelefon, tablet).
- **Automatikus kiértékelés és eredmények megjelenítése:** A rendszer automatikusan javítsa az egyszerűbb típusú feladatokat (pl. feleletválasztós kérdések), és az eredményeket egyértelműen jelenítse meg a diákoknak.
- **Felhasználói fiókok kezelése:** A tanárok és diákok számára személyre szabott fiókok biztosítása.
- **Adatbiztonság és stabil működés:** A rendszer biztosítsa az adatok biztonságos tárolását és a folyamatos, megbízható működést.
- **Reszponzív dizájn:** Az alkalmazás minden eszközön megfelelően működjön és jól nézzen ki.
- **Feladatlapok státusza:** A tanárok megnézhesék, hogy a diákok hogy haladnak a feladatlapokkal és láthassák az eredményeiket a már kitöltött feladatlapoknál.
- **Felhasználói értesítések:** A rendszer értesítse a felhasználókat bizonyos eseményekről emailben, mint például, ha egy diáknak új feladatlap érkezett.
- **Felhasználói hitelesítés és jogosultságkezelés:** A rendszer biztonságos bejelentkezést biztosítson JWT tokenekkel és a felhasználók szerepkörük alapján csak a releváns funkciókhoz férhessenek hozzá.
- **Adminisztrátori jogosultságok kezelése:** Az adminisztrátorok számára lehetőség legyen felhasználókat kezelni (lássa az összes felhasználót és tudja törölni azokat ha szükség van rá)

2. Fejlesztői dokumentáció

Ez a dokumentáció a Pitchfork Forms web alkalmazás fejlesztését mutatja be.

2.1. Hardver- és szoftverkövetelmények a helyi futtatáshoz

2.1.1. Hardverkövetelmények

- **Minimális követelmények:**
 - Processzor: 2 GHz-es, 2 magos CPU
 - Memória: 4 GB RAM
 - Tárhely: 500 MB szabad hely
- **Ajánlott követelmények:**
 - Processzor: 2,5 GHz-es, 4 magos CPU
 - Memória: 8 GB RAM
 - Tárhely: 1 GB szabad hely

2.1.2. Szoftverkövetelmények

- **Operációs rendszer:**
 - Windows 10 vagy újabb
- **Böngésző:**
 - Google Chrome (ajánlott)
 - Mozilla Firefox
 - Microsoft Edge
 - Brave
- **Egyéb követelmények:**
 - Node.js (14-es vagy újabb verzió)

- npm csomagkezelő

2.1.3. Telepítés és program indítása

2.1.3.1. Frontend telepítése:

- **Követelmények telepítése:**
 - Telepítsd a Node.js és npm csomagkezelőt.
- **Projekt klónozása:**

```
git clone https://github.com/getrektgit/pitchforkformsfrontend.git  
cd pitchforkformsfrontend
```

- **Függőségek telepítése:**

```
npm install
```

- **Fejlesztői szerver indítása:**

```
npm run dev
```

- Az alkalmazás elérhető lesz a `http://localhost:5173` címen.

2.1.3.2. Backend telepítése:

- **Projekt klónozása:**

```
git clone https://github.com/getrektgit/pitchforkforms-backend.git  
cd pitchforkforms-backend
```

- **Függőségek telepítése:**

```
npm install
```

- **Adatbázis beállítása:**

```
node seed
```

- **Admin fiók bejelentkezés:**
 - email cím: pitchforkformsnotify@gmail.com
 - jelszó: pitchfork123
- **Környezet változók beállítása:**
 - Hozz létre egy .env fájlt a .env.example fájl alapján.

```
PORT=your_port_here
# Example: PORT=3000
SECRET_KEY=your_secret_key_here
# Example: SECRET_KEY=grwehfdngrhjtrjkzt
REFRESH_SECRET=your_refresh_secret_here
# Example: REFRESH_SECRET=hephtmjjztjzukupőkukuzhdf
NODE_ENV=your_environment_here
# Example: NODE_ENV=development
# Example: NODE_ENV=production
# Example: NODE_ENV=test
MAIL_USER=your_email_here
# Example: example@example.com
MAIL_PASS=your_email_password_here
# Example: example_password

DB_HOST=your_db_host_here
# Example: DB_HOST=localhost
DB_USER=your_db_user_here
# Example: DB_USER=root
# Example: DB_USER=your_db_user
DB_PASSWORD=your_db_password_here
# Example: DB_PASSWORD=none
DB_NAME=your_db_name_here
# Example: DB_NAME=pitchforkforms
```

- A MAIL_PASS létrehozásához létre kell hozni egy applikációs jelszót a gmail-fiókon belül, amit ezen a linken tudunk létrehozni:
<https://myaccount.google.com/apppasswords>

```
MAIL_PASS=your_email_password_here
# Example: example_password
```

- Fejlesztői szerver indítása:

```
node server.js
```

- A backend elérhető lesz a <http://localhost:3000> címen.

2.2. Fejlesztőkörnyezet

A projekt fejlesztéséhez az alábbi fejlesztői környezetet használtuk:

Visual Studio Code (VS Code):

- **Miért választottuk?**
 - **Könnyű használat:** A VS Code egyszerűen kezelhető, és számos bővítménnyel testreszabható.
 - **Bővítmények támogatása:** A projekt során használtunk olyan bővítményeket, mint az ESLint a kódminőség biztosítására és a kódformázásra, ES7+ React/Redux/React-Native snippets, Simple React Snippets.
 - **Beépített terminál:** A beépített terminál lehetővé tette a backend és frontend szerverek egyidejű futtatását.
 - **Git integráció:** A verziókezeléshez használt Git egyszerűen kezelhető a VS Code beépített eszközeivel.
 - **Beépített debugger:** A beépített hibakereső lehetővé tette a frontend (React) és backend (Node.js) kód egyszerű és hatékony debugolását.
 - **GitLens bővítmény:** A GitLens bővítmény segítségével könnyen nyomon követhettük a kódváltozásokat, commitokat és a csapattagok által végzett módosításokat.
 - **Snippets támogatás:** A VS Code lehetővé tette egyedi kódrészletek (snippets) létrehozását, amelyekkel gyorsabban tudtunk ismétlődő kódokat generálni, például React komponenseket.
 - **Közösségi támogatás:** A VS Code hatalmas közösségi támogatással rendelkezik, így könnyen találhattunk megoldásokat a felmerülő problémákra, valamint számos bővítményt és eszközt használhattunk a fejlesztés során.

2.2.1. Fejlesztéshez használt egyéb szoftverek

- **XAMPP:**
 - **Miért választottuk?**

- **Egyszerű telepítés:** A XAMPP egy komplett, könnyen telepíthető környezetet biztosít, amely tartalmazza az Apache szervert és a MySQL adatbázist is.
- **PHPMyAdmin:** A beépített PHPMyAdmin segítségével grafikus felületen keresztül tudtuk kezelni az adatbázisokat, ami gyorsabb és átláthatóbb adatbázis-kezelést tett lehetővé.
- **Platformfüggetlenség:** Windows, Linux és macOS rendszereken is használható, így ha a csapattagok különböző operációs rendszert használnak, akkor is egységes fejlesztőkörnyezet van biztosítva.
- **Postman:**
 - **Miért választottuk?**
 - **API tesztelés egyszerűen:** A Postman segítségével könnyedén tesztelni tudtuk az API végpontokat, ellenőrizni a válaszokat, hibakódokat és a JSON formátumú adatokat.
 - **Kérés-sorozatok tárolása:** A projekt során használt API hívásokat elmenthettük kollekciókba, így gyorsan és egyszerűen újra lefuttathattuk azokat a fejlesztés vagy tesztelés során.
 - **Egyszerű felhasználói felület:** A grafikus kezelőfelület megkönnyítette a paraméterek, fejlécek és törzsek kezelését, még azok számára is, akik kevésbé jártasak az API-k működésében.
 - **GitHub:**
 - **Miért választottuk?**
 - **Verziókezelés és együttműködés:** A GitHub lehetővé tette számunkra, hogy hatékonyan kövessük a projekt fejlesztési folyamatát, visszakereshetővé tett minden változtatást, és egyszerűbbé tette a csapatmunkát több fejlesztő között.
 - **Távoli tárolás:** A GitHub a fejlesztés alatt biztosította a forráskód biztonságos, felhőalapú tárolását, így a csapattagok bárhol, bármikor elérhették a projektet.
 - **Pull request-ek és kódfelülvizsgálat:** A pull request funkció révén strukturáltan tudtunk új fejlesztéseket integrálni, valamint lehetőséget adott egymás kódjának átnézésére, hibák kiszűrésére.

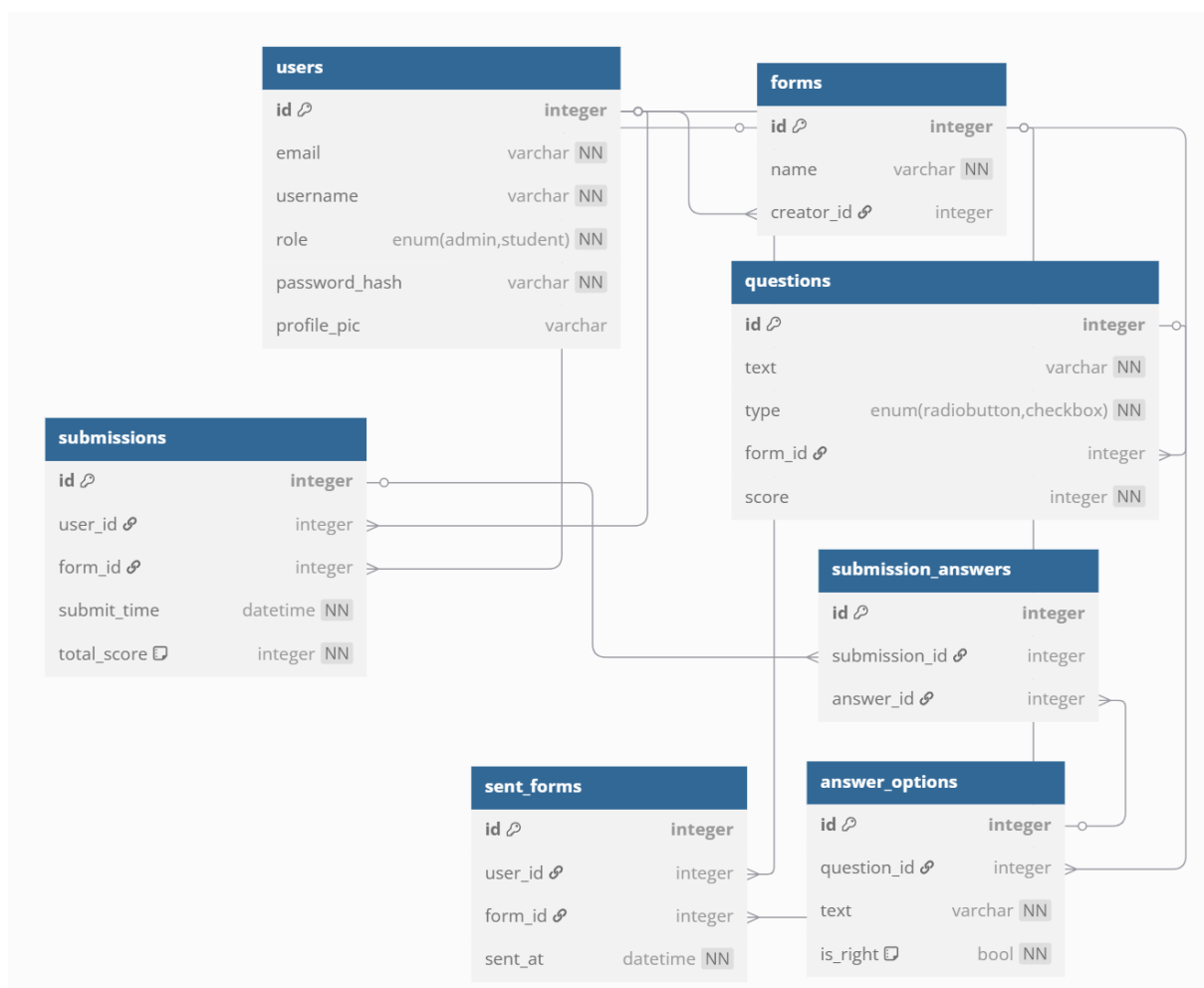
2.3. Adatszerkezet

2.3.1. Adatbázis tervezés/dbdiagram.io használata

A Pitchfork Forms webalkalmazás adatbázisának tervezése során vizuális támogatásként a dbdiagram.io szolgáltatást használtuk. Ez a weboldal lehetővé tette, hogy gyorsan és átlátható módon hozzunk létre egy relációs adatmodell-diagramot, amely a MySQL adatbázis struktúráját szemlélteti.

A diagram vizuálisan ábrázolja az adatbázis főbb entitásait, azok mezőit, valamint a köztük lévő kapcsolati viszonyokat (pl. idegen kulcsok). Ez a fajta ábrázolás különösen hasznos volt az adatbázis szerkezetének kezdeti megtervezésében és későbbi finomhangolásában, mivel segítette az összefüggések jobb megértését és az esetleges redundanciák vagy tervezési hibák korai kiszűrését.

A dokumentációban szereplő diagram pillanatképet nyújt az adatbázis struktúrájáról, így a fejlesztők és későbbiekben a karbantartást végző személyek is könnyebben eligazodnak a rendszer felépítésében.



2.3.2. Az adatbázis szerkezete

2.3.2.1. *users* tábla

A *users* tábla tárolja az alkalmazásba regisztrált felhasználók adatait. Minden felhasználó rendelkezik egyedi azonosítóval (*id*), e-mail címmel, felhasználónévvel, szerepkörrel (*admin* vagy *student*), valamint jelszó hash-sel a biztonságos hitelesítéshez. Opcionálisan profilképet is lehet tárolni (*profile_pic*).

2.3.2.2. *forms* tábla

A *forms* tábla tartalmazza a tanárok által létrehozott feladatlapokat. Minden feladatlaphoz tartozik egy név (*name*) és létrehozó tanár azonosítója (*creator_id*).

2.3.2.3. *questions* tábla

A *questions* tábla a feladatlapokhoz tartozó kérdéseket tartalmazza. Minden kérdés szövege (*text*), típusa (*radiobutton* vagy *checkbox*), az adott feladatlap azonosítója (*form_id*), valamint az elérhető pontszám (*score*) itt kerül tárolásra.

2.3.2.4. *answer_options* tábla

Ez a tábla tárolja az egyes kérdésekhez tartozó válaszlehetőségeket. Minden válasz tartalmazza a szöveget (*text*), hogy melyik kérdéshez tartozik (*question_id*), és egy logikai mezőt (*is_right*), amely megjelöli, hogy az adott válasz helyes-e.

2.3.2.5. *submissions* tábla

A *submissions* tábla rögzíti a diákok által beküldött feladatlapokat. Tartalmazza, hogy melyik diák (*user_id*) mikor küldte be (*submit_time*) az adott feladatlapot (*form_id*), és mekkora összpontszámot ért el (*total_score*).

2.3.2.6. *submission_answers* tábla

Ez a tábla kapcsolja össze a diák beküldött válaszait az egyes válaszlehetőségekkel. Minden rekord egy *submission_id*-hez és egy kiválasztott *answer_id*-hoz kapcsolódik, ezzel rögzítve, hogy a diák milyen válasz(oka)t adott.

2.3.2.7. *sent_forms* tábla

A `sent_forms` tábla dokumentálja, hogy melyik felhasználónak melyik feladatlap lett kiküldve, és mikor (`sent_at`). Ez segít nyomon követni a kiküldések történetét, és lehetővé teszi a kitöltési státuszok követését.

2.4. Fájlszerkezet

2.4.1. Frontend fájlszerkezete részekre bontva

- **pitchforkformsfrontend (Frontend könyvtár)**
 - Ez a könyvtár tartalmazza a Pitchfork Forms alkalmazás frontend részének teljes forráskódját. A projekt modern React-alapú fejlesztési környezetet használ.
- **coverage:**
 - Ez a mappa a tesztelési lefedettségi riportokat tartalmazza, amelyeket a jest generál. A lcov-report almappa HTML formában is megjeleníti az egyes fájlokhoz tartozó lefedettségi adatokat.
- **node_modules:**
 - A Node.js által telepített külső csomagokat tartalmazza. Ezt a mappát a npm install vagy yarn install parancs hozza létre automatikusan a package.json alapján.
- **src:**
 - A fő forráskód mappa, amely tartalmazza a React komponenseket, tesztfájlokat, stílusokat és alkalmazás belépési pontjait.
- **__tests__:**
 - Ez a mappa tartalmazza az egységteszteket. Jelenleg például az AdminPage.test.jsx és MainPage.test.jsx fájlok találhatók itt, amelyek a megfelelő oldalak működését tesztelik.
- **Components:**
 - A projekt összes újrafelhasználható React komponense itt található, beleértve oldalakat, formákat, modálokat és navigációs elemeket.
- **Pages:** Teljes oldalkomponensek (pl. MainPage.jsx, AdminPage.jsx, StudentPage.jsx)
- **Egyéb komponensek:**
 - FormCard.jsx, FillOutFormFormat.jsx, DefaultFormFormat.jsx, CompletedFormCard: Űrlapokhoz kapcsolódó elemek.
 - LoginModal.jsx, RegisterModal.jsx: Bejelentkezési és regisztrációs felületek.
 - Navbar.jsx: Navigációs sáv.
- **images:**

- Ez a mappa tartalmazza a statikus képeket, például a carousel komponenshez tartozó képfájlokat (carouselpic1.jpg, carouselpic2.jpg, carouselpic3.jpg).
- **Gyökérfájlok:**
 - **App.jsx, main.jsx:** Az alkalmazás belépési pontjai, innen indul a renderelés.
 - **index.css, App.css:** Globális és alkalmazásszintű stílusok.
 - **.gitignore:** Git konfiguráció, a verziókövetésből kizárt fájlok listája.
 - **babel.config.js, vite.config.js:** Transzpilációs és build beállítások.
 - **jest.config.js, jest.setup.js:** Tesztkörnyezet konfigurációi.
 - **package.json, package-lock.json:** Függőségek és projekt metaadatok.
 - **README.md:** Projektleírás, telepítési és használati útmutató.

2.4.2.Backend fájlszerkezete részekre bontva

- **__tests__**
 - Tesztek a backend funkcionalitás ellenőrzésére.
 - **userRoutes.test.js:** Teszteli a userRoutes-hoz kapcsolódó végpontokat (pl. regisztráció, bejelentkezés).
- **Config**
 - Konfigurációs fájlokat tartalmaz.
 - **database.js:** Adatbázis-kapcsolat beállításai.
- **Coverage**
 - Automatikusan generált mappa, amely a tesztelés lefedettségi adatait tárolja (pl. Jest által létrehozva).
- **Middlewares**
 - Köztes rétegek, amelyek a kérések feldolgozása előtt futnak.
 - **authMiddleware.js:** Ellenőrzi, hogy a felhasználó be van-e jelentkezve (token validálása).
 - **roleCheckMiddleware.js:** Ellenőrzi, hogy a felhasználónak van-e admin jogosultsága (csak admin).
- **Routes**
 - API végpontokat kezelő fájlok.
 - **formRoutes.js:** A feladatlapokkal kapcsolatos végpontok (létrehozás, beküldés, megtekintés).

- **userActions.js:** Felhasználói műveletek: profil szerkesztés, eredmények megtekintése.
- **userAuth.js:** Regisztráció, bejelentkezés, kijelentkezés kezelése.
- **Utils**
 - Segédfüggvények, amiket más részek többször használnak.
 - **notifyUser.js:** E-mail értesítés küldése a felhasználónak.
 - **queryHelper.js:** Segéd az SQL lekérdezésekhez (db.query, paraméterkezelés).
- **Egyéb fájlok:**
 - **.env, .env.example:** Környezeti változók (pl. adatbázis jelszó, port).
 - **.gitignore:** Git által figyelmen kívül hagyott fájlok listája.
 - **jest.config.js:** Jest tesztelési konfiguráció.
 - **package.json, package-lock.json:** Projektfüggőségek és szkriptek.
 - **pitchforkforms_updated.sql:** SQL fájl az adatbázis szerkezetéhez.
 - **README.md:** A projekt indítási és telepítési útmutatója.
 - **seed.js:** Alapadatok feltöltése az adatbázisba (seedelés).
 - **server.js:** A szerver belépési pontja (Express alkalmazás indítása).

2.5. Programozási kihívások

A webalkalmazás fejlesztése során számos technikai kihívással találkoztunk, amelyek megoldása elengedhetetlen volt a rendszer biztonságos, megbízható és felhasználóbarát működéséhez. Ezek a kihívások érintették többek között a felhasználói hitelesítést, az űrlapkezelést, az adatbiztonságot, valamint az automatizált értékelési folyamatokat. Az alábbiakban részletezzük a legjelentősebb problémákat, azok megoldási stratégiáit, és az alkalmazott algoritmusokat.

2.5.1. Problémák

- Bejelentkezés (Access/Refresh Token)
- Regisztráció
- Űrlap létrehozása
- Űrlap szerkesztése
- Egy űrlap kitöltése
- Űrlap kiküldés és email-es értesítő
- Kiküldött űrlapok kilistázása
- Kitöltött űrlapok kilistázása
- Diák megtekintheti a saját válaszait a már kitöltött űrlapokon
- Felhasználói profil szerkesztése

2.5.2. Megoldásuk

Ebben a részben részletezzük azokat a technikai megoldásokat, amelyekkel a fenti problémákat leküzdöttük a webalkalmazás fejlesztése során.

2.5.2.1. Bejelentkezés (Access/Refresh Token)

2.5.2.1.1 Backend

- **Kihívás:** A rendszernek biztonságos, állapotfüggetlen hitelesítést kellett biztosítania, amely lehetővé teszi a gyors API-hozzáférést, és ugyanakkor megakadályozza az illetéktelen hozzáférést. Emellett szükség volt arra, hogy a felhasználó ne lépjen ki automatikusan minden egyes access token lejárata után.
- **Megoldás:** A bejelentkezés során a felhasználótól megadott email és jelszó alapján történik az azonosítás. A jelszót bcrypt segítségével hasonlítjuk össze a tárolt hash-sel. Sikeres hitelesítés után kétféle JSON Web Token generálunk:
 - Access Token: rövid élettartamú (1 óra), a kliens ezzel éri el a védett API-kat.

- Refresh Token: hosszabb élettartamú (3 óra), biztonságosan HTTP-only cookie-ban kerül tárolásra. Ezzel kérhető új access token anélkül, hogy újra be kellene jelentkezni.
- A refresh tokeneket szerveroldalon egy `activeRefreshTokens` gyűjteményben követjük, így lehetőség van azok érvénytelenítésére is például kijelentkezéskor.
- **Kódrészlet:**

```
const token = jwt.sign(  
  { id: user.id, email: user.email, role: user.role },  
  SECRET_KEY,  
  { expiresIn: "1h" }  
);  
  
const refreshToken = jwt.sign(  
  { id: user.id, username: user.username },  
  process.env.REFRESH_SECRET,  
  { expiresIn: "3h" }  
);  
  
activeRefreshTokens.add(refreshToken);  
  
res.cookie("refreshToken", refreshToken, {  
  httpOnly: true,  
  secure: true,  
  sameSite: "Strict",  
  maxAge: 3 * 60 * 60 * 1000,  
});
```

2.5.2.1.2 Frontend

- **Kihívás:** A frontend oldalon egy olyan felhasználóbarát és biztonságos bejelentkezési felületet kellett létrehozni, amely hatékonyan kommunikál a backenddel, képes kezelni a hibákat, és figyelembe veszi a különböző felhasználói szerepköröket (pl. admin, diák). Fontos szempont volt a reszponzivitás, valamint az állapotkezelés és az automatikus navigáció megfelelő kezelése is.
- **Megoldás:** A bejelentkezési funkció React-ben, MUI (Material UI) komponensek segítségével valósult meg, modern felhasználói élményt biztosítva. A `LoginModal` komponens egy modális ablakban jeleníti meg a bejelentkezési űrlapot. A beírt email és jelszó az `axios` segítségével POST-kérésként kerül továbbításra a `/auth/login` végpontra.
 - **Sikeres bejelentkezés esetén:**
 - A szervertől kapott `accessToken` a `localStorage`-ba kerül eltárolásra.

- A „Remember Me” opció bejelölése esetén egy plusz flag (rememberMe) szintén a localStorage-ba kerül, amely alapján a kliens eldöntheti, hogy megőrizze-e a felhasználói állapotot.
- A felhasználó szerepkörének megfelelő oldalra történik átirányítás (pl. admin felületre vagy diák dashboardra).
- A onLoginSuccess callback segítségével a szülő komponens frissítheti az alkalmazás globális állapotát (pl. beállíthatja a felhasználó adatait).
- Sikertelen bejelentkezés esetén a hibaüzenet egy Alert komponensben jelenik meg a felületen.
- **Kódrészlet:**

```
const response = await axios.post('/auth/login', formData, {
  withCredentials: true,
});

localStorage.setItem('accessToken', response.data.token);

rememberMe
  ? localStorage.setItem('rememberMe', 'true')
  : localStorage.removeItem('rememberMe');

if (onLoginSuccess) {
  onLoginSuccess({
    id: response.data.id,
    email: formData.email,
    username: response.data.username,
    role: response.data.role,
  });
}

if (response.data.role === 'admin') {
  navigate('/admin');
} else if (response.data.role === 'student') {
  navigate('/student');
} else {
  navigate('/');
}
```

2.5.2.2. Regisztráció

2.5.2.2.1 Backend

- **Kihívás:** Biztonságos és hibamentes felhasználói regisztráció megvalósítása, amely megelőzi a hiányos adatokkal történő regisztrációt, valamint megfelelően tárolja a jelszót.

Emellett szükség volt az alapértelmezett jogosultsági szint (pl. „student”) automatikus beállítására is.

- **Megoldás:** A regisztráció során az alábbi mezők megadása kötelező: email, username, password. A jelszót bcrypt segítségével hash-eljük (titkosítjuk), majd az adatokat SQL-lekérdezéssel tároljuk el az adatbázisban. A felhasználó szerepköre (role) alapértelmezetten "student".
- **Kódrészlet:**

```
const passwordHash = await bcrypt.hash(password, 10);
const role = "student"
const sql = "INSERT INTO users (email, username, role, password_hash,
profile_pic) VALUES (?, ?, ?, ?, ?)";

const response = await dbQuery(sql, [email, username, role,
passwordHash, profile_pic])
if (response.length === 0) {
  res.status(404).json({ message: "Hibas keres" })
}
else {
  res.status(201).json({ message: "Sikeres regisztracio!" })
}
```

2.5.2.2.2 Frontend

- **Kihívás:** Egy átlátható, felhasználóbarát regisztrációs felület megvalósítása, amely figyelmeztet a hibás vagy hiányos adatokra, vizuálisan visszajelzést ad a jelszó erősségéről, és megfelelően kezeli az állapotokat (pl. betöltés, siker, hiba). Emellett biztosítani kellett a jelszó láthatóságának váltását, illetve egy egyszerű módot a belépési felületre való átváltásra.
- **Megoldás:** A regisztráció egy modal ablakban jelenik meg, ahol a felhasználó megadhatja a username, email, password, és confirm password mezőket. A mezők valós idejű validációval vannak ellátva. A jelszó erősségét vizuális indikátor mutatja. A regisztráció gombra kattintva axios segítségével POST-kérés indul a backend felé. Sikeres regisztráció esetén a felhasználó automatikusan átirányításra kerül a főoldalra. Hiba esetén figyelmeztető üzenet jelenik meg.
- **Kódrészlet:**
 - **Jelszó erősségi indikátor:**

```
const PasswordStrengthIndicator = ({ password }) => {
  const getStrength = () => {
    if (!password) return 0;
```

```
let strength = 0;

// Length check
if (password.length >= 8) strength += 1;
if (password.length >= 12) strength += 1;

// Complexity checks
if (/[A-Z]/.test(password)) strength += 1;
if (/[0-9]/.test(password)) strength += 1;
if (/^[A-Za-z0-9]/.test(password)) strength += 1;

return Math.min(strength, 5);
};

const strength = getStrength();
const strengthText = ['Very Weak', 'Weak', 'Moderate', 'Strong', 'Very Strong'][strength - 1] || '';
const color = ['error', 'error', 'warning', 'success', 'success'][strength - 1] || '';
```

- **Validáció:**

```
const validateForm = () => {
  let valid = true;
  const newErrors = {
    username: '',
    email: '',
    password: '',
    checkPassword: '',
  };

  if (!formData.username) {
    newErrors.username = 'Username is required';
    valid = false;
  }

  if (!formData.email) {
    newErrors.email = 'Email is required';
    valid = false;
  }

  if (!formData.password) {
    newErrors.password = 'Password is required';
    valid = false;
  }

  if (formData.password !== formData.checkPassword) {
    newErrors.checkPassword = 'Passwords do not match';
    valid = false;
  }
}
```

```
}

setFormErrors(newErrors);
return valid;
};
```

2.5.2.3. Űrlap létrehozás

2.5.2.3.1 Backend

- **Kihívás:** Szükség volt egy olyan funkcionalitásra, amely lehetővé teszi a felhasználók számára, hogy dinamikusan hozhassanak létre űrlapokat, amelyek több kérdést és azokon belül válaszlehetőségeket is tartalmaznak. A kérdések különböző típusúak lehetnek, és pontérték is társulhat hozzájuk. Fontos volt, hogy minden egyes új űrlap automatikusan a bejelentkezett felhasználóhoz kapcsolódjon.
- **Megoldás:** A bejelentkezett felhasználó azonosítóját (`req.user.id`) felhasználva rögzítjük az űrlap nevét az adatbázisban. Ezt követően a megadott kérdések egyenként kerülnek rögzítésre a kapcsolódó válaszlehetőségeikkel együtt. A server oldalon biztosítottuk, hogy a beküldött adat szerkezete helyes legyen, és csak akkor történjen mentés, ha minden szükséges mező (űrlap neve, kérdések tömbje stb.) rendelkezésre áll.
- **Kódrészlet:**

```
const formResult = await dbQuery("INSERT INTO forms (name, creator_id) VALUES  
(?, ?)", [name, req.user.id]);  
const formId = formResult.insertId;  
  
for (const question of questions) {  
  const { text, type, score, answers } = question;  
  
  const questionResult = await dbQuery(  
    "INSERT INTO questions (text, type, form_id, score) VALUES  
(?, ?, ?, ?)",  
    [text, type, formId, score]  
  );  
  const questionId = questionResult.insertId;  
  
  for (const answer of answers) {  
    const { text, is_right } = answer;  
    await dbQuery(  
      "INSERT INTO answer_options (question_id, text,  
is_right) VALUES (?, ?, ?)",  
      [questionId, text, is_right]  
    );  
  }  
}
```

```
db.commit((err) => {
  if (err) {
    return db.rollback(() => {
      console.error("Commit hiba:", err);
      res.status(500).json({ message: "Szerverhiba!" });
    });
  }
  res.status(201).json({ message: "Űrlap sikeresen létrehozva!",
formId });
});
```

2.5.2.3.2 Frontend

- **Kihívás:** Szükség volt egy intuitív és dinamikus felületi megoldásra, amellyel a felhasználók könnyen létrehozhatnak űrlapokat, amelyek többféle típusú kérdést (egyszeres vagy többszörös választás) és válaszlehetőségeket tartalmaznak. Az űrlaphoz pontérték is rendelhető minden kérdéshez.
- **Megoldás:**
 - A React-re épülő felületen a felhasználó megadhatja az űrlap nevét, hozzáadhat tetszőleges számú kérdést, és azokon belül válaszlehetőségeket. A felület biztosítja a valós idejű validációt: nem engedi elmenteni az űrlapot, ha hiányzik az űrlap neve, nincs kérdés megadva, vagy bármely kérdés hibás (pl. nincs szöveg, nincs válasz, üres válaszopció, nincs megjelölve helyes válasz).
 - **Funkcionalitás:**
 - Új kérdés hozzáadása, törlése.
 - Kérdés típusának választása (egyszeres/többszörös választás).
 - Kérdés szövegének, pontértékének megadása.
 - Válaszlehetőségek kezelése: szöveg, helyes válasz jelölése.
 - Felhasználói visszajelzések snackbar komponenssel (sikeres mentés, hibák pl.: nincsen válaszlehetőség a kérdésre, nincsen szövege a kérdés mezőnek, nincsen a feladatlapnak neve).
 - Automatikus görgetés az új kérdéshez.
 - **Mentés:** A „Mentés” gombra kattintva a rendszer JSON formátumban összegyűjti az űrlap adatait, és API-hívással továbbítja a szerver felé. A felhasználói jogosultságokat access token biztosítja a kérés fejlécében.
- **Kódrészlet:**
 - **Form validálása:**


```

if (!formName.trim()) {
  showPopup("The form needs a name!", "warning");
  return;
}
if (questions.length === 0) {
  showPopup("You need to add at least one question!", "warning");
  return;
}
for (let i = 0; i < questions.length; i++) {
  const q = questions[i];
  if (!q.text.trim()) {
    showPopup(`Question ${i + 1} is missing text!`, "warning");
    return;
  }
  if (!q.answers || q.answers.length === 0) {
    showPopup(`Question ${i + 1} has no answers!`, "warning");
    return;
  }
  const emptyAnswers = q.answers.filter(a => !a.text.trim());
  if (emptyAnswers.length > 0) {
    showPopup(`Question ${i + 1} has empty answer fields!`, "warning");
    return;
  }
}
}

```

- **Kérdés hozzáadása:**

```

const handleAddQuestion = () => {
  setQuestions(prev => [
    ...prev,
    {
      text: "",
      type: "radiobutton",
      score: 0,
      isMultiple: false,
      answers: [{ text: "", isCorrect: false }]
    }
  ]);
};

<Box sx={{ display: 'flex', justifyContent: 'center', mt: { xs: 4, md: 6 } }}>
  <Button
    onClick={handleAddQuestion}
    variant="contained"
    color="success"
    startIcon={<AddCircleOutlineIcon />}
    sx={{
      px: { xs: 4, md: 6 },
      py: { xs: 1.5, md: 2 },
    }}
  />
</Box>

```

```
fontWeight: 'bold',
textTransform: 'none',
borderRadius: 2,
boxShadow: 3,
'&:hover': {
  backgroundColor: 'success.dark',
},
}},
>
  Add Question
</Button>
</Box>
```

2.5.2.4. Űrlap szerkesztése

2.5.2.4.1 Backend

- **Kihívás:** A meglévő űrlapok módosításához olyan funkcionalitásra volt szükség, amely lehetővé teszi a felhasználó számára az űrlap teljes tartalmának (kérdések, válaszok) lekérését, majd ezek szerkesztését. A kérdések és válaszaik dinamikusan változhatnak, új kérdések adhatók hozzá, régiek frissíthetők vagy törölhetők, így a módosítás összetett adatkezelést igényel.
- **Megoldás:**
 - **Lekérdezés:**
 - A /get-all végpont segítségével a rendszer lekéri az adott űrlaphoz tartozó összes kérdést és válaszlehetőséget. Az adatok strukturált JSON formában érkeznek vissza, így a frontend könnyen meg tudja jeleníteni azokat a szerkesztőfelületen.
 - **Frissítés:**
 - A /update-form/:id végpont fogadja az új űrlapnevet, valamint a teljes kérdéslistát a frissített válaszokkal együtt. A már meglévő kérdéseket frissítjük, míg az újonnan létrehozott kérdéseket beszúrjuk. A kérdésekhez tartozó válaszlehetőségek minden esetben újra eltárolásra kerülnek, a régieket előtte töröljük.
- **Kódrészlet:**

```
for (const question of questions) {
  const { id: questionId, text, type, score, answers } =
question;

  let currentQuestionId = questionId;
```

```
        if (currentQuestionId) {
            await dbQuery(
                "UPDATE questions SET text = ?, type = ?, score = ?
WHERE id = ? AND form_id = ?",
                [text, type, score, currentQuestionId, formId]
            );

            await dbQuery("DELETE FROM answer_options WHERE
question_id = ?", [currentQuestionId]);
        } else {
            const questionResult = await dbQuery(
                "INSERT INTO questions (text, type, form_id, score)
VALUES (?, ?, ?, ?)",
                [text, type, formId, score]
            );
            currentQuestionId = questionResult.insertId;
        }

        for (const answer of answers) {
            const { text, is_right } = answer;
            await dbQuery(
                "INSERT INTO answer_options (question_id, text,
is_right) VALUES (?, ?, ?)",
                [currentQuestionId, text, is_right]
            );
        }
    }
}
```

2.5.2.4.2 Frontend

- **Kihívás:** A felhasználónak intuitív és dinamikus szerkesztőfelületre volt szüksége, amely lehetővé teszi a már meglévő űrlapok betöltését, kérdések és válaszok szerkesztését, új kérdések hozzáadását, valamint az űrlap mentését. A válaszlehetőségek típusától (egyválasztós/többválasztós) függően eltérő logikát kellett alkalmazni.
- **Megoldás:**
 - **Lekérdezés:**
 - A useEffect segítségével a komponens betöltéskor meghívja a /form/get-all végpontot, amely az adott űrlap minden adatát (név, kérdések, válaszok) visszaküldi. A válaszokat isCorrect mezővel egészítjük ki, és a isMultiple érték kiszámításával megállapítjuk, hogy a kérdés több helyes választ tartalmaz-e.
 - **Szerkesztés:**
 - A felhasználó szerkesztheti a kérdések szövegét, típusát (radiobutton vagy checkbox), pontszámát, valamint a hozzájuk tartozó válaszokat. A

DefaultFormFormat komponens segítségével minden kérdéshez külön szerkesztőfelület jelenik meg.

- Lehetőség van új kérdés hozzáadására (Add Question), valamint meglévő kérdések törlésére is.
- **Frissítés:**
 - A Save Form gombra kattintva validáljuk az adatokat (pl. üres kérdésszöveg, üres válasz), majd egy strukturált payload segítségével PUT kérésként küldjük el az adatokat a /form/update-form/:id végpontra.
 - A payload tartalmazza az űrlap nevét, valamint a kérdések listáját a frissített válaszokkal. A válaszok isCorrect mezője is_right kulcsa alakul.
- **Kódrészlet:**

```
const processedQuestions = res.data.questions.map((q) => ({
  ...q,
  answers: q.answers.map((a) => ({
    ...a,
    isCorrect: a.is_right,
  })),
  isMultiple: q.answers.filter((a) => a.is_right).length > 1,
}));

setFormData({ ...res.data, questions: processedQuestions });
setName(res.data.name);

.....

useEffect(() => {
  if (bottomRef.current) {
    bottomRef.current.scrollToView({ behavior: 'smooth' });
  }
}, [formData?.questions?.length]);

.....

const showPopup = (message, severity = 'info') => {
  setPopup({ open: true, severity, message });
};

const handleClosePopup = () => {
  setPopup({ ...popup, open: false });
};

const saveQuestionAttribute = (index, key, value) => {
  const updatedQuestions = [...formData.questions];
  updatedQuestions[index] = { ...updatedQuestions[index], [key]: value };
};
```

```

    setFormData({ ...formData, questions: updatedQuestions });
  };

  const deleteQuestion = (index) => {
    const updatedQuestions = formData.questions.filter( (_, i) => i !== index );
    setFormData({ ...formData, questions: updatedQuestions });
  };
  .....
const handleSaveForm = async () => {
  if (!formName.trim()) {
    showPopup('The form needs a name!', 'warning');
    return;
  }

  if (!formData.questions || formData.questions.length === 0) {
    showPopup('You need to add at least one question!', 'warning');
    return;
  }

  for (let i = 0; i < formData.questions.length; i++) {
    const q = formData.questions[i];
    if (!q.text.trim()) {
      showPopup(`Question ${i + 1} is missing text!`, 'warning');
      return;
    }
    if (!q.answers || q.answers.length === 0) {
      showPopup(`Question ${i + 1} has no answers!`, 'warning');
      return;
    }
    const emptyAnswers = q.answers.filter(a => !a.text.trim());
    if (emptyAnswers.length > 0) {
      showPopup(`Question ${i + 1} has empty answer fields!`, 'warning');
      return;
    }
  }

  const payload = {
    name: formName,
    questions: formData.questions.map(q => ({
      id: q.id,
      text: q.text,
      type: q.isMultiple ? 'checkbox' : 'radiobutton',
      score: Number(q.score),
      answers: q.answers.map(a => ({
        text: a.text,
        is_right: a.isCorrect
      })))
    })))
  });

```

```
};  
.....  
{formData.questions.map((question, index) => (  
  <Box key={question.id ?? `new-${index}`} sx={{ mb: { xs: 4, md: 5 }  
}}>  
  <DefaultFormFormat  
    index={index}  
    question={question}  
    saveQuestionAttribute={saveQuestionAttribute}  
    deleteQuestion={deleteQuestion}  
  />  
</Box>  
))}
```

2.5.2.5. Űrlap kitöltése

2.5.2.5.1 Backend

- **Kihívás:** A felhasználók számára lehetővé kellett tenni egy adott űrlap kérdéseinek lekérdezését és az arra adott válaszok beküldését úgy, hogy minden beküldés megfelelően naplózásra kerüljön, és a válaszok mentése biztonságos és egyértelmű módon történjen.
- **Megoldás:**
 - A kitöltés két fő végpontra oszlik:
 - Űrlap lekérése kitöltéshez
 - Végpont: GET /get-form/:id
 - A végpont visszaadja az űrlap nevét, készítőjét, státuszát, valamint a kapcsolódó kérdéseket és válaszlehetőségeket.
 - A válaszban minden kérdés tartalmazza a válaszlehetőségek szövegeit és azonosítóit (de nem tartalmaz értékelési információt).

- **Kódrészlet:**

```
for (let question of questions) {
    const answers = await dbQuery(
        "SELECT id, text FROM answer_options WHERE question_id = ?",
        [question.id]
    );
    question.answers = answers;
}

res.json({
    form: {
        id: form[0].id,
        name: form[0].name,
        creator_id: form[0].creator_id,
        questions: questions
    }
});
```

- **Űrlap válaszainak beküldése**
 - Végpont: POST /submit
 - A végpont fogadja a beküldött válaszok azonosítóit, valamint a beküldő felhasználó és az űrlap azonosítóját.
 - A rendszer minden beküldést naplóz a submissions táblában, és menti a válaszokat a submission_answers táblába.
 - A beküldés részeként kiszámításra kerül a kitöltés összpontszáma is, amely a későbbi értékelés alapját képezi, ennek részletes működése külön részben kerül bemutatásra.

- **Kódrészlet:**

```
const submissionResult = await dbQuery(
    "INSERT INTO submissions (user_id, form_id, submit_time, total_score) VALUES (?, ?, NOW(), ?)",
    [user_id, form_id, totalScore]
);
const submissionId = submissionResult.insertId;

for (const answer of answers) {
    await dbQuery(
        "INSERT INTO submission_answers (submission_id, answer_id) VALUES (?, ?)",
        [submissionId, answer]
    );
}
```

```

        db.commit((err) => {
            if (err) {
                return db.rollback(() => {
                    console.error("Commit hiba:", err);
                    res.status(500).json({ message: "Szerverhiba a mentés
során!" });
                });
            }

            res.status(201).json({ message: "Válaszok sikeresen
elmentve!", submissionId });
        });

```

2.5.2.5.2 Frontend

- **Kihívás:** A frontendnek lehetővé kell tennie a felhasználók számára, hogy kitöltsék az űrlapokat, válaszokat adjanak a kérdésekre, és elküldjék azokat. Emellett megfelelő hibakezelést és felhasználói visszajelzést is biztosítani kell a kitöltési folyamat során.
- **Megoldás:** A frontend rész biztosítja az űrlap kérdéseinek megjelenítését és a válaszok gyűjtését. A felhasználók válaszait kezelhetjük a kérdések típusai (pl. checkbox, radio) alapján. A beküldés előtt a rendszer ellenőrzi, hogy legalább egy választ megadtak-e. Végül a válaszok elküldése egy POST kéréssel történik a backendhez.
- **Kódrészlet:**
- **Űrlap adatainak lekérése:**

```

useEffect(() => {
    axios.get(`/form/get-form/${id}`, {
        headers: {
            Authorization: `Bearer ${accessToken}`
        }
    })
    .then(res => {
        setForm(res.data.form);
        setLoading(false);
    })
    .catch(err => {
        console.error('Error fetching form:', err);
        setLoading(false);
    });
}, [id]);

```

- **Válaszok megváltoztatásának kezelése:**

```

const handleAnswerChange = (questionId, answerId, isMultiple) => {
    setSelectedAnswers(prev => {
        if (isMultiple) {
            const current = prev[questionId] || [];

```



```

        const updated = current.includes(answerId)
          ? current.filter(id => id !== answerId)
          : [...current, answerId];
        return { ...prev, [questionId]: updated };
      } else {
        return { ...prev, [questionId]: [answerId] };
      }
    });
  });
};

```

- **Válaszok mentése és feladatlap beküldése:**

```

const handleSubmit = async () => {
  const allSelected = Object.values(selectedAnswers).flat();

  if (allSelected.length === 0) {
    alert("Please select at least one answer before submitting.");
    return;
  }

  try {
    const response = await axios.post("/form/submit", {
      form_id: id,
      user_id: user.id,
      answers: allSelected
    }, {
      headers: {
        Authorization: `Bearer ${accessToken}`
      }
    });

    setSubmitStatus({ type: "success", message: response.data.message });
  } catch (err) {
    const message = err.response?.data?.message || "Submission failed.";
    setSubmitStatus({ type: "error", message });
  }
  navigate(`/user/form/view/${id}`)
};

```

2.5.2.6. Űrlap kiküldése és e-mail-es értesítő

2.5.2.6.1 Backend

- **Kihívás:** Szükség volt arra, hogy egy űrlapot a rendszer automatikusan minden tanulónak kiküldjön, de ne küldje ki többször ugyanannak a felhasználónak. Ezenfelül fontos volt, hogy a kiküldésről a tanulók e-mailes értesítést is kapjanak.
- **Megoldás:**

- A megvalósítás két egymásra épülő részből áll:
 1. Kiküldés adatbázisba mentése
 - Végpont: POST /send-to-students
 - A szerver lekérdezi az összes "student" szerepkörű felhasználót.
 - Minden egyes tanulóhoz ellenőrzi, hogy az adott űrlapot már korábban kiküldték-e számára (sent_forms táblán keresztül).
 - Csak azoknak menti új sorral a kiküldést (sent_forms) akiknél még nem történt ilyen.
 - Minden mentés tartalmazza a form_id, user_id és sent_at mezőket.

- **Kódrészlet:**

```
for (const student of students) {
    const alreadySent = await dbQuery(
        "SELECT id FROM sent_forms WHERE user_id = ? AND form_id = ?",
        [student.id, form_id]
    );

    if (alreadySent.length === 0) {
        await dbQuery(
            "INSERT INTO sent_forms (user_id, form_id, sent_at) VALUES
            (?, ?, ?)",
            [student.id, form_id, now]
        );
    }
    await notifyUser(form_id);
    res.json({
        message: `A form (${formData.name}) sikeresen kiküldve
        ${students.length} tanulóknak.`,
    });
}
```

1. E-mailes értesítés küldése

- A kiküldés után meghívásra kerül a notifyUser(form_id) függvény.
- Ez a függvény a utils mappában található, és a nodemailer csomag segítségével küld e-maileket.
- Lekérdezi a kiküldött űrlap nevét és azon tanulók e-mailjeit, akikhez az űrlap eljutott.
- Minden címzett részére személyre szabott e-mail kerül kiküldésre HTML formátumban.

- A levelek tartalmazzák az űrlap nevét amit a diáknak ki kell töltenie és arra hívja fel a figyelmet, hogy új űrlap vár a kitöltésre.

- **Kódrészlet:**

```
const transporter = nodemailer.createTransport({
  service: 'gmail',
  auth: {
    user: process.env.MAIL_USER,
    pass: process.env.MAIL_PASS
  }
});

for (const student of students) {
  const subject = `Új űrlap érkezett: ${formName}`;
  const text = `Kedves Diák!<br><br>
    Új űrlap érkezett számodra a Pitchfork Forms rendszerén
    keresztül: <strong>${formName}</strong>.<br>
    Kérjük, lépj be a rendszerbe, és töltsd ki a hozzád rendelt
    űrlapot.<br><br>
    Üdvözlettel,<br>
    Pitchfork Forms csapat`;

  await transporter.sendMail({
    from: '"Pitchfork Forms" <noreply@pitchforkforms.eu>',
    to: student.email,
    subject: subject,
    html: text,
  });
}
```

2.5.2.7. Kiküldött űrlapok kilistázása

2.5.2.7.1 Backend

- **Kihívás:** A felhasználóknak szükségük volt arra, hogy könnyen áttekinthessék, milyen űrlapokat küldtek már ki nekik, de amelyeket még nem töltöttek ki.
- **Megoldás:** A megoldás egy GET típusú végpont segítségével történik:
 - Végpont: GET /pending/:userId
 - A szerver lekérdezi azokat az űrlapokat, amelyeket az adott userId-hez tartozó felhasználónak már kiküldtek (sent_forms), de még nem töltötte ki őket (submissions táblában nincs rekord az adott form_id és user_id párosra).
 - A válaszban az ilyen függő űrlapok listája szerepel: id, name, creator_id.

- **Kódrészlet:**

```
const pendingForms = await dbQuery(`
  SELECT f.id, f.name, f.creator_id
  FROM sent_forms sf
  JOIN forms f ON sf.form_id = f.id
  WHERE sf.user_id = ?
  AND f.id NOT IN (
    SELECT form_id FROM submissions WHERE user_id = ?
  )
`,
  [userId, userId]
);
res.json({ forms: pendingForms });
```

2.5.2.7.2 Frontend

- **Kihívás:** A frontend oldalnak képesnek kell lennie a tanuló számára megjeleníteni a számára függő űrlapokat, amelyeket már kiküldtek, de még nem töltött ki.
- **Megoldás:** A megoldás az volt, hogy a frontendben egy GET kérés történt a backendhez a felhasználó id-jával, hogy lekérje a számára függő űrlapokat, és ezeket megjelenítse egy listában. A válasz a lekért űrlapok listáját tartalmazza, és ha nincsenek függő űrlapok, akkor egy üzenet jelenik meg.
- **Kódrészlet:**

```
const response = await axios.get(`/user/pending/${user.id}`, {
  headers: {
    Authorization: `Bearer ${accessToken}`,
  },
});

<Grid container spacing={3}>
  {forms.map((form) => (
    <Grid item xs={12} sm={6} md={4} lg={3} key={form.id} margin={6}>
      <FormCard formName={form.name} formId={form.id} />
    </Grid>
  ))}
</Grid>
```

2.5.2.8. Kitöltött űrlapok kilistázása

2.5.2.8.1 Backend

- **Kihívás:** A rendszernek biztosítania kellett, hogy a felhasználók visszakereshessék, mely űrlapokat töltötték már ki, és lássák azokhoz kapcsolódó alapvető értékelési adatokat (pontszám, beküldés ideje stb.).

- **Megoldás:** Ez a funkció egy GET típusú végponton keresztül valósul meg:
 - Végpont: GET /forms/completed/:userId
 - A lekérdezés visszaad minden űrlapot, amelyet a megadott userId-hoz tartozó felhasználó már beküldött (megjelenik a submissions táblában).
 - A válaszban szerepel:
 - az űrlap neve, azonosítója és készítője (form_id, name, creator_id),
 - beküldés időpontja (submit_time),
 - a felhasználó által elért pontszám (total_score),
 - az adott űrlapon elérhető maximális pontszám (max_score).
- **Kódrészlet:**

```
const completedForms = await dbQuery(  
  `SELECT  
    f.id AS form_id,  
    f.name,  
    f.creator_id,  
    s.submit_time,  
    s.total_score,  
    COALESCE(qs.max_score, 0) AS max_score  
  FROM submissions s  
  JOIN forms f ON s.form_id = f.id  
  LEFT JOIN (  
    SELECT form_id, SUM(score) AS max_score  
    FROM questions  
    GROUP BY form_id  
  ) qs ON f.id = qs.form_id  
  WHERE s.user_id = ?  
  `,  
  [userId]  
);  
res.json({ forms: completedForms });
```

2.5.2.8.2 Frontend

- **Kihívás:** A frontendnek képesnek kell lennie arra, hogy megjelenítse a felhasználó által kitöltött űrlapokat és azokhoz kapcsolódó információkat, például a pontszámot és a beküldés időpontját.
- **Megoldás:** A megoldás az volt, hogy a frontend egy GET kérést küldött a backendnek a kitöltött űrlapok adatainak lekérésére. A válaszban szereplő adatokat egy táblázatban jeleníti meg, amely tartalmazza az űrlap nevét, státuszát, beküldés időpontját és a pontszámokat.

- **Kódrészlet:**

```
{completedForms.length === 0 ? (  
  <Typography variant="body1" textAlign="center">  
    No completed forms yet.  
  </Typography>  
): (  
  <Grid container spacing={3} justifyContent="center">  
    {completedForms.map((form) => (  
      <Grid item key={form.form_id}>  
        <CompletedFormCard  
          formName={form.name}  
          formId={form.form_id}  
          totalScore={form.total_score}  
          maxScore={form.max_score}  
          submitTime={form.submit_time}  
        />  
      </Grid>  
    ))}  
  </Grid>  
)}
```

2.5.2.9. Diák megtekintheti a saját válaszait a már kitöltött űrlapokon

2.5.2.9.1 Backend

- **Kihívás:** A diákok számára biztosítani kell, hogy meg tudják tekinteni a már kitöltött űrlapjaikat, nemcsak a kérdéseket és válaszlehetőségeket, hanem azt is, hogy mely válaszokat választották ki, illetve melyek voltak helyesek. A lekérdezésnek egyszerre kell hatékonyan és részletesen működnie.
- **Megoldás:** Az endpoint egy összetett SQL-lekérdezést használ, amely:
 - lekéri az adott formhoz tartozó összes kérdést és válaszlehetőséget,
 - megjelöli, hogy mely válaszlehetőségeket jelölte be a diák (subquery a submission_answers táblára),
 - és azt is, hogy melyik válasz a helyes (is_right mező alapján).
- A visszakapott adatokból a backend logika struktúrált JSON-t épít, ahol minden kérdéshez listázza:
 - a válaszlehetőségek szövegét,
 - hogy helyes-e (is_right_answer),
 - és hogy a diák kiválasztotta-e (is_answer_selected).
- Ezáltal a diák részletesen láthatja, mit választott, és mi lett volna a helyes, ami később értékeléshez, tanuláshoz vagy visszajelzéshez is használható.

- **Kódrészlet:**

```
const questions = [];
  completedFormDetails.forEach((row) => {
    let question = questions.find((q) => q.question_id ===
row.question_id);

    if (!question) {
      question = {
        question_id: row.question_id,
        question_text: row.question_text,
        answer_options: [],
      };
      questions.push(question);
    }

    question.answer_options.push({
      answer_option_id: row.answer_option_id,
      answer_option_text: row.answer_option_text,
      is_right_answer: row.is_right_answer,
      is_answer_selected: row.is_answer_selected,
    });
  });

res.json({ formDetails: questions });
```

2.5.2.9.2 Frontend

- **Kihívás:** A frontendnek világosan és felhasználóbarát módon kellett megjelenítenie a kitöltött űrlap részleteit. Fontos szempont volt a vizuális visszacsatolás: a tanuló egyértelműen lássa, mely válaszokat jelölte be, és azok helyesek voltak-e.
- **Megoldás:** A React-alapú oldal lekéri az aktuális felhasználó adott űrlaphoz tartozó válaszait, és két szekcióra bontva jeleníti meg azokat:
- **Your Answer(s):** A diák által kiválasztott válaszlehetőségek, színkóddal jelezve, hogy azok helyesek vagy hibásak voltak.
- **Correct Answer(s):** Az összes helyes válaszlehetőség, függetlenül attól, hogy a diák kijelölte-e.

- **Kódrészlet:**

```

{formDetails.map((question) => (
    <Paper key={question.question_id} sx={{ p: { xs: 2, md: 3 },
mb: 4 }}>

        <Typography variant="h6" gutterBottom>
            {question.question_text}
        </Typography>

        <Grid container spacing={2} mt={2}>
            <Grid item xs={12} md={6}>
                <Typography variant="subtitle1" gutterBottom>
                    Your Answer(s)
                </Typography>
                {question.answer_options
                    .filter((opt) => opt.is_answer_selected)
                    .map((answerOption) => (
                        <Chip
                            key={answerOption.answer_option_id}
                            label={answerOption.answer_option_text

                                color={answerOption.is_right_answer ?

                                    variant="outlined"
                                    sx={{ mr: 1, mb: 1 }}
                                />
                            )))
                </Grid>

            <Grid item xs={12} md={6}>
                <Typography variant="subtitle1" gutterBottom>
                    Correct Answer(s)
                </Typography>
                {question.answer_options
                    .filter((opt) => opt.is_right_answer)
                    .map((correctOption) => (
                        <Chip
                            key={correctOption.answer_option_id}
                            label={correctOption.answer_option_text

                                color="success"
                                sx={{ mr: 1, mb: 1 }}
                            />
                        )))
                </Grid>
            </Grid>
        </Paper>
    )
)}

```


2.5.3.Algoritmusok

Az oldal tökéletes működése érdekében alkalmaznunk kellett segédfüggvényeket.

- **QueryHelper.js**

- A fájl tartalmaz egy segédfüggvényt az SQL lekérdezések megkönnyítésére, illetve a kód leegyszerűsítésére és az átláthatóság növelésére.
- Forráskód:

```
const db = require("../config/database");

// Segédfüggvény az SQL lekérdezéshez
const dbQuery = (sql, params = []) => {
  return new Promise((resolve, reject) => {
    db.query(sql, params, (err, results) => {
      if (err) reject(err);
      else resolve(results);
    });
  });
};

module.exports = dbQuery
```

- **deleteSentFormsByUserId**

- Ez a segédfüggvény a userActions.js -ben található, feladata egy felhasználó kitörlésekor az összes neki kiküldött űrlap törlése a (sent_forms) táblából.
- Forráskód:

```
const deleteSentFormsByUserId = async (userId) => {
  const sql = "DELETE FROM sent_forms WHERE user_id = ?";
  const result = await dbQuery(sql, [userId]);
}
```

- **Automatikus pontszámítás**

- A rendszernek automatikusan ki kell értékelnie a diákok által beküldött válaszokat, mégpedig úgy, hogy figyelembe veszi:
 - a kérdés típusát (egyszeres vagy többszörös választás),
 - a helyes válasz(ok) meglétét,
 - a kérdéshez tartozó pontszámokat,
 - és ezek alapján igazságosan kiszámítja az elérhető összpontszámot.
- **Fontos, hogy:**
 - Radiobutton típusnál csak egy válasz lehet helyes.

- Checkbox típusnál több válasz is helyes lehet, és részpontszámot kell adni minden helyes válaszáért.
- A `total_Score` függvény minden beküldött válasz (`answer_option ID`) alapján külön-külön lekérdezi a hozzá tartozó kérdést és annak típusát. Ezután:
 - **Radiobutton esetén:**
 - Ha a válasz helyes (`is_right = true`), akkor a teljes pontszámot hozzáadja az összpontszámhoz.
 - **Checkbox esetén:**
 - Csak a helyes válaszokra ad részpontot.
 - Lekérdezi, hány helyes válasz tartozik a kérdéshez.
 - A kérdés teljes pontszámát elosztja ennyi felé, és ennek egy részét adja hozzá minden helyes válasznál.
 - **Így biztosított, hogy:**
 - a teljes pont csak akkor jár, ha a diák minden helyes választ bejelöl checkboxnál,
 - hibás válasz nem von le pontot, de nem is ad hozzá.
 - Az így kapott összpontszám (`totalScore`) az adatbázisba kerül, amikor a diák beküldi az űrlapot.
- **Kódrészlet:**

```
const total_Score = async (answers) => {
  let totalScore = 0;
  for (const answerId of answers) {
    const questionData = await dbQuery(`
      SELECT q.id as question_id, q.type, q.score, ao.is_right
      FROM questions q
      INNER JOIN answer_options ao ON q.id = ao.question_id
      WHERE ao.id = ?
    `, [answerId]);

    if (questionData.length === 0) continue;

    const { question_id, type, score, is_right } = questionData[0];

    if (type === 'radiobutton') {
      if (is_right) {
        totalScore += score;
      }
    } else if (type === 'checkbox') {
      if (is_right) {
```

```
const rightCountData = await dbQuery(`
  SELECT COUNT(*) AS right_count
  FROM answer_options
  WHERE question_id = ? AND is_right = TRUE
`, [question_id]);

const rightCount = rightCountData[0].right_count;
if (rightCount > 0) {
  totalScore += score / rightCount;
}
}
}
return totalScore;
};
```

2.6. Tesztelés

2.6.1. Backend tesztelés

A backend tesztelés célja az volt, hogy ellenőrizzük, hogy az alkalmazás logikája (például adatbázisműveletek, hitelesítés, hibakezelés) helyesen működik. Segített megelőzni a hibákat, gyorsította a fejlesztést, és biztosította, hogy a változtatások ne rontsák el a meglévő funkciókat.

2.6.1.1. User route tesztek

- POST /auth/register: Regisztráció
 - Cél: Új felhasználó regisztrálása.
 - Tesztelt esetek:
 - Sikeres regisztráció: Helyes adatokkal regisztrál, a jelszó bcrypt-tel hash-elésre kerül.
 - Hiányzó mezők: Kötelező adatok (pl. email) hiányában hibát dob.
 - Adatbázis hiba: Sikertelen adatbázis művelet esetén fellépő hiba kezelése.
- POST /auth/login: Belépés
 - Cél: Meglévő felhasználó bejelentkeztetése és token generálása.
 - Tesztelt esetek:
 - Hiányzó mezők: Email vagy jelszó kihagyása.
 - Nem létező felhasználó: Email alapján nem található felhasználó.
 - Sikeres bejelentkezés: Tokenek generálása és refresh cookie beállítása (access és refresh token).
- PUT /user/users/:id
 - Cél: Egy adott felhasználó adatainak módosítása.
 - Tesztelt esetek:
 - Hiányzó kötelező mezők: Email vagy felhasználónév hiánya.
 - Nem létező felhasználó: Ha nincs érintett sor az adatbázisban.
 - Sikeres módosítás: Az adatok módosítása sikeres.

2.6.1.2. Form routes tesztek

- POST /form/save-forms: Űrlap készítés
 - Cél: Új űrlap mentése az adatbázisba

- Tesztelt esetek:
 - Sikeres mentés:
 - Teszteli, hogy ha minden adat helyesen meg van adva (név, kérdések, válaszok), az űrlap mentésre kerül az adatbázisba.
 - Ellenőrzi a tranzakció (`beginTransaction`, `commit`) sikerességét.
 - Ellenőrzi, hogy az adatbázis-lekérdezések (`dbQuery`) megfelelően futnak le.
 - Hiányzó adatok:
 - Ha nincs megadva űrlap név, vagy a kérdések listája, hibaüzenetet dob.
 - Tranzakciós hiba:
 - Ha a `beginTransaction` hibát dob, a szerver "Szerverhiba" üzenetet küld vissza.
- PUT `/form/update-form/:id`: Űrlap módosítás
 - Cél: Egy adott űrlap adatainak módosítása és mentése.
 - Tesztelt esetek:
 - Sikeres módosítás: Frissíti az űrlap nevét, a kérdések és válaszok listáját, illetve megvárja míg a tranzakció sikeresen lezárul.
 - Hibás kérés: Ha a kérés nem tartalmaz új űrlapnevet vagy kérdéseket, illetve válaszokat hibát dob.
 - Tranzakciós hiba + rollback: Ha a `dbQuery` egy adott ponton hibát dob (pl. kérdés frissítése során), akkor a tranzakció rollback-el visszafordul és szerverhibát jelez.

2.6.1.3. Mockok a tesztek során:

- **dbQuery**: Mockolni kellett, hogy ne használja a rendes adatbázist a teszt során.
- **bcryptjs**: jelszó hash-elés/ellenőrzés
- **jsonwebtoken, jsonwebtoken.verify**: token generálás és verifikáció
- **db.query**: közvetlen SQL hívások mockolása (bejelentkezésnél)
- **db.beginTransaction, db.commit, db.rollback**: Mockokkal szimulálják a tranzakciók működését.

2.6.1.4. Tesztkörnyezet:

- **supertest:** HTTP kérések szimulálására.
- **jest.mock:** Külső modulok (pl. bcryptjs, jsonwebtoken, dbQuery) mockolása a kontrollált környezet érdekében.
- Tokenek és hash-elés is mockolásra kerül, hogy ne legyen szükség valódi titkosításra/titkos kulcsokra.

2.6.2. Frontend tesztelés

A frontend tesztelés célja az volt, hogy biztosítsuk az oldalak és komponensek helyes működését, valamint hogy a felhasználói élmény stabil maradjon változtatások után is. A tesztek segítettek megelőzni hibákat, egyszerűsítették a hibakeresést, és hozzájárultak a megbízható felület kialakításához.

A `@testing-library/react` segítségével felhasználóközpontú teszteket írtunk, amelyek valós interakciókat szimulálnak, például gombnyomást, navigációt és API válaszok kezelését.

2.6.2.1. AdminPage tesztek

- Cél: Az adminfelület stabilitásának és funkcionalitásának ellenőrzése (pl. űrlapok listázása, hiba- és üres állapotok kezelése, navigáció).
- Tesztelt esetek:
 - Betöltési állapot:
 - Ellenőrzi, hogy az oldal betöltésekor megjelenik a spinner (töltésjelző).
 - Sikeres adatlekérés:
 - Mockolt API válasz alapján a felület két űrlapot jelenít meg FormCard komponensként.
 - Hibakezelés:
 - Hiba esetén (pl. sikertelen API válasz) megjelenik egy hibaüzenet a felhasználó számára.
 - Üres állapot:
 - Ha nincs még egyetlen létrehozott űrlap sem, tájékoztató szöveg és egy “Create” gomb jelenik meg.
 - Navigáció:
 - A “Create” gombra kattintás átirányít az új űrlap létrehozó oldalra (`/admin/create-form`).

- Mockolt elemek:
 - axios.get: Az API hívások szimulálásához.
 - FormCard, AddIcon: UI elemek, amelyeket nem szükséges teljes funkcionalitásukban tesztelni.

2.6.2.2. *MainPage tesztek*

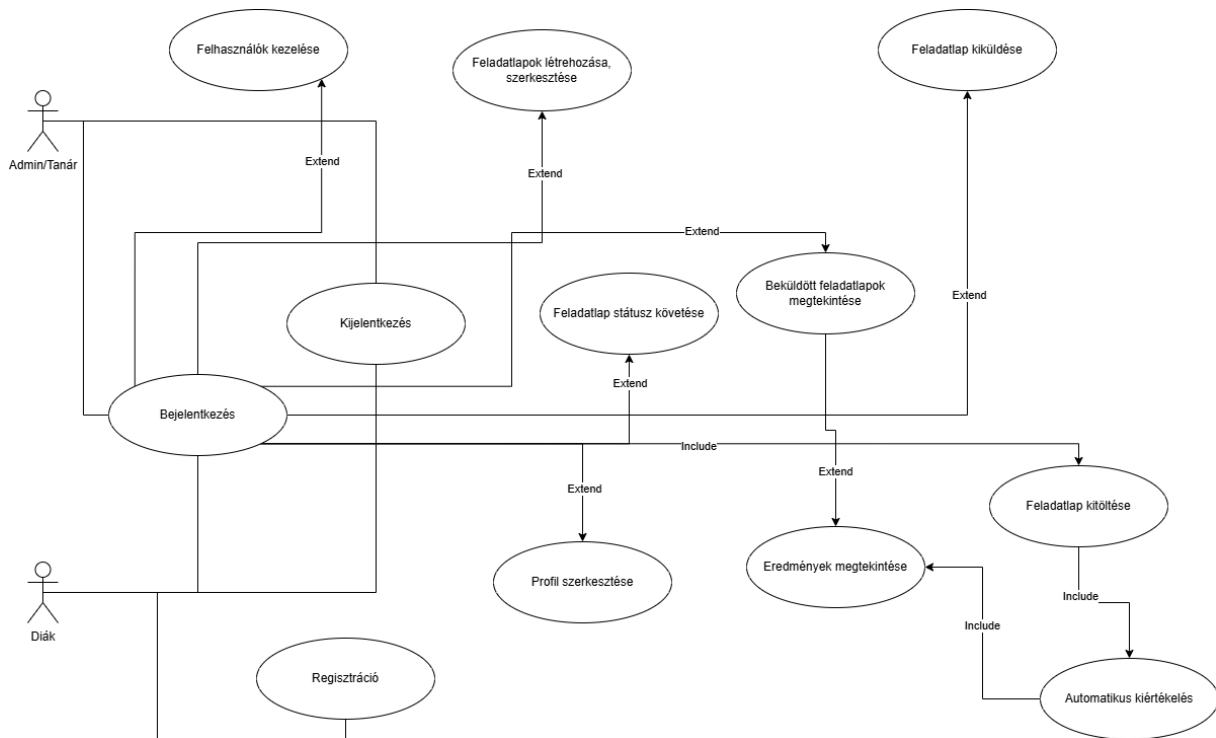
- Cél: A főoldal tartalmainak, valamint a regisztráció/bejelentkezés modal-os interakciók helyes működésének tesztelése.
- Tesztelt esetek:
 - Statikus tartalmak:
 - Az oldal főcíme és leírása helyesen jelenik meg.
 - Carousel komponens:
 - Megfelelően betöltődik a kezdőlapra (mockolt komponenssel).
 - Regisztrációs modal nyitása:
 - A “Get Started” és a “Create Your Free Account” gombokra kattintva megjelenik a regisztrációs modal.
 - Modal bezárása:
 - A regisztrációs ablak bezárható a “Close” gomb segítségével.
 - Mockolt elemek:
 - Carousel, RegisterModal, LoginModal: Mivel ezek külön komponensek, a tesztben csak a jelenlétük és interakcióik kerültek ellenőrzésre.

2.6.2.3. *Tesztkörnyezet*

- Eszközök:
 - @testing-library/react: Komponens- és UI-tesztelés.
 - jest: Tesztfutató és mockolási lehetőségek.
- Mockolások:
 - Külső modulok és komponensek (pl. axios, modalok, ikonok) mockolása, hogy a tesztek gyorsabbak és megbízhatóbbak legyenek.

2.6.3.Felhasználói esetek

2.6.3.1. Use case diagram



- Felhasználók
 - Diák
 - A tanuló, aki feladatlapokat tölt ki, regisztrál és megtekinti az eredményeit.
 - Tanár/Admin
 - Olyan felhasználó, aki jogosult a rendszer adminisztrációjára és feladatlapok kezelésére.
- Funkciók
 - Bejelentkezés
 - A felhasználó (admin/tanár vagy diák) bejelentkezik a rendszerbe.
 - Kijelentkezés
 - A felhasználó kijelentkezik a rendszerből.
 - Regisztráció
 - A diák új fiókot hoz létre a rendszerben.
 - Profil szerkesztése
 - A felhasználó módosíthatja személyes adatait. (pl. név, email)
 - Felhasználók kezelése

- Az admin/tanár hozzáfér a felhasználók listájához, szerkesztheti vagy törölheti őket.
- Feladatlapok létrehozása
 - Az admin/tanár új feladatlapokat készíthet.
- Feladatlapok szerkesztése
 - Az admin/tanár módosíthatja a meglévő feladatlapokat.
- Feladatlap kiküldése
 - Az admin/tanár kiküldi a diákoknak a kitöltendő feladatlapokat.
- Feladatlap státusz követése
 - A tanár nyomon követi, mely diákok küldték be a feladatlapokat és látja a pontszámokat.
- Beküldött feladatlapok megtekintése
 - A diák megtekintheti beküldött feladatlapjait és azok eredményeit.
- Feladatlap kitöltése
 - A diák kitölti a kiadott feladatlapot
 - Include: automatikus kiértékelést von be beküldéskor.
- Automatikus kiértékelés
 - A rendszer automatikusan kiértékeli a beküldött feladatlapot.
- Eredmények megtekintése
 - A diák megtekintheti beküldött feladatlapjainak pontszámát.
- Kapcsolatok
 - Include
 - Kötelező részfolyamat, pl. a bejelentkezés mindig része a rendszer használatnak.
 - Extend
 - Opcionális vagy feltételes bővítés, pl. a diák csak bejelentkezés után láthatja a beküldött feladatlapokat.

2.6.4.Felhasználói történetek: Given-When-Then

2.6.4.1.Tanárok/Adminok felhasználói történetei

- Felhasználók kezelése
 - **Given:** egy bejelentkezett tanár vagy adminisztrátor,

- **When:** megnyitja a felhasználók kezelőfelületét,
- **Then:** láthatja az összes felhasználó listáját, és törölhet egyes felhasználókat.
- Feladatlapok létrehozása és szerkesztése
 - **Given:** egy bejelentkezett tanár vagy adminisztrátor,
 - **When:** új feladatlapot kíván létrehozni vagy egy meglévőt szerkeszt,
 - **Then:** hozzáadhat különböző típusú kérdéseket, szerkesztheti vagy törölheti azokat.
- Feladatlapok kiküldése
 - **Given:** egy elkészült feladatlap,
 - **When:** a tanár/adminisztrátor kiválasztja a címzetteket és elküldi a feladatlapot,
 - **Then:** a kiválasztott diákok megkapják a feladatlapot kitöltésre.
- Kitöltött feladatlapok megtekintése
 - **Given:** egy tanár/adminisztrátor,
 - **When:** megnyitja egy feladatlap eredményeit,
 - **Then:** láthatja a diákok válaszait és azok kiértékelését.
- Automatikus kiértékelés
 - **Given:** egy diák beküldött egy feladatlapot,
 - **When:** az tartalmaz feleletválasztós kérdéseket,
 - **Then:** a rendszer automatikusan kiértékeli ezeket a kérdéseket.
- Feladatlapok státuszának nyomon követése
 - **Given:** egy tanár/adminisztrátor megnyitja egy feladatlap kezelőfelületét,
 - **When:** ellenőrzi a státuszokat,
 - **Then:** láthatja, hogy mely diákok kapták meg és töltötték ki a feladatlapot.

2.6.4.2. Diákok felhasználói történetei

- Feladatlapok kitöltése
 - **Given:** egy bejelentkezett diák,
 - **When:** megnyit egy számára kiküldött feladatlapot,
 - **Then:** kitöltheti azt és beküldheti a rendszerbe.
- Eredmények megtekintése
 - **Given:** egy diák,
 - **When:** megnyitja a korábban beküldött feladatlapját,

- **Then:** láthatja a kiértékelt eredményét és a pontszámát.

2.6.4.3. Általános felhasználói történetek

- Biztonságos hitelesítés
 - **Given:** egy felhasználó be szeretne lépni az alkalmazásba,
 - **When:** megadja a helyes felhasználónevét és jelszavát,
 - **Then:** JWT alapú tokennel hitelesítve lesz, és hozzáférést kap a jogosultságainak megfelelő funkciókhoz.
- Reszponzív dizájn
 - **Given:** egy felhasználó bármilyen eszközről (asztali gép, mobil, tablet) használja az alkalmazást,
 - **When:** megnyitja azt a böngészőben,
 - **Then:** az alkalmazás felülete megfelelően igazodik az eszköz képernyőméretéhez.
- Felhasználói profilok kezelése
 - **Given:** egy bejelentkezett felhasználó,
 - **When:** megnyitja a profilbeállításait,
 - **Then:** szerkesztheti a nevét, email címét és feltölthet profilképet.

2.7. Továbbfejlesztési lehetőségek

A projekt jelenlegi állapotában egy stabil, működőképes feladatlap-kezelő webalkalmazást biztosít a tanár(admin) és diákok számára. A jövőben az alábbi irányokban képzelhető el a rendszer továbbfejlesztése:

- **Tanár role hozzáadása:** Egy tényleges tanári szerepkör hozzáadása ami elkülönül az admin szerepkörétől. Ez lenne az egyik elsődleges változtatás a továbbfejlesztésnél.
- **Diákok csoportosítása:** Minden tanár létre hozhatna csoportokat osztályok alapján vagy foglalkozások alapján és így külön tudna feladatlapokat kiküldeni egyes osztályoknak / diákoknak.
- **Képfeltöltés és multimédiás tartalmak támogatása:** Lehetőség biztosítása képek, videók vagy hanganyagok beágyazására a feladatlapok kérdéseinél, amely tovább növeli az interaktivitást és a kérdéstípusok sokszínűségét.
- **Nyílt kérdések manuális javításának felülete:** A jelenlegi automatikus értékelés mellett a tanárok számára felületet biztosíthatunk az olyan kérdések kézi javítására, amelyeket a rendszer nem tud önállóan kiértékelni (pl. esszék).
- **Statisztikai modul:** Részletesebb statisztikák megjelenítése tanároknak és diákoknak (pl. átlagpontoszám, gyakori hibák, teljesítmény időbeli alakulása), amely segíti a tanítási és tanulási folyamat elemzését.
- **Offline kitölthetőség:** Olyan funkció beépítése, amely lehetővé teszi a feladatlapok letöltését és offline kitöltését, majd későbbi feltöltését.
- **Többnyelvűség támogatása:** A felhasználói felület és a rendszer több nyelvre történő fordítása, amellyel szélesebb körben is használhatóvá válik az alkalmazás.
- **Tananyag csatolási lehetőség:** A tanárok nemcsak feladatlapokat, hanem kapcsolódó tananyagokat (pl. PDF, prezentáció, linkek) is feltölthetnek, ezzel kiegészítve az oktatást.

3. Felhasználói dokumentáció









Ebben a fejezetben a Pitchfork Forms web alkalmazás használatát, követelményeit és a hibák kezelését mutatjuk be.

3.1. Program funkcionalitásainak bemutatása

A Pitchfork Forms egy webalapú alkalmazás, amely lehetővé teszi a tanárok és diákok számára a feladatlapok kezelését. Jelenleg a tanár és az adminisztrátor jelenleg ugyanazt a szerepkört tölti be, **(ez később változni fog)** így az alábbi funkciók mindkét felhasználótípus számára elérhetők:

3.1.1. Tanárok és adminisztrátorok számára

- **Felhasználók kezelése:** Összes meglévő felhasználó megjelenítése és egyes felhasználók törlése.

Students					
Profile	Username	Email	Role	Actions	
	Student	test@example.com	student		VIEW FORMS
	student2	student2@gmail.com	student		VIEW FORMS
	student3	student3@gmail.com	student		VIEW FORMS
	student4	student4@gmail.com	student		VIEW FORMS

- **Feladatlapok létrehozása és szerkesztése:** Különböző típusú kérdéseket tartalmazó feladatlapok létrehozása.

- **Feladatlapok kiküldése:** Feladatlapok diákoknak történő elküldése.
- **Kitöltött feladatlapok megtekintése:** A diákok által beküldött eredmények megtekintése.
- **Automatikus kiértékelés:** Az egyszerű kérdés típusok automatikus javítása (felelet választós kérdések).
- **Feladatlapok státuszának nyomon követése:** Megtekinthető, hogy mely diákok kapták meg a feladatlapokat, és azok kitöltési státusza.

Forms sent out to - Student				BACK
Form Name	Status	Submitted At	Score	
Sample Form A	Completed	2024. 04. 01. 10:00:00	15 / 15	
Sample Form B	Completed	2025. 05. 04. 15:27:27	5 / 5	
teszt	Not Completed	N/A	N/A	

3.1.2. Diákok számára

- **Feladatlapok kitöltése:** A tanárok által kiküldött feladatlapok egyszerű és gyors kitöltése.

több kérdés

1. Kék az ég?

☒ igen ☐ nem ☐ talán

2. hány lába van 2 kutyának?

☒ 8 ☐ 9 ☐ 4 ☐ 6

3. hány kérdés van ebben a feladatlapban

☐ 10 ☒ 9 ☐ 2 ☐ 6

4. melyik állat gyorsabb

☐ tehén ☒ ló

5. ez egy kérdés?

☐ igen ☒ nem

6. ingyen pont

☒ asd ☐ bbb ☒ ccc

7. ez is ingyen van

☐ L ☒ W ☐ C

8. nem jut eszembe több kérdés

☒ kék ☐ pálcikaember

9. ez az utolsó kérdés?

☐ igen ☐ nem ☒ nemtör

SUBMIT FORM

- **Eredmények megtekintése:** A beküldött feladatlapok eredményeinek megtekintése.

Your Answers

Your Answer(s)	Correct Answer(s)
<input type="radio"/> igen	<input checked="" type="radio"/> nem

Kék az ég?

Your Answer(s)	Correct Answer(s)
<input type="radio"/> 8	<input checked="" type="radio"/> 8

hány lába van 2 kutynak?

Your Answer(s)	Correct Answer(s)
<input type="radio"/> 9	<input checked="" type="radio"/> 10

hány kérdés van ebben a feladatlapban

Your Answer(s)	Correct Answer(s)
<input type="radio"/> ló	<input checked="" type="radio"/> ló

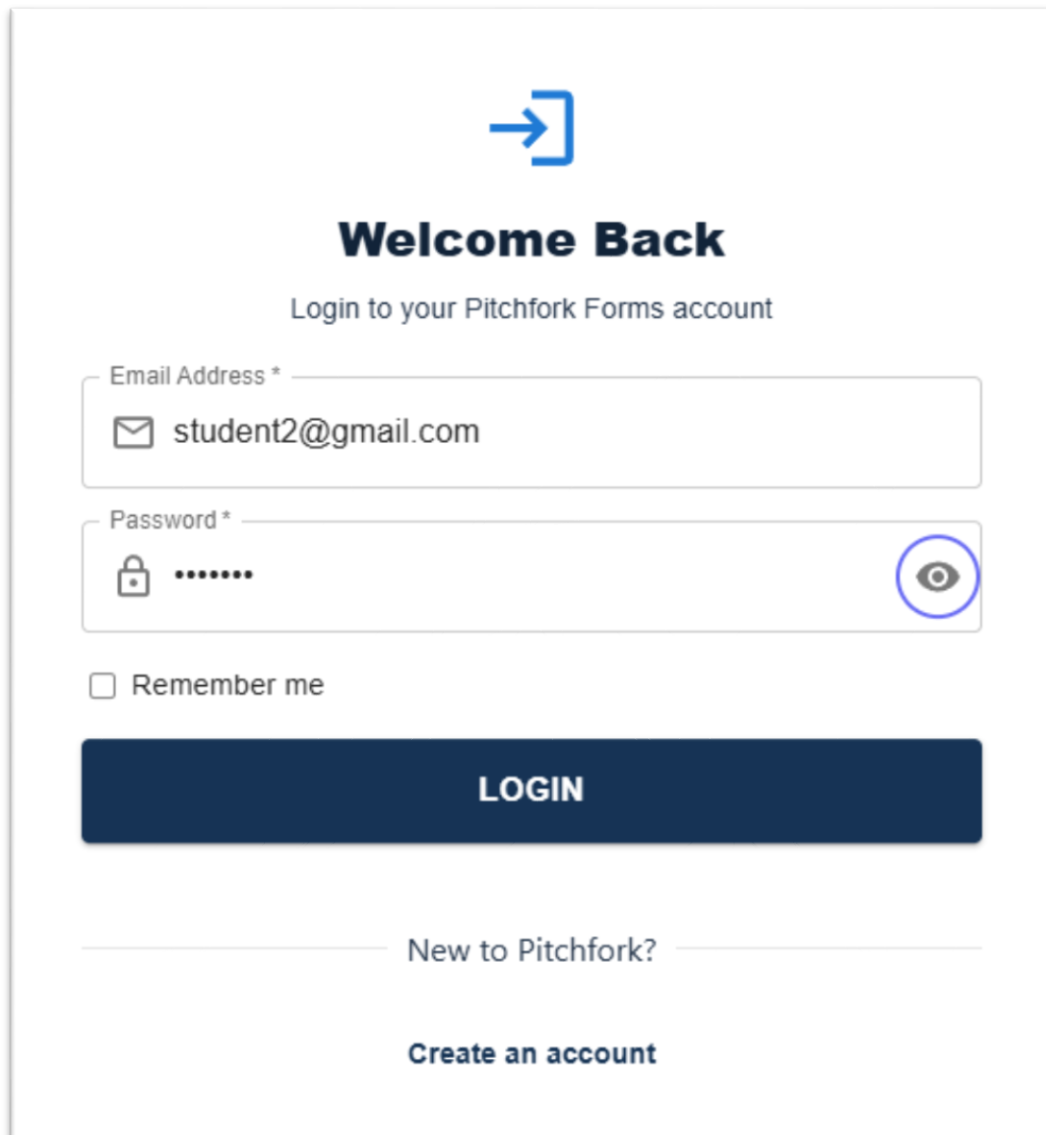
melyik állat gyorsabb

Your Answer(s)	Correct Answer(s)
<input type="radio"/> nem	<input checked="" type="radio"/> igen

ez egy kérdés?

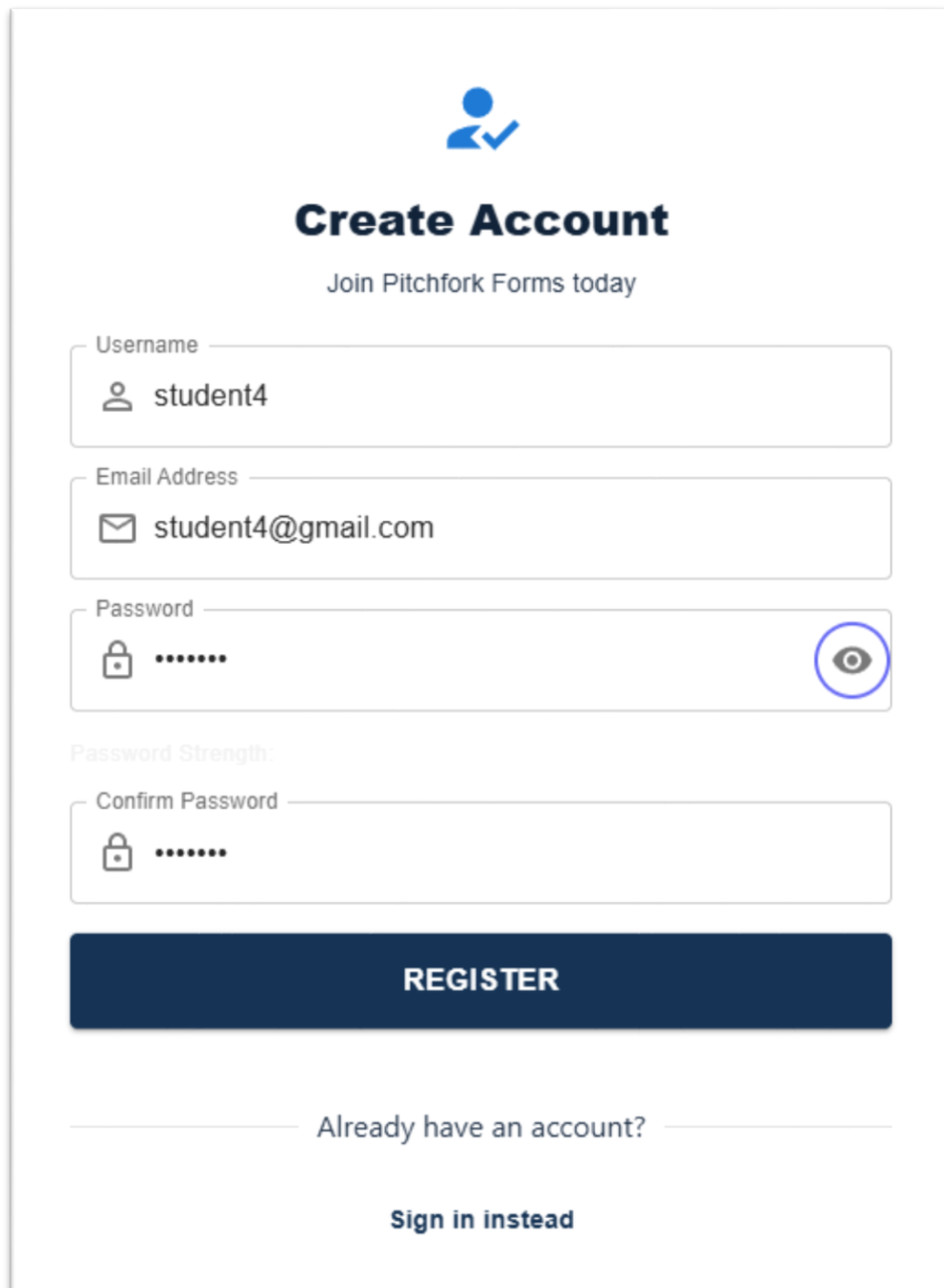
3.1.3.Általános funkciók

- **Bejelentkezés/Regisztráció:** A felhasználók beléphetnek, regisztrálhatnak az oldalon.
 - **Bejelentkezés:**



The image shows a login form for Pitchfork Forms. At the top, there is a blue icon of a right arrow pointing into a square bracket. Below this is the heading "Welcome Back" in bold, followed by the text "Login to your Pitchfork Forms account". The form contains two input fields: "Email Address *" with the value "student2@gmail.com" and an envelope icon, and "Password *" with a lock icon and a series of dots. To the right of the password field is a toggle icon (an eye inside a circle) to show or hide the password. Below the password field is a checkbox labeled "Remember me". A large dark blue button with the text "LOGIN" in white is positioned below the checkbox. At the bottom, there is a link "New to Pitchfork?" followed by a line separator and the text "Create an account".

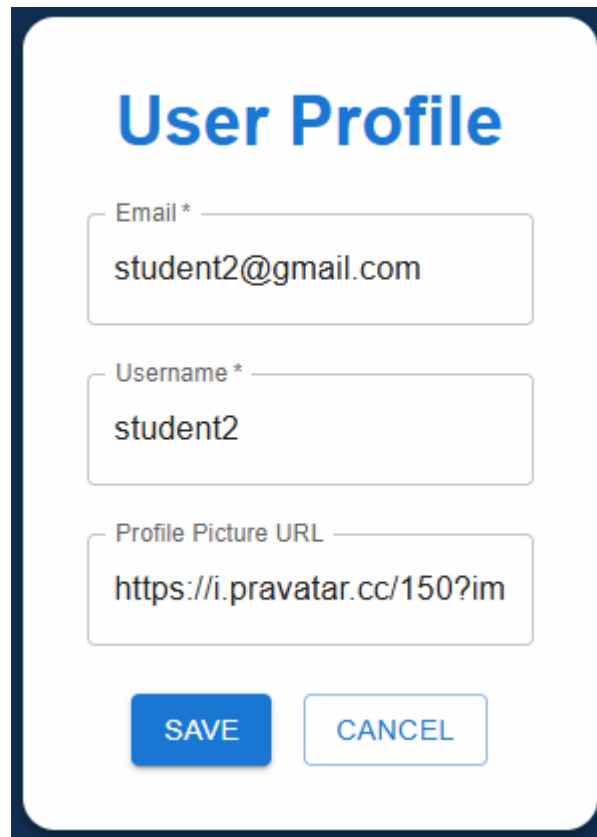
- **Regisztráció**



The image shows a 'Create Account' form for Pitchfork Forms. At the top, there is a blue icon of a person with a checkmark. Below it, the title 'Create Account' is displayed in a large, bold, dark blue font, followed by the subtitle 'Join Pitchfork Forms today' in a smaller, grey font. The form consists of several input fields: 'Username' with a person icon and the text 'student4'; 'Email Address' with an envelope icon and the text 'student4@gmail.com'; 'Password' with a lock icon, masked dots, and a toggle eye icon circled in blue; and 'Confirm Password' with a lock icon and masked dots. A 'Password Strength:' label is positioned above the confirm password field. At the bottom of the form is a large, dark blue 'REGISTER' button. Below the button, there is a link 'Already have an account?' and a 'Sign in instead' button.

- **Biztonságos hitelesítés:** JWT-alapú bejelentkezés és jogosultságkezelés.
- **Reszponzív dizájn:** Az alkalmazás minden eszközön (asztali gép, laptop, mobiltelefon, tablet) megfelelően működik.

- **Felhasználói profilok kezelése:** A felhasználók szerkeszthetik saját profiljukat (pl. név, email, profilkép).

A screenshot of a 'User Profile' form. The title 'User Profile' is at the top in blue. Below it are three input fields: 'Email *' with the value 'student2@gmail.com', 'Username *' with the value 'student2', and 'Profile Picture URL' with the value 'https://i.pravatar.cc/150?im'. At the bottom are two buttons: 'SAVE' (solid blue) and 'CANCEL' (outlined blue).

3.1.4. Hibajelzések, magyarázatuk, megoldásuk

3.1.4.1. "Failed to fetch forms" hiba

- **Magyarázat:** A frontend nem tudja elérni a backend `/admin/students-forms/:id` végpontját.
- **Megoldás:**
 - Ellenőrizd, hogy a backend fut-e (`http://localhost:3000`).
 - Győződj meg arról, hogy a frontend és backend ugyanazon a hálózaton fut.

3.1.4.2. "Invalid credentials" hiba bejelentkezéskor

- **Magyarázat:** Hibás email vagy jelszó lett megadva.
- **Megoldás:**
 - Ellenőrizd a beírt adatokat.
 - Ha elfelejtetted a jelszót, kérj segítséget az adminisztrátortól.

3.1.4.3. "Access denied" hiba

- Magyarázat: A felhasználó nem rendelkezik megfelelő jogosultságokkal az adott művelethez.
- **Megoldás:**
 - Ellenőrizd, hogy a megfelelő szerepkörrel (admin/tanár, diák) vagy-e bejelentkezve.

3.1.4.4. "Server error" hiba

- Magyarázat: A backend szerver hibát észlelt.
- **Megoldás:**
 - Ellenőrizd a backend konzolját a részletes hibaüzenetért.
 - Győződj meg arról, hogy az adatbázis megfelelően van konfigurálva.

3.1.4.5. "No response from server" hiba

- Magyarázat: A frontend nem tudott kapcsolatot létesíteni a backenddel.
- **Megoldás:**
 - Ellenőrizd a hálózati kapcsolatot.
 - Győződj meg arról, hogy a backend fut.

3.2. Információkérés lehetőségei

Szerzők elérhetőségei:

- Németh Bence: nebence11@gmail.com
- Németh Mirkó Máté: nemeth.mikro@gmail.com
- Ragats Bálint: balintragats@gmail.com

4. Összefoglalás

A Pitchfork Forms egy könnyen kezelhető, digitális űrlapkezelő weboldal, amelyet kifejezetten iskolák számára fejlesztettünk. A célja, hogy egyszerűbbé, gyorsabbá és átláthatóbbá tegye az iskolai feladatok, dolgozatok lebonyolítását.

A rendszer lehetővé teszi, hogy a tanár saját maga hozza létre és kezelje az online űrlapokat, amelyeket a diákok könnyedén ki tudnak tölteni és beküldeni akár otthonról, akár az iskolából.

A Pitchfork Forms két fő részből áll:

- Adminisztrációs felület: ahol az iskola dolgozói létrehozzák és kezelik az űrlapokat, valamint nyomon követhetik azokat.
- Nyilvános felület: ahol a felhasználók (pl. diákok) egyszerűen kitölthetik az űrlapokat interneten keresztül, bármilyen eszközről.

A rendszer gyorsan bevezethető és könnyen testre szabható. Segítségével az iskola papírmunkája jelentősen csökken, az adminisztráció pedig sokkal hatékonyabbá válik.

5. Irodalomjegyzék

Backend:

- Express.js dokumentáció: <https://expressjs.com/>
- Node.js dokumentáció: <https://nodejs.org/en/docs>
- MySQL dokumentáció: <https://dev.mysql.com/doc/>
- JWT dokumentáció: <https://jwt.io/introduction>
- React dokumentáció: <https://react.dev>
- Material UI dokumentáció: <https://mui.com>
- NPM dokumentáció: <https://www.npmjs.com>
- React router dokumentáció: <https://reactrouter.com/>
- Axios dokumentáció: <https://axios-http.com/docs/intro>
- Jest dokumentáció: <https://jestjs.io/docs/getting-started>
- Vite dokumentáció: <https://vitejs.dev/>
- Stackoverflow: <https://stackoverflow.com/questions>
- GitHub dokumentáció: <https://docs.github.com/en>

6. Melléklet

6.1. Gitlab Link

- <https://gitlab.vasvill.hu/szoftverfejleszto-es-tesztelo-vizsgaremek-2024-25/nmm-nb-rb-pitchforkforms>

6.2. Github Link

- <https://github.com/getrektgit/pitchforkForms>