



M Ű E G Y E T E M 1 7 8 2

**Budapesti Műszaki és Gazdaságtudományi Egyetem**  
Villamosmérnöki és Informatikai Kar  
Méréstechnika és Információs Rendszerek Tanszék

Le Ngoc Toan

# **WEBES ŰRLAPKEZELŐ ALKALMAZÁS FEJLESZTÉSE**

KONZULENS

Eredics Péter

BUDAPEST, 2024

# Tartalomjegyzék

<b>FELADATKIÍRÁS .....</b>	<b>4</b>
<b>Összefoglaló .....</b>	<b>5</b>
<b>.Abstract.....</b>	<b>6</b>
<b>1. Bevezetés .....</b>	<b>7</b>
Személyes motiváció .....	7
<b>2. Specifikáció.....</b>	<b>9</b>
2.1 Funkcionális követelmények .....	10
<b>3. Tervezés .....</b>	<b>12</b>
3.1 Alkalmazás architektúrája.....	12
3.2 Backend keretrendszer .....	12
3.3 Frontend technológia .....	13
3.4 Adatbázis .....	14
<b>4. Implementáció.....</b>	<b>16</b>
4.1 Backend implementáció.....	16
4.1.2 Adatbázis séma .....	17
4.1.3 Projektstruktúra.....	21
4.1.4 Fájlstruktúra .....	22
4.1.5 Backend szerver .....	25
4.1.6 Felhasználói kezelés: .....	26
4.2 Frontend implementáció .....	27
4.2.1 Fájlstruktúra .....	27
4.2.2 Kód felépítés: .....	30
4.2.3 Felhasználói felület tervezése .....	30
4.2.4 Webalkalmazás publikálása .....	31
<b>5. Az alkalmazás bemutatása .....</b>	<b>32</b>
<b>6. Összegzés.....</b>	<b>49</b>
<b>7. Továbbfejlesztési lehetőségek .....</b>	<b>50</b>
<b>8. Irodalomjegyzék.....</b>	<b>51</b>
<b>9. Kódrészletek jegyzéke .....</b>	<b>52</b>

# HALLGATÓI NYILATKOZAT

Alulírott **Le Ngoc Toan**, szigorló hallgató kijelentem, hogy ezt a szakdolgozatot/diplomatervet meg nem engedett segítség nélkül, saját magam készítettem, csak a megadott forrásokat (szakirodalom, eszközök stb.) használtam fel. Minden olyan részt, melyet szó szerint, vagy azonos értelemben, de átfogalmazva más forrásból átvettem, egyértelműen, a forrás megadásával megjelöltem.

Hozzájárulok, hogy a jelen munkám alapadatait (szerző(k), cím, angol és magyar nyelvű tartalmi kivonat, készítés éve, konzulens(ek) neve) a BME VIK nyilvánosan hozzáférhető elektronikus formában, a munka teljes szövegét pedig az egyetem belső hálózatán keresztül (vagy hitelesített felhasználók számára) közzétegye. Kijelentem, hogy a benyújtott munka és annak elektronikus verziója megegyezik. Dékáni engedéllyel titkosított diplomatervek esetén a dolgozat szövege csak 3 év eltelte után válik hozzáférhetővé.

Kelt: Budapest, 2024. 12. 13.

.....  
Le Ngoc Toan

# FELADATKIÍRÁS

A webes űrlapkezelő alkalmazás fejlesztése elengedhetetlen a digitális kommunikáció és adatgyűjtés hatékonyságának növelésében. Az automatizált űrlapkezelés nemcsak a felhasználói élményt javítja, hanem jelentős időmegtakarítást és hibacsökkentést is eredményez a különböző szektorokban, mint például az üzleti, oktatási vagy egészségügyi területeken. Ezzel a megoldással lehetőséget teremtünk arra, hogy a szervezetek egyszerűbben és biztonságosabban gyűjtsenek, kezeljenek és elemezzenek adatokat, így hozzájárulva a döntéshozatali folyamatok hatékonyságához és a szolgáltatások minőségének javításához.

Jelen szakdolgozat keretein belül a feladat egy a nagy szolgáltatók (Google, Microsoft) hasonló megoldásaihoz hasonló webes űrlapkezelő alkalmazás fejlesztése. Az alkalmazás használatával a felhasználónak képesnek kell lennie új űrlapot létrehozni, az űrlapon megjelenített mezőket egyesével felrögzíteni (mező típusát és egyéb paramétereit beállítani), majd ezt az űrlapot egy megfelelő URL formájában másokkal megosztani. A megosztott URL használatával a kitöltő felhasználók legyenek képesek az űrlap mezőit kitölteni, majd a program az űrlap tulajdonosa számára adjon a kapott válaszokból jól áttekinthető összefoglalást.

A hallgató feladatának a következőkre kell kiterjednie:

Tekintse át a piacon elérhető hasonló megoldásokat, és gyűjtse össze a megvalósításra érdemes funkciókat!

Tervezze meg a kiválasztott funkciók ellátására képes rendszert!

Implementálja a megtervezett rendszert!

Mutassa be az elkészült rendszer működését!

Határozza meg az esetleges továbbfejlesztési lehetőségeket!

# Összefoglaló

Szakdolgozatom célja egy olyan űrlapkezelő webalkalmazás fejlesztése, amely a mindennapokban széleskörűen alkalmazható, és lehetővé teszi az adatok gyors és hatékony gyűjtését. Az összegyűjtött felhasználói adatok elemzése során értékes információkra lehet bukkanni, amelyek különösen hasznosak lehetnek a mesterséges intelligencia számára, mivel az AI rendszerek rendkívül adatigényesek.

A projekt kezdetén áttekintettem a népszerű űrlapkezelő alkalmazásokat, például a Google Forms és a Microsoft Forms megoldásait. Elsősorban a funkcionalitásra, valamint a felhasználói felületek kialakítására helyeztem a hangsúlyt, hogy megértsem az ilyen rendszerek működésének alapjait.

Első lépésként az adatbázis modelljét terveztem meg. Különös figyelmet fordítottam az adatok rendszerezett tárolására, valamint az entitások közötti kapcsolatok kialakítására. Ezután a backend fejlesztését kezdtem el, amely során az ASP.NET Core keretrendszert választottam, mivel ezt a technológiát tanulmányaim során már elsajátítottam, és ideálisnak tartottam az adatvezérelt rendszerek megvalósításához.

A backend elkészítése után a frontend, azaz a felhasználói felület fejlesztése következett. Ehhez a React.js keretrendszert használtam, amely számos támogatott könyvtárat és fejlesztőbarát funkciót biztosít. A React segítségével, webes asztali felhasználói felületet hoztam létre, amely egyszerű felhasználói élményt nyújt.

A kész alkalmazást feltelepítettem egy webes környezetbe. A backend a Microsoft Azure felhőplatformon fut, míg a frontendet a GitHub Pages szolgáltatás segítségével publikáltam. A webes felület és a backend között HTTP-kéréseken keresztül zajlik a kommunikáció; a frontend a kéréseket továbbítja az Azure-on futó backend felé, amely az üzleti logikának megfelelően kiszolgálja azokat.

## **.Abstract**

The goal of my thesis is to develop a form management web application that can be widely used in everyday life and enables the fast and efficient collection of data. The analysis of the collected user data can uncover valuable insights, which can be particularly beneficial for artificial intelligence, as AI systems are highly data-dependent.

At the beginning of the project, I reviewed popular form management applications such as Google Forms and Microsoft Forms. I focused primarily on functionality and the design of user interfaces to understand the fundamental workings of such systems.

As a first step, I designed the database model. I paid special attention to the organized storage of data and the establishment of relationships between entities. Following this, I began developing the backend, for which I chose the ASP.NET Core framework. This technology was familiar to me from my studies and was ideal for implementing data-driven systems.

After completing the backend, I moved on to developing the frontend, i.e., the user interface. For this, I used the React.js framework, which offers numerous supported libraries and developer-friendly features. Using React, I created a modern, responsive user interface that provides a simple and intuitive user experience.

I deployed the completed application in a web environment. The backend runs on the Microsoft Azure cloud platform, while the frontend was published using GitHub Pages. Communication between the web interface and the backend occurs via HTTP requests; the frontend sends requests to the Azure-hosted backend, which processes them based on the business logic and serves the appropriate responses.

# 1. Bevezetés

Az űrlapkezelés webalkalmazás egy olyan webs digitális platform, amely lehetővé teszi a felhasználók és a rendszerek közötti interakciókat. Az internet elterjedésével, a technológia rohamos fejlődésével és a nagy mennyiségű adatok feldolgozásának szükségességével jelentős szerepet kapott az űrlapalkalmazások használata a mindennapi életben. A webes űrlapok széles körben, számos területen alkalmazhatók. Az egyik jól ismert példa, amikor egy cég meg akarja tudni, hogy a vevők mennyire elégedettek a termékeikkel, és ehhez visszacsatolást várnak a vevőktől egy termékelégedettség-felmérő űrlap segítségével. Az ilyen adatok legtöbbször hozzájárulhatnak a jövőbeli üzleti döntésekhez, amelyek a felhasználói igényeknek megfelelően jobb irányba terelhetik a cég a termékeit. A modern technológiai fejlesztői eszközök és funkciók, mint például az azonnali visszacsatolás, a valós idejű validáció, valamint a szemléletes és ismerős elemek használata, tovább növelik az űrlapok hatékonyságát és a felhasználói élményt.

Ugyanakkor az űrlapkezelés számos kihívással is jár. Egy teljesen új, azaz a jelenlegi piacon elérhető webes űrlapalkalmazásoktól teljes mértékben eltérő alkalmazást a felhasználók nem szívesen használnak, mivel az jelentős kezdeti befektetést igényel a részükről. Az emberi természet olyan, hogy az egyszerűbbet választja, tehát azt, amelyik azonnal használható, és nem igényel időráfordítást a program megtanulására. Emellett a tervezés során a nem megfelelő adatvalidáció, valamint a túlzsúfolt űrlapok szintén hozzájárulhatnak ahhoz, hogy csökkenjen a felhasználók elköteleződése, ami a program használatának hanyatlásában mutatkozik meg.

Tehát egy jó minőségű webes űrlap megtervezése korántsem olyan egyszerű és könnyű, mint annak használata felhasználóként. A szakdolgozat egyik célja bemutatni az elkészült terméket, valamint a tervezési folyamatot és az ott meghozott döntéseket.

## Személyes motiváció

A tanulmányaim során elsajátított készségeimmel igyekeztem egy teljesen önállóan működő webes alkalmazást elkészíteni. A fejlesztés során olyan eszközökkel és technológiákkal találkoztam, amelyek számomra nem voltak ismertek. Ez a mai

informatika világában gyakori jelenség, és ezáltal lehetőségem nyílt az adaptációs képességemet próbára tenni. Emellett igyekeztem elsajátítani a szoftverfejlesztési praktikákat és a fejlesztői gondolkodásmódot is.

Az évek során több különböző projekten dolgoztam, de a szakdolgozatom készítésekor egy összetettebb program fejlesztése keltette fel az érdeklődésemet. Emellett a modern technológiák, különösen a backend fejlesztés az ASP.NET segítségével és a React.js iránt is érdeklődtem, mivel a mai alkalmazások jelentős részét ezekkel az eszközökkel valósítják meg. Ezért ezeknek a technológiáknak az elmélyítésére is kiemelt figyelmet fordítottam.

Feladatként egy űrlapkezelő alkalmazás fejlesztését választottam, mivel ennek megvalósítása elég nagy kihívást jelentett, és a tudásomhoz mérten megfelelő nehézségű feladatnak találtam. Az űrlapkezelő alkalmazást különösen érdekesnek tartottam, mert hidat képez a felhasználó és az alkalmazás között, ezzel egyszerűsítve a kommunikációt. A mesterséges intelligencia jelenlegi tendenciái miatt az űrlapokat hasznosnak tartom, mivel ezek széles körben elterjedt információgyűjtő eszközök. A mai világban, ahol az adatok kincset érnek, az ilyen megoldások különösen fontosak.



## 2. Specifikáció

Az űrlapkezelő alkalmazás lehetőséget biztosít arra, hogy különböző űrlapokat hozzunk létre. Két fajta űrlap lehetséges: *Forms* (űrlap) és *Quiz* (kvíz). A kettő csak abban tér el, hogy a *Quiz* típusnál az egyes kitöltések után megjeleníti a helyes válaszokat, amelyeket az űrlap létrehozója állított be. Ezek alapján van egy összpontszám, amely a helyes válaszokra kapott pontokból tevődik össze. A *Forms* típus csak elmenti az adott válaszokat, de nem jelenít meg helyes válaszokat. A *Forms* típus célja legfőképpen az adatgyűjtés és a válaszokból készített statisztikai elemzés. A *Quiz* típus szintén tudja az előbb említett funkciókat, ugyanakkor itt a kérdésekhez általában tartozik egy egyértelmű helyes válasz, amelyet a felhasználónak ki kell találnia.

Egy kitöltött *Quiz* űrlap során megjeleníti a helyes megoldásokat, amint ez lehetséges. Az ilyen űrlapok kitöltésére nincs időkorlát, tehát bármikor ki lehet tölteni, viszont csak egyetlen egyszer engedi meg a rendszer. Az egyes kérdésekhez képek is tartozhatnak, amelyek a kérdéshez kapcsolódnak.

Egy űrlap létrehozása során meg lehet adni a címet és a leírást is. Tetszőleges mennyiségű kérdést lehet létrehozni. Be lehet állítani, hogy milyen típusú (*Form* vagy *Quiz*) legyen az űrlap, és minden kérdéshez fel lehet tölteni legfeljebb egy képet. Minden kérdéshez az alábbi három válaszadási típus közül lehet választani: *textbox*, *checkbox*, *radiobox*. A kérdések sorrendje a kérdések létrehozási sorrendjével egyezik meg. Létrehozás során ki lehet törölni a nem kívánt kérdéseket is. Az űrlap elmentése után még lehet szerkeszteni annak tartalmát, majd elmenteni a módosításokat. A módosítás és az űrlap létrehozása egy gomb segítségével történik. Az űrlapok módosítását csak az a felhasználó tudja elvégezni, aki létrehozta azokat. Más felhasználók közvetlenül nem tudják szerkeszteni azokat.

Minden felhasználónak joga van űrlapot létrehozni, módosítani és kitölteni. A felhasználók az alkalmazás kezdőoldalán érhetik el a már korábban létrehozott űrlapokat. Van egy keresősáv, amely segít a korábban létrehozott kérdőívekre való rákeresésben azok címük alapján.

Minden egyes Formhoz tartozik három különböző gomb, amelyek a következő funkciókat valósítják meg: szerkesztés, statisztika megtekintése és törlés. A szerkesztés gombbal lehet módosítani a kérdőív kérdéseit és egyéb tulajdonságait. A statisztika

gombokkal meg lehet tekinteni a felhasználók válaszainak statisztikáját. Minden kérdéshez tartozik egy kördiagram és egy oszlopdiagram. A törlés gombbal véglegesen törölhető az adott űrlap.

Minden kérdésnél be lehet pipálni, hogy az adott kérdés kötelező legyen kitölteni, és ezt a rendszer ellenőrizni is fogja.

Az űrlapokhoz URL alapján lehet hozzáférni, amelyet az űrlap létrehozója oszt meg. A kezdőoldalon található egy másik keresősáv, amely ilyen URL-ek elérésében segít.

## 2.1 Funkcionális követelmények

### Felhasználók kezelése:

Minden felhasználó ugyanazokat a funkciókat éri el. Az űrlapot létrehozó felhasználó jogosult az űrlapokat módosítani, törölni, és hozzáférhet azok statisztikáihoz. A többi felhasználónak nincs ilyen jogosultsága.

### Űrlapok létrehozása és szerkesztése:

A felhasználó űrlap létrehozásakor a következő mezőtípusokat adhatja hozzá az űrlaphoz:

- **TextBox** (Szöveges mező): Egy szöveges választ lehet megadni, amely tetszőleges hosszúságú lehet.
- **CheckBox** (Jelölőnégyzetek): Legalább 1 és legfeljebb 4 lehetséges válaszopció közül bármennyit ki lehet választani.
- **RadioBox** (Rádiógombok): Legalább 1 és legfeljebb 4 lehetséges válaszopció közül 1-et lehet kiválasztani.

### Űrlapok kitöltése:

- A felhasználók böngészőn keresztül érhetik el az aktív űrlapokat.
- A rendszer biztosítja a bevitt adatok validálását (például kötelező mezők ellenőrzése, megfelelő formátumok betartása).

### Adatok tárolása és elemzése:

Az űrlapokhoz kapcsolódó kitöltési adatokat egy központi adatbázis tárolja. Az űrlapot létrehozó felhasználó statisztikákat tekinthet meg az űrlapokra beérkezett válaszokról, például:

- Válaszarány: Kördiagram formájában.
- Leggyakoribb válaszok: Oszlopdiagram formájában.

**URL-alapú hozzáférés:**

- Az űrlapok egyedi URL-en keresztül érhetők el, így könnyen megoszthatók.

## 3. Tervezés

### 3.1 Alkalmazás architektúrája

Az alkalmazás fejlesztése során a Layered architektúrát alkalmaztam, mivel az űrlapkezelő alkalmazást közepes méretűnek tekintettem. Emellett a jövőbeli fejlesztések szempontjából is ideálisnak tartom ezt az architektúrát, mivel jól skálázható, és hosszú távon is hatékonyan működik.

Az alkalmazás felépítése lehet olyan is, hogy minden réteg egyetlen projektben található. Ezt nevezzük monolitikus architektúrának. Ezzel egyszerűen és gyorsan lehet alkalmazást készíteni, amely kezdetben jól működik. Azonban az ilyen kódok karbantarthatósága és átláthatósága egyre rosszabbá válik, ha a kódbázis jelentősen növekedni kezd. A projekt kezdetekor ezt számításba vettem, és inkább a biztonságosabb, több rétegű architektúrát választottam.

A Layered architektúrának köszönhetően az alkalmazás két jól elkülöníthető részből áll: a frontendből és a backendből. A backend szolgáltatások (API-k) biztosítják az üzleti logikát és az adatkezelést, míg a React-alapú frontend a modern Single Page Applications (SPA) számára készült.

A **több rétegű architektúra** előnyei, amelyeket figyelembe vettem: [\[1\]](#), [\[2\]](#)

- Lehetővé teszi a megjelenítési réteg, az üzleti logika és az adatkezelés szétválasztását.
- A rétegek logikus elválasztása lehetővé teszi azok független fejlesztését és karbantartását.
- A szétválasztott logika miatt a rétegek külön-külön fejleszthetők és tesztelhetők, ami jobb karbantarthatóságot és egyszerűbb hibakeresést eredményez.

### 3.2 Backend keretrendszer

Tanulmányaim során megismerkedtem a C# nyelvvel, és mivel a hozzá kapcsolódó technológiákkal (MS SQL, REST API, LINQ, Entity Framework) már korábban is volt tapasztalatom, az ASP.NET Core használata tűnt a legkézenfekvőbb választásnak ezek elmélyítésére, így ezt választottam.

Ugyanakkor a tanulmányaim során nemcsak az ASP.NET Core technológiával volt lehetőségem foglalkozni, hanem például Node.js-sel és Java Springgel is. A Java nyelv kísértetiesen hasonlít a C# nyelvre, így ez is egy opció lett volna. Mindamellett a Spring-technológiával inkább csak elméleti szinten tudtam foglalkozni, ezért valódi tapasztalati hiányom volt benne.

A Node.js technológiával is dolgoztam, hiszen készítettem korábban egy kis projektet ezzel. Ezen a technológián belül a mikroservices architektúrát használják a leggyakrabban, ami számomra nem volt annyira kényelmes és megszokott.

Az **ASP.NET Core** előnyei, amelyeket figyelembe vettem: [\[3\]](#), [\[4\]](#), [\[5\]](#)

- Lehetővé teszi egyszerű és hatékony REST API-k készítését. Ide tartozik az automatikus modell-ellenőrzés, a JSON-alapú válaszok generálása, valamint az endpoint routing rendszere.
- Az Entity Framework Core egyszerűsíti az adatbázis-műveletek (CRUD) kezelését, és támogatja a LINQ lekérdezéseket, ami rugalmas és hatékony adatkezelést biztosít.
- Az ASP.NET Core szorosan integrálódik a Visual Studio és a Visual Studio Code fejlesztői környezetekkel, amelyeket a fejlesztési folyamat során használtam.
- Beépített támogatást nyújt a JWT tokenek kezeléséhez, amelyek modern autentikációs és autorizációs megoldásként szolgálnak.

### 3.3 Frontend technológia

A React.js egy népszerű keretrendszer, azonban tanulmányaim során saját magamnak kellett utánajárnom és megtanulnom a használatát, mivel az egyetemen sajnos nem találtam róla oktatási anyagot. Ennek ellenére az internetes segédanyagok sokat segítettek abban, hogy megértsem a keretrendszert, így könnyen le tudtam küzdeni az akadályokat.

A projekt elején fontolóra vettem az Angular keretrendszert is. A frontend-technológiák nagy része ugyanazokat az eszközöket (JavaScript, HTML, CSS) használja, így valószínűleg bármelyik keretrendszert könnyen meg tudtam volna tanulni, mivel ezeket az eszközöket már ismerem. Az Angular azért merült fel lehetőségként, mert korábban már készítettem egy projektet ezzel. Ennek a keretrendszernek az alapjait is ismerem, és nagyjából ugyanolyan funkciói vannak, mint a Reactnek.

A Vue.js is egy opció lehetett volna. Habár ezzel korábban nem volt tapasztalatom, az elmondások szerint hasonlít az Angularra, csak valamivel könnyebb annál. A projekt elején azonban sok időt fordítottam a backend fejlesztésére, és az idő szűkössége miatt sajnos nem hagytam elegendő időt arra, hogy kipróbáljam a Vue.js keretrendszert.

A **React.js** előnyei, amelyeket figyelembe vettem: [\[6\]](#)

- A keretrendszer alapjait nagyon könnyen el lehet sajátítani. Már egy közepes szintű JavaScript-tudással is minőségi alkalmazásokat lehet fejleszteni vele.
- Nagy és aktív közösséggel rendelkezik, ami hosszú távon kifejezetten előnyös.
- Számos elérhető könyvtár könnyíti meg a fejlesztést, és a keretrendszert rendszeresen frissítik.

### 3.4 Adatbázis

A projekt folyamán SQL-alapú adatbázisokat vizsgáltam, mivel kezdetben úgy gondoltam, hogy számos sémát kell létrehozni, és a közöttük lévő kapcsolatokat kialakítani. Az SQL-alapú adatbázisok alkalmasak az adatok hierarchikus tárolására, valamint kiválóan kezelik az adatok közötti kapcsolatokat.

A Node.js technológia megismerése során találkoztam a MongoDB-vel, amely egy NoSQL-alapú adatbázis. Ez egy olyan adatbázis, amely nem tárol relációs sémát, tehát nincs megköthető, hogy az egyes adatobjektumok hogyan nézzenek ki. Bár a MongoDB számos előnyt kínál az SQL-alapú megoldásokkal szemben, például rugalmasságot és skálázhatóságot, nem kezeli jól a relációk közötti kapcsolatokat, ami a projektem szempontjából kulcsfontosságú volt.

Kezdetben megvizsgáltam a SQLite-ot is, mivel korábban volt minimális tapasztalatom ezzel. Ez egy SQL-alapú adatbázis, így szintén jó opció lehetett volna. Azonban a SQLite hátránya, hogy nagy adatmennyiség esetén nem ideális, és korlátozott erőforrások mellett nem mindig hatékony. Emellett csak kevés gyakorlati tapasztalatom volt vele, ezért végül másik alternatívát kerestem.

Végül az MSSQL mellett döntöttem. Alapvetően olyan technológiát kerestem, amely könnyen integrálható az Entity Framework technológiával, és az MSSQL ezt tökéletesen támogatta. Ezért is választottam ezt az adatbázist a projektemhez.

Az **MSSQL** előnyei, amelyeket figyelembe vettem: [\[7\]](#), [\[8\]](#)

- Zökkenőmentesen együttműködik a .NET és Azure alkalmazásokkal, amelyeket a fejlesztés során használtam.
- A projekt során érzékeny felhasználói jelszavakat is tároltam, amelyeket titkosítani kellett. Az MSSQL natív titkosítási támogatást biztosít, ezáltal magas szintű biztonságot nyújtott az alkalmazás számára.

## 4. Implementáció

A backend kódforrása a [\[21\]](#)-as számú kódrészletek jegyzékében található.

### 4.1 Backend implementáció

A munkám kezdetén az elsődleges célom az volt, hogy elkészítsem a backend vázát. Ugyanakkor szükségem volt egy kezdetleges adatbázis modellre, mivel a Code-First megközelítést (kód alapú fejlesztési technikát) alkalmaztam. Ez egy olyan adatbázis-kezelési megközelítés, ahol először a programkódot írjuk meg, és a kód változásai után le tudjuk generálni az adatbázist. Ezzel meghatározzuk az adatbázis struktúráját és sémáját a program kódban, így nem kell közvetlenül hozzányúlni az adatbázishoz a létrehozásakor.

A Code-First alapú adatbázis létrehozása a következő lépésekből áll:

- **Modellek létrehozása:** A modellek (osztályok) az adatbázisban tábláknak felelnek meg. Az egyes osztályoknak meg lehet adni attribútumokat (oszlopok, tulajdonságok), továbbá a kapcsolataikat is meg lehet adni szintén attribútumként.
- **DbContext osztály:** A modellek létrehozása után az ApplicationDbContext osztályt kell létrehozni és konfigurálni. Ez az osztály egy központi szereplő az Entity Framework Core-ban, amely az adatbázis kezelését végzi. A programban létrehozott modell mindegyikéhez található egy DbSet<TEntity> típusú attribútum a DbContext osztályban. Az OnModelCreating metódusban meg lehet adni, hogy az egyes tábláknak milyen kapcsolatai legyenek a többi táblával, és ezt az Entity Framework figyelembe is veszi. Ezt *Navigation Property*-nek nevezzük, amikor az osztály egy attribútum segítségével hozzáfér egy másik táblához a kapcsolatán keresztül.
- **Adatbázis létrehozása:** Ha a modellek és a DbContext osztály is elkészült, akkor az adatbázist létre lehet hozni. Ehhez a kódban létre kell hozni egy migrációt, amely segítségével módosíthatjuk az adatbázis struktúráját és sémáját. Majd az adatbázist frissíteni kell, és ezáltal a változások be fognak következni.



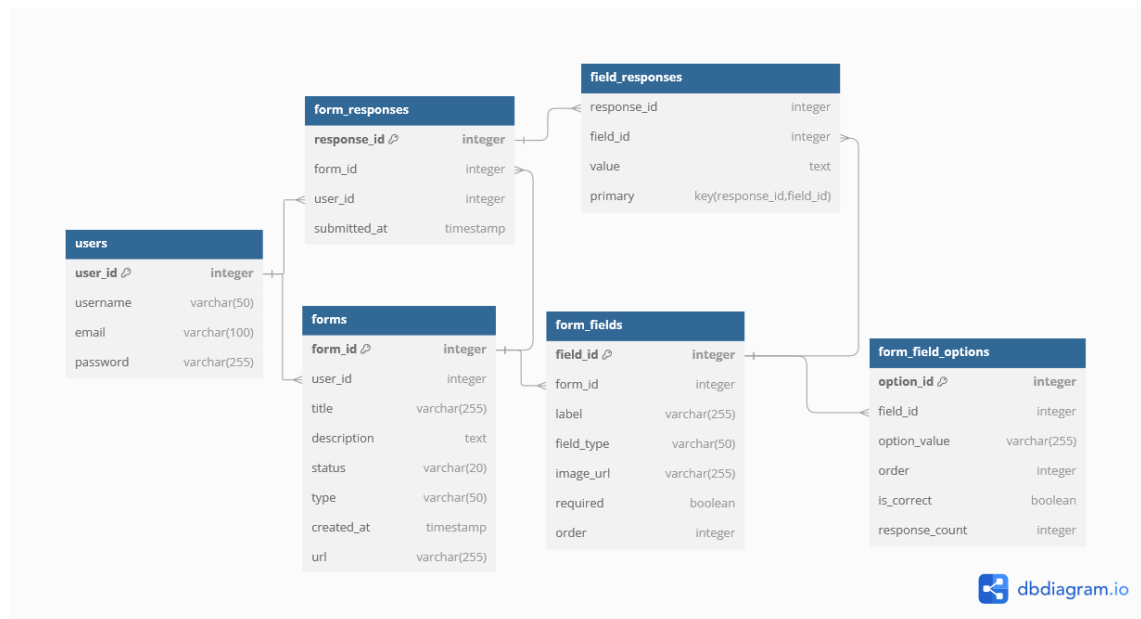
A DbContext osztály implementációját lásd a [9]-es számú kódrészletet a kódrészletek jegyzékében.

Új migráció létrehozására és az adatbázis frissítésére ezt a két kódot használtam a Developer PowerShell-ben:

```
dotnet ef migrations add init
dotnet ef database update
```

### 4.1.2 Adatbázis séma

Az alábbi képen látható az űrlap osztály diagramja.



1. ábra. Az űrlap adatbázis sémája

Minden osztályhoz tartozik egy elsődleges kulcs, amely egyértelműen azonosítja az adott entitást. Az alábbiakban olvasható az osztályok magyarázata:

#### 4.1.2.3 AppUser

Az AppUser egy osztály ami leszármazik a IdentityUser-tól. Az AppUser egy olyan osztály, amely leszármazik az IdentityUser ősosztályból. Az IdentityUser tartalmaz többek között egyedi UserId, e-mail cím és hash-elt jelszót tároló tulajdonságokat. Bár az ősosztály más tulajdonságokkal is rendelkezik, ezekre a program fejlesztése során nem volt szükség. Összességében az osztály felhasználói információkat tárol.

Az AppUser leszármazott osztály további *Navigation Property*-ket ad hozzá, amelyekkel az osztály kapcsolatokat alakít ki más táblákkal. Az AppUser tábla kapcsolódik a **Forms** (űrlapok) és a **FormResponses** (űrlapválaszok) táblákhoz.

A Navigation Property-k lehetővé teszik, hogy az AppUser táblából közvetlenül elérhetők legyenek az Űrlapok és az Űrlapválaszok.

Az **AppUser** osztálynál Navigation Property-k lehetővé teszik:

- A felhasználó által létrehozott kérdőívek közvetlen elérését az AppUser táblából (a **Forms** tulajdonság révén).
- A felhasználó által kitöltött kérdőívek közvetlen elérését az AppUser táblából (**FormResponses** segítségével)

Az osztály modell kódját lásd a [\[10\]](#)-es számú kódrészletet a kódrészletek jegyzékében.

#### 4.1.2.4 Form

A form az osztály az űrlap alapvető információit tárolja. Az osztály főbb tulajdonságai a következők:

- **Title:** az űrlap címe
- **Description:** az űrlap leírása
- **CreatedAt:** az űrlap létrehozási dátuma
- **Type:** egy típus, amely "Form" vagy "Quiz" értéket vehet fel
- **Url:** egy fix hosszúságú, véletlenszerűen generált stringet tárol.
- **UserId:** egy idegen kulcs attribútum, amely egyértelműen meghatározza, hogy ki az adott űrlap tulajdonosa.
- **Status:** egy fölösleges tulajdonság, amelyet végül nem használtam fel, de az adatbázis tervezésekor fontolóra vettem.

A **Form** osztálynál Navigation Property-k lehetővé teszik, hogy:

- A Form táblából elérjük azt a felhasználót, aki létrehozta az űrlapot (a **UserId** és **User** tulajdonság révén).
- Megszerezzük az összes kérdést, amely az adott űrlaphoz kapcsolódik (a **FormFields** tulajdonság révén).
- Hozzáférünk az űrlaphoz kapcsolódó felhasználói válaszokhoz (a **FormResponses** tulajdonság segítségével).

Az osztály modell kódját lásd a [\[11\]](#)-es számú kódrészletet a kódrészletek jegyzékében.

#### 4.1.2.5 FormField

A FormField egy olyan osztály, amely a kérdéshez tartozó tulajdonságokat tartalmazza. Az osztály főbb tulajdonságai a következők:

- **Label:** A kérdést tartalmazó szöveg.
- **FieldType:** Meghatározza a kérdés típusát, például Textbox, Checkbox, vagy Radiobox.
- **ImageUrl:** Egy URL, amely a kérdéshez tartozó kép hivatkozását tárolja. Mivel az SQL táblák nem tudnak közvetlenül képi információkat tárolni, ezért csak a kép linkjét mentem el.
- **Required:** Megadja, hogy az adott kérdés kitöltése kötelező-e.
- **Order:** Egy fölösleges tulajdonság, amelyet a tervezés korai fázisában hoztam létre, de végül nem használtam fel.

A FormField osztály Navigation Property-i lehetővé teszik, hogy:

- A FormField táblából elérjük a kérdéshez tartozó kérdőívet (a **FormId** és **Form** tulajdonságok segítségével).
- Megszerezzük az összes felhasználói választ, amely az adott kérdéshez kapcsolódik (a **FieldResponses** tulajdonság révén).
- Megszerezzük az összes lehetséges válaszopciót, amely az adott kérdéshez kapcsolódik (a **FormFieldOptions** tulajdonság révén).

Az osztály modell kódját lásd a [\[12\]](#)-es számú kódrészletet a kódrészletek jegyzékében.

#### 4.1.2.6 FormFieldOption

A FormFieldOption egy olyan osztály, amely a kérdéshez tartozó válaszopciókat tartalmazza.

Az osztály főbb tulajdonságai:

- **OptionValue:** A válaszopció szöveges értéke.
- **IsCorrect:** Meghatározza, hogy az opció helyes válasz-e.
- **ResponseCount:** Megmutatja, hogy az adott válaszopciót hányan választották.

- **Order:** Egy fölösleges tulajdonság, amelyet a tervezés során adtam hozzá, de végül nem használtam.
- **FieldId:** Egy idegen kulcs, amely a FormField táblához kapcsolódik, és az adott kérdést azonosítja.

A **FormFieldOption** osztály Navigation Property-je lehetővé teszi, hogy:

- A FormFieldOption táblából elérjük az opcióhoz tartozó kérdést a (**FieldId** és **FormField** tulajdonságok segítségével).

Az osztály modell kódját lásd a [\[13\]](#)-as számú kódrészletet a kódrészletek jegyzékében

#### 4.1.2.7 FormResponse

A FormResponse egy olyan osztály, amely egy kérdőívre adott válaszokat (FieldResponse-kat) foglalja egy gyűjteménybe.

Az osztály főbb tulajdonságai:

- **FormId:** Idegen kulcs, amely meghatározza, hogy melyik űrlaphoz tartozik a válasz.
- **UserId:** Idegen kulcs, amely megmondja, hogy ki töltötte ki a kérdőívet.
- **SubmittedAt:** A válasz benyújtásának időpontja.

A **FormResponse** osztály Navigation Property-i lehetővé teszi, hogy:

- A FormResponse táblából elérjük a kérdőívet kitöltő felhasználót (a **UserId** és **User** tulajdonságok segítségével).
- A FormResponse táblából elérjük azt az űrlapot, amelyet a felhasználó kitöltött (a **FormId** és **Form** tulajdonságok segítségével).

A FormResponse táblából elérhetjük a kérdőívhez tartozó válaszok listáját (a **FieldResponses** tulajdonság segítségével).

Az osztály modell kódját lásd a [\[14\]](#)-es számú kódrészletet a kódrészletek jegyzékében.

#### 4.1.2.8 FieldResponse

A FieldResponse egy olyan osztály, amely egy felhasználó által egy kérdésre megadott választ rögzíti

Az osztály főbb tulajdonságai:

- **Value:** Egy kérdésre adott szöveges felhasználói választ reprezentál.
- **FieldId:** Idegen kulcs, amely meghatározza, hogy mely kérdéshez tartozik a válasz.
- **ResponseId:** Idegen kulcs, amely meghatározza, hogy melyik kitöltött űrlap válaszhhoz tartozik az adott felhasználói válasz.

A **FieldResponse** osztály Navigation Property-i lehetővé teszik, hogy:

- A FieldResponse táblából elérjük, hogy mely kérdéshez tartozik az adott válasz (a **FieldId** és a **FormField** tulajdonság segítségével).
- A FieldResponse táblából elérjük, hogy az adott kérdés válasza mely kérdőív válaszához tartozik (a **ResponseId** és **FormResponse** tulajdonságok segítségével).

Az osztály modell kódját lásd a [\[15\]](#)-ös számú kódrészletet a kódrészletek jegyzékében.

### 4.1.3 Projektstruktúra

Az backend kialakításakor a repository mintát (repository pattern) alkalmaztam. Ez egy szoftvertervezési minta, amelynek célja, hogy elválassza az üzleti logikát (pl. a service réteget) az adat-hozzáférési logikától (pl. adatbázis műveletek). Ezt egy absztrakciós réteggel éri el, amely az adatforrással (pl. adatbázissal) végzett műveleteket kezeli, miközben elrejti a részleteket az alkalmazás többi része elől.

Az implementált backend-ben kettő alapvető réteg valósítja meg ezt a mintát:

#### Controller:

- A felhasználói kérések belépési pontja. API-végpontokat biztosító metódusokat tartalmaz.
- Feldolgozza az érkező kéréseket (pl. HTTP GET, POST, PUT, DELETE), és továbbítja azokat a megfelelő szolgáltatás (service) réteghez.
- Felelős a válaszok előállításáért is. A programomban minden kérést egy JSON adat válasszal tér vissza
- Az alkalmazásban ez valósítja meg az üzleti logikát mivel egyszerű logikát valósítanak meg. Nagyobb projekthez egy Service réteget érdemes bevezetni a Controller és a Repository réteg között.

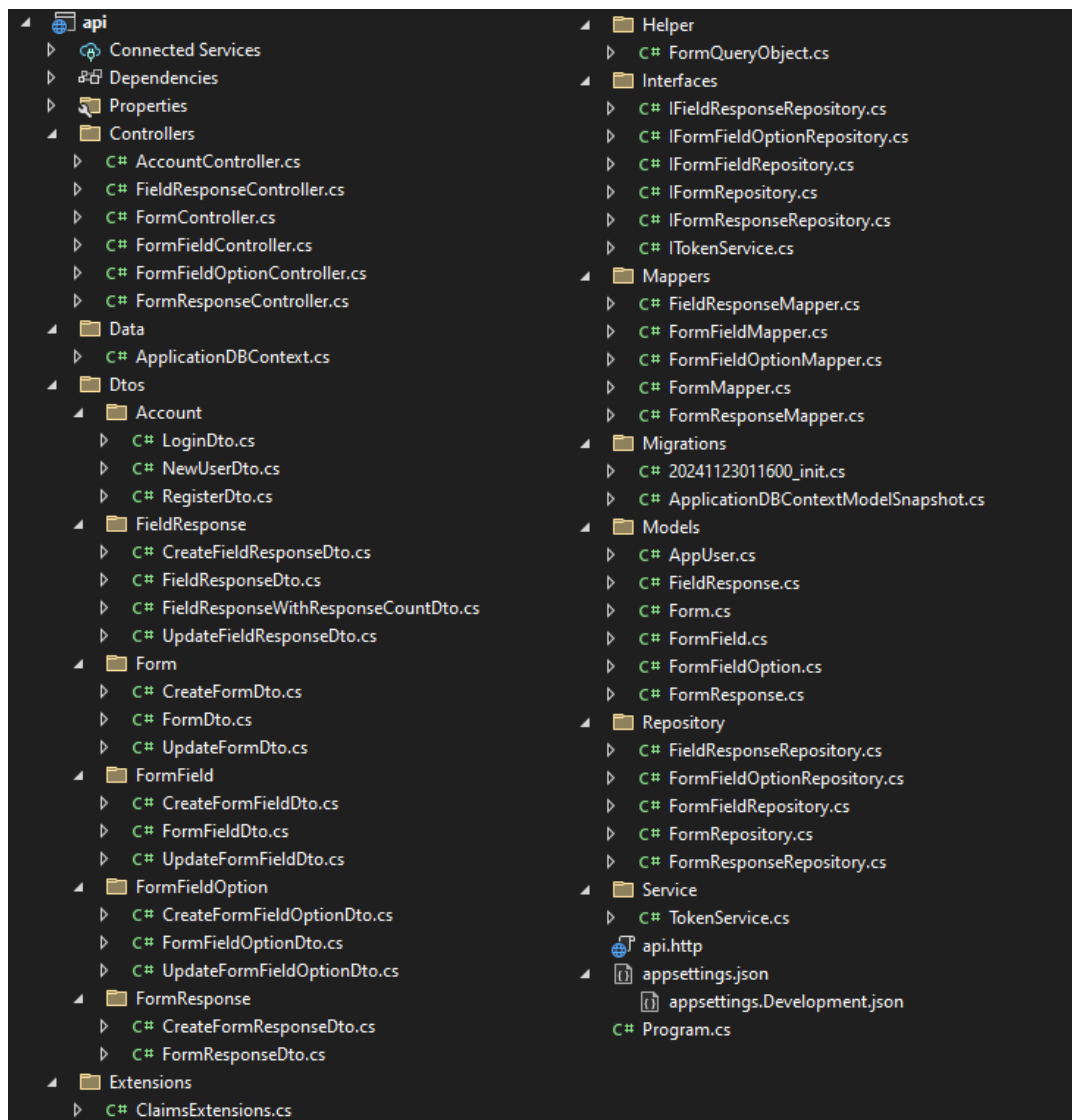
## Repository:

- Egy absztrakciós réteg, amely az adat-hozzáférésért felelős.
- Az adatbázisműveleteket végzi (CRUD: Create, Read, Update, Delete, magyarul: Létrehozás, Olvasás, Módosítás, Törlés).
- Könnyíti a szolgáltatásréteg munkáját azzal, hogy nem szükséges közvetlenül az adatbázissal foglalkoznia

A saját alkalmazásomban használtam a repository mintát, lásd a [\[16\]](#)-os számú kódrészletet a kódrészletek jegyzékében.

### 4.1.4 Fájlstruktúra

Az alábbi képeken látható látható a fájl struktúra:



2. ábra. A backend fájlstruktúrája

A következő fő mappák és fájlok találhatóak az alkalmazásban:

## 1. Controllers

Az API végpontokat kezelő osztályok. Az egyes entitásokhoz külön-külön vezérlő készült.

- **AccountController.cs:** Az autentikáció és a felhasználókezelés műveletei.
- **FieldResponseController.cs:** A mezőválaszokkal kapcsolatos végpontok.
- **FormController.cs:** A formok kezelésére szolgáló végpontok.
- **FormFieldController.cs:** A form mezőinek kezelésére szolgáló végpontok.
- **FormFieldOptionController.cs:** A form mező opcióinak végpontjai.
- **FormResponseController.cs:** A form válaszainak kezelése.

## 2. Data

Az adatbázissal kapcsolatos konfigurációkat és kontextust tartalmazza.

- **ApplicationDBContext.cs:** Az adatbázis-kapcsolatot és entitások konfigurációját biztosító osztály.

## 3. Dtos

Az adatok továbbítására és fogadására szolgáló objektumok (DTO-k). Ezek az adatok formázásáért és validálásáért felelősek.

## 4. Extensions

Kiterjesztett funkciókat biztosító segédosztályok.

- **ClaimsExtensions.cs:** A felhasználói jogok kezelése.

## 5. Helper

Segédosztályokat tartalmazó mappa

- **FormQueryObject.cs:** Segédobjektum a formok lekérdezéséhez szükséges paraméterek kezelésére.

## 6. Interfaces

Az interfészek gyűjtőhelye, amelyek az egyes repository osztályok szerződését (interfészét) definiálják.

- **IFieldResponseRepository.cs:** Mezőválaszokhoz tartozó adatbázisműveletek interfésze.
- **IFormFieldOptionRepository.cs:** Form mező opciókhoz kapcsolódó adatbázisműveletek interfésze.
- **IFormFieldRepository.cs:** Form mezőkhöz kapcsolódó adatbázisműveletek interfésze.
- **IFormRepository.cs:** Formok adatbázisműveleteinek interfésze.
- **IFormResponseRepository.cs:** Form válaszok adatbázisműveleteinek interfésze.
- **ITokenService.cs:** Tokenkezeléshez használt interfész.

## 7. Mappers

Az entitások és DTO-k közötti átalakítást végző statikus osztályok.

- **FieldResponseMapper.cs:** A mezőválaszok átalakításáért felel.
- **FormFieldMapper.cs:** A form mezők átalakításáért felel.
- **FormFieldOptionMapper.cs:** A form mező opciók átalakításáért felel.
- **FormMapper.cs:** A formok átalakításáért felel.
- **FormResponseMapper.cs:** A form válaszok átalakításáért felel.

## 8. Models

Az alkalmazás által használt entitások definíciói, amelyek az adatbázisban tárolódnak.

- **AppUser.cs:** A felhasználó entitás kiterjesztése.
- **FieldResponse.cs:** Mezőválasz entitás.
- **Form.cs:** Form entitás.
- **FormField.cs:** Form mező entitás.
- **FormFieldOption.cs:** Form mező opció entitás.
- **FormResponse.cs:** Form válasz entitás.

## 9. Repository

Az adatbázis-hozzáférési logikát megvalósító osztályok.



- **FieldResponseRepository.cs:** Mezőválaszok adatbázisműveleteinek implementációja.
- **FormFieldOptionRepository.cs:** Form mező opciók adatbázisműveleteinek implementációja.
- **FormFieldRepository.cs:** Form mezők adatbázisműveleteinek implementációja.
- **FormRepository.cs:** Formok adatbázisműveleteinek implementációja.
- **FormResponseRepository.cs:** Form válaszok adatbázisműveleteinek implementációja.

## 10. Service

Az üzleti logika és speciális szolgáltatások kezelése. A legtöbb üzleti logika a controller osztályokban van implementálva

- **TokenService.cs:** Felhasználói autentikációhoz szükséges JWT tokenek kezelése.

## 11. Migrations

Az adatbázis-migrációkat tartalmazó mappa.

## 12. Konfigurációs fájlok

- **appsettings.json:** Az alkalmazás konfigurációs beállításai mint például az adatbázis kapcsolat. Itt lehet beállítani, hogy mely adatbázishoz csatlakozzon a program
- **Program.cs:** Az alkalmazás belépési pontja.

### 4.1.5 Backend szerver

A Microsoft Azure felhőplatform kiválóan integrálható a C# nyelvvel, és ezt a munkám során ki is használtam. A projekt nagy részében lokális adatbázis-szervert használtam, de a projekt végén amikor már a teljes program összeállt igénybe vettem a felhőszolgáltatást. Ennek segítségével az interneten keresztül érhettem el a program API-ját és funkcióit.

Először létrehoztam egy SQL szerveret az Azure-ban, ahol tárolni tudtam az összes űrlaphoz kapcsolódó adatot. A lokális adatbázis adatai nem vesztek el, mivel ezeket előbb exportáltam, majd migráltam az új felhős adatbázisba.

Ezután a backend alkalmazást is feltöltöttem az Azure-ba, hogy közvetlenül hozzáférhessen a felhős adatbázishoz, és a későbbiekben kiszolgálhassa a felhasználói kéréseket. A programban konfigurálnom kellett, hogy melyik adatbázishoz csatlakozzon. Ezt egy úgynevezett *connection string* segítségével adtam meg az appsettings.json fájlban, lásd a [\[17\]](#)-es számú kódrészletet a kódrészletek jegyzékében.

A képeket is el kellett tárolnom egy szerveren. Az Azure-ra is fel lehet tölteni képeket, ugyanakkor inkább a Cloudinary felhőalapú megoldását választottam. Itt mindenféle fájlt fel lehet tölteni. [\[18\]](#)

#### 4.1.6 Felhasználói kezelés:

A program elkészítése során ügyeltem arra, hogy modern és biztonságos autentikációs és autorizációs eszközöket használjak. Az egyik ilyen technológia, amelyet alkalmaztam, a **JWT** (JSON Web Token), amely széles körben használatos webalkalmazásokban és API-kban. A JWT egy kompakt, ugyanakkor biztonságos token, amelyet egy szerver generál (az én esetemben az alkalmazás), és általában egy digitális aláírással hitelesít.

Az ASP.NET Core környezetben a JWT gyakran token-alapú hitelesítéshez szolgál. Felhasználó regisztrációja során a szerver a jelszót hash-eli, egy salt hozzáadásával tovább növelve a biztonságot, majd a titkosított jelszót tárolja az adatbázisban. Bejelentkezéskor a megadott jelszót ugyanazon az algoritmuson keresztül hash-eli, majd összehasonlítja az adatbázisban tárolt értékkel. Ha az egyezés sikeres, a szerver generál egy JWT-t, amelyet visszaküld a kliensnek.

A kliens ezt a tokent minden további kérésnél elküldi a backend szervernek az Authorization fejlécben. Ez lehetővé teszi, hogy az API ellenőrizze a token érvényességét, azonosítsa a felhasználót, és megállapítsa annak jogosultságait az adott művelet elvégzésére. A token tartalmazhat lejáratási időt, ami növeli a biztonságot, mivel a lejárt tokeneket a rendszer már nem fogadja el.

A felhasználókezelő rendszerben lehetőség van a jelszók validációjára is, például minimális hossz vagy speciális karakterek megkövetelésére. Ezt a validációt a Program.cs fájlban állítottam be az alábbi módon:

```
builder.Services.AddIdentity<AppUser, IdentityRole>(options =>
{
    options.Password.RequireDigit = true;
```

```
options.Password.RequireLowercase = false;  
options.Password.RequireUppercase = false;  
options.Password.RequireNonAlphanumeric = false;  
options.Password.RequiredLength = 4;  
})
```

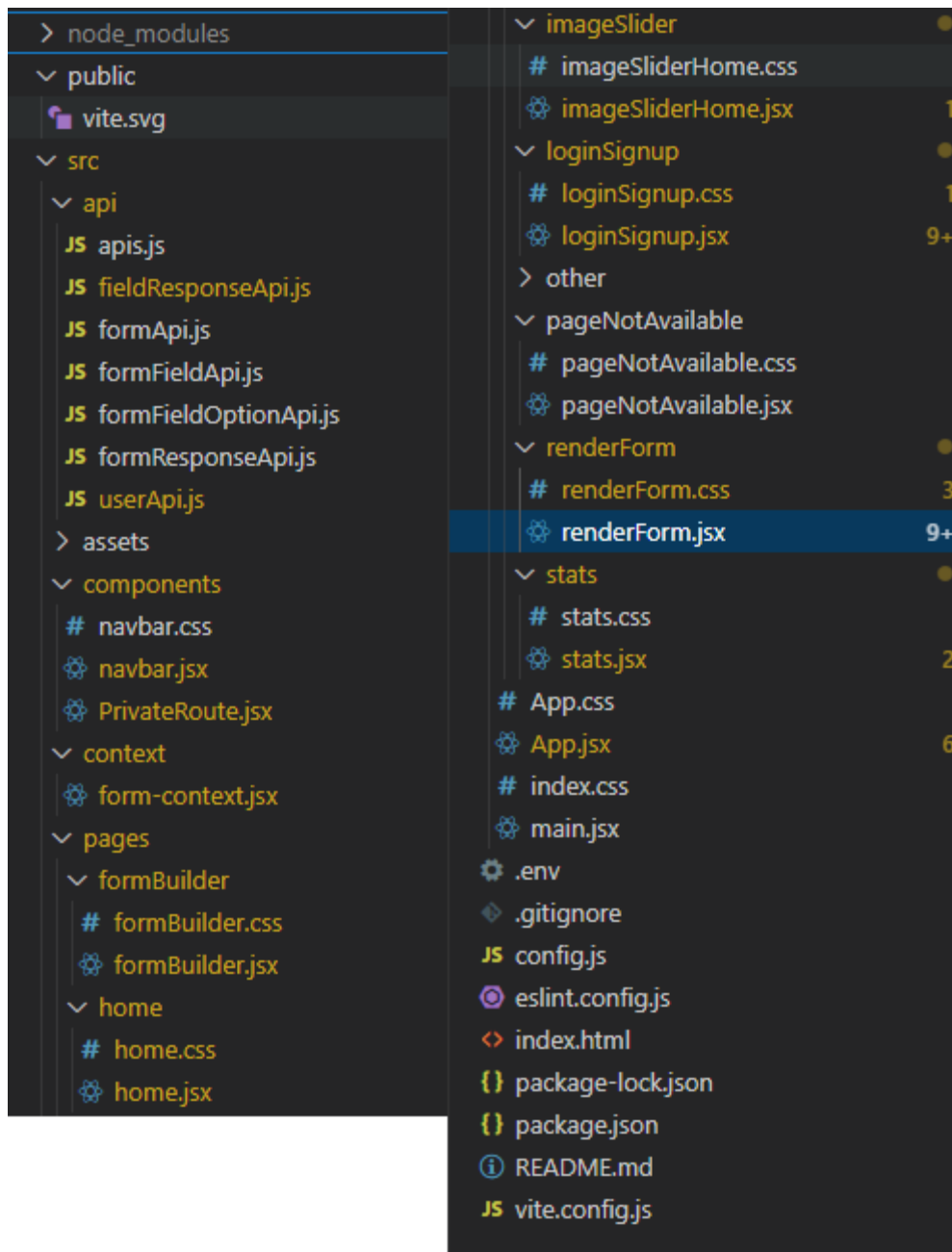
Ebben a beállításban látható, hogy a jelszónak tartalmaznia kell számot, és legalább 4 karakterből kell állnia.

## 4.2 Frontend implementáció

A frontend kódforrása a [\[22\]](#)-es számú kódrészletek jegyzékében található.

### 4.2.1 Fájlstruktúra

Az alábbi képen látható a fájlstruktúra:



3 ábra. A frontend fájlstruktúra

A következő fő mappák és fájlok találhatóak az alkalmazásban:

1. **node modules mappa:** A projekt által használt függőségeket tartalmazza
2. **src mappa:** Az alkalmazás forráskódja ebben a mappában található, ez tartalmazza az összes fő komponensét, oldalt, stíluslapot és egyéb segédfájlokat.
3. **api fájlok:** az alkalmazás által használt API hívások kezelésére szolgál.
4. **assets:** Az alkalmazásban használt statikus fájlokat, mint például képeket, ikonokat tárolók

**5. components:** Az újrahasznosítható komponensek gyűjteménye, amelyeket az alkalmazás különböző részein használhatunk.

- **navbar.jsx és navbar.css:** A navigációs sáv vizuális megjelenítéséért és működéséért felelős. A navbar.jsx tartalmazza a JSX kódot, amely a menüelemeket definiálja, míg a navbar.css gondoskodik a stílusokról.

**6. context:** A globális állapotkezelésért felelős.

**form-context.jsx:** Az űrlapokkal kapcsolatos adatokat osztja meg a különböző kódrészletekkel

**7. pages**

- **formBuilder.jsx és formBuilder.css:** Az űrlapkészítő oldal, amely lehetővé teszi a felhasználók számára űrlapok létrehozását.
- **pageNotAvailable.jsx és pageNotAvailable.css:** Egy oldal, amely nem tartalmaz tartalmat, ugyanakkor jelzi a felhasználó számára, hogy az oldal nem elérhető. Olyan esetekben hasznos, például amikor a felhasználó megpróbálja újra megnyitni a belépési oldalt, annak ellenére, hogy már be van jelentkezve.
- **home.jsx és home.css:** Az alkalmazás kezdőoldala a bejelentkezett felhasználók számára, amely a felhasználó által már elkészített űrlapokat jeleníti meg. Itt lehetőség van arra is, hogy az egyes űrlapokra az URL szöveg segítségével keressünk rá.
- **imageSliderHome.jsx és imageSliderHome.css:** A be nem jelentkezett felhasználók kezdőoldala. Itt néhány kép látható az alkalmazásról, amelyek felváltva jelennek meg a kezdőlap tetején.
- **loginSignup.jsx és loginSignup.css:** A bejelentkezési és regisztrációs oldal. Ez a fájl tartalmazza a felhasználói hitelesítési logikát.
- **renderForm.jsx és renderForm.css:** Egy meglévő űrlap megjelenítéséért és kitöltéséért felelős oldal.
- **stats.jsx és stats.css:** Egy űrlap statisztikai oldalát kezeli, amely megjeleníti az egyes kérdésekre adott válaszok arányát és mennyiségét.

**8. gyökérfájlok**

- **.env:** Környezeti változók tárolása. Itt tárolom a backend program URL-jét

- **App.jsx:** Az alkalmazás fő belépési pontja, amely meghatározza a route-okat és az alapvető komponenseket.

#### 4.2.2 Kód felépítés:

Egy oldal felépítése általában egy JSX és egy CSS fájlból áll. A **JSX** kiterjesztésű fájlok felelősek az oldal logikájáért, valamint az egyes vezérlők működésének megvalósításáért. A JSX lehetővé teszi, hogy HTML-szerű szintaxist használjunk JavaScript fájlokban, amelyeket React-komponensekben írunk meg.

A **CSS** kiterjesztésű fájlok az oldal megjelenítéséért felelnek. Ezek határozzák meg az oldal stílusát, például a színeket, betűtípusokat és elrendezéseket. A React alkalmazásokban általában különálló **.css** fájlokat használunk a stílusok meghatározására, de előfordulhat, hogy a stílusokat közvetlenül a **.jsx** fájlokban definiáljuk.

Egy oldalhoz tartozó egyszerű példa kódrészletet lásd a [\[19\]](#)-as számú kódrészletet a kódrészletek jegyzékében.

A React.js alkalmazás az API-kezelő fájlokban definiált funkciók segítségével kommunikál a backenddel. Az API kérések során leggyakrabban adatokat kérünk le, amelyeket az adott oldalakon a szükséges logikák szerint jelenítünk meg.

Például egy űrlap adatainak betöltéséhez lásd a [\[20\]](#)-as kódrészletnél a kódrészletek jegyzékében.

Az ilyen API-kéréseket megvalósító metódusokban általában az **async** és **await** kulcsszavakat használjuk. Ezek lehetővé teszik, hogy a program megvárja az adott kódsor végrehajtását (pl. egy API-hívás eredményét), mielőtt folytatná a következő művelettel. Így biztosíthatjuk, hogy a szükséges adatok betöltése vagy feldolgozása befejeződjön, mielőtt a kód többi része lefutna.

#### 4.2.3 Felhasználói felület tervezése

A webes alkalmazást igyekeztem egyszerűen és áttekinthetően megtervezni, hasonló módon, mint a többi népszerű űrlapkezelő alkalmazást. Törekedtem arra, hogy minden hibát kijavítsak, és minden fontos felhasználói műveletnél egyértelmű visszajelzést adjak a művelet sikerességéről.

A felhasználói felület tervezése során elsősorban a webes környezetre összpontosítottam, így a munkám nagy részében erre optimalizáltam az alkalmazást. A

munkafolyamat végén azonban rájöttem, hogy a webalkalmazás felhasználói élménye telefonon nem volt megfelelő. Emiatt bevezettem a **responsive design** elveit, amelyek lehetővé teszik, hogy a stílustulajdonságok automatikusan változzanak a képernyő szélességének függvényében (jellemzően kisebb méret esetén).

Ennek eredményeként a tartalom telefonon kissé eltérően jelenik meg, mint asztali felületen, azonban a felhasználó tartalmi és funkcionális szempontból nem tapasztal érdemi különbséget.

Egyszerű példa (imageSliderHome.css):

```
/* General container styling */
.slide-home-container {
  display: flex;
  justify-content: space-between;
  align-items: center;
  padding: 20px;
  height: 100vh;
  background-color: #ffffff;
}

/* Mobile-specific styling */
@media screen and (max-width: 768px) {
  .slide-home-container {
    flex-direction: column; /* Stack the elements vertically on smaller screens */
    justify-content: flex-start; /* Align items to the top */
    height: auto; /* Allow the container to adjust height */
    margin-top: 80px;
  }
}
```

Ebben a példában az alapértelmezett stílust a `.slide-home-container` osztályra alkalmazza a böngésző, de a `@media screen` blokkban lévő `slide-home-container` stílust akkor alkalmazza, ha a képernyő szélessége legfeljebb 768 pixel, azaz mobil felületen van a felhasználó.

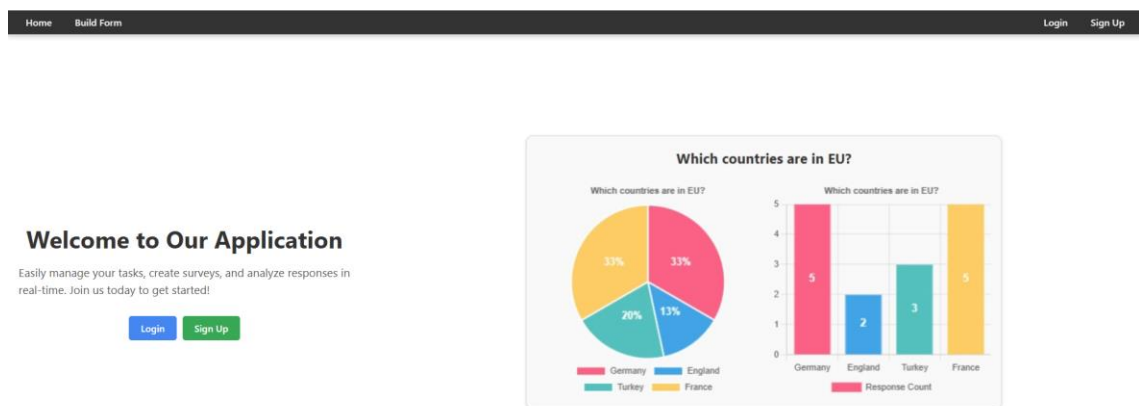
#### 4.2.4 Webalkalmazás publikálása

A webalkalmazást a saját GitHub-fiókom alá töltöttem fel a GitHub segítségével. A folyamat során létrehoztam egy saját kódtárat (repository). A feltöltés előtt konfigurálnom kellett az alkalmazás `package.json` fájljának beállításait.

## 5. Az alkalmazás bemutatása

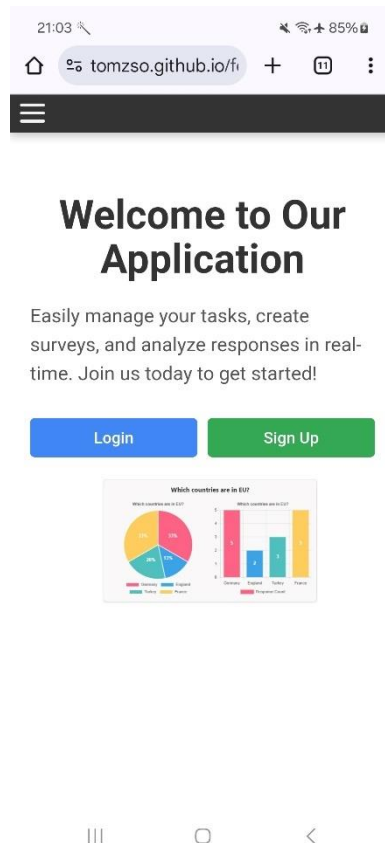
A webalkalmazás URL-je: <https://tomzso.github.io/form/>

Az alábbiakban bemutatom a kész alkalmazást mobiltelefonos és asztali számítógépes képernyőkön. Mindkét eszközön a Google Chrome böngészővel jelenítem meg az oldalt. Az oldal megnyitásakor a kezdőlap (Home) fogad minket, ahol az oldal tetején található a navigációs sáv. A navigációs sáv segítségével elérhetjük az alábbi oldalakat: Home (kezdőlap), Build Form (űrlap létrehozási oldal), Login (bejelentkezési oldal), és Sign Up (regisztrációs oldal).



4. ábra. A kezdőoldal az asztali felhasználói felületen

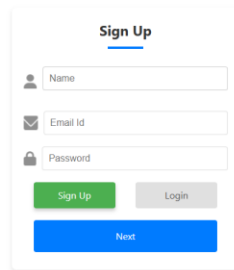




5. ábra. A kezdőoldal a mobil felületen

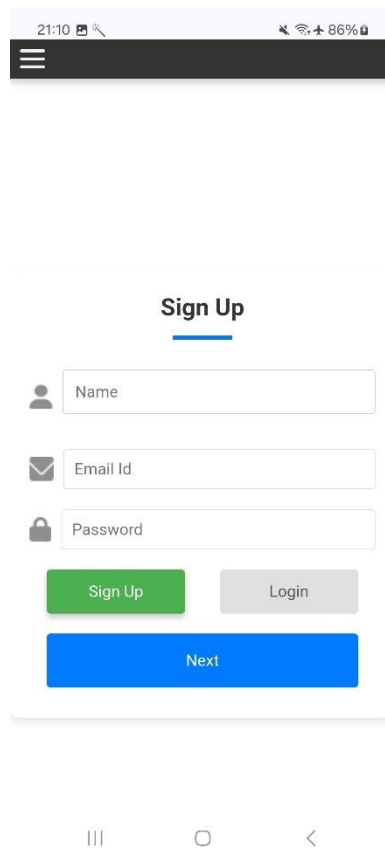
A kezdőlap után, ha még nincs felhasználói fiókunk, a regisztrációs (Sign Up) oldalon kell létrehoznunk egyet. Ehhez meg kell adnunk egy egyedi felhasználónevet, egy e-mail címet, valamint egy jelszót. A jelszónak legalább 4 karakter hosszúnak kell lennie, és tartalmaznia kell egy számot. Az e-mail címnek nincs funkcionális jelentősége.

Az oldalon látható a *Sign Up* és a *Login* gomb. Ezek csak oldal navigáló gombok, és a zöld színű gomb azt jelzi, hogy éppen melyik oldalon van a felhasználó.



The image shows a desktop browser window displaying a 'Sign Up' form. The form is centered and has a white background with a subtle shadow. It contains three input fields: 'Name' with a person icon, 'Email Id' with an envelope icon, and 'Password' with a lock icon. Below the fields are two buttons: a green 'Sign Up' button and a grey 'Login' button. At the bottom of the form is a large blue 'Next' button. The title 'Sign Up' is at the top of the form, underlined.

6. ábra A regisztrációs oldal asztali felületen



The image shows a mobile phone screen displaying the same 'Sign Up' form. The status bar at the top shows the time 21:10, signal strength, Wi-Fi, and 86% battery. A black hamburger menu icon is on the left. The form itself is centered and has a white background. It contains the same three input fields and buttons as the desktop version. The title 'Sign Up' is at the top of the form, underlined. At the bottom of the screen are three mobile navigation icons: a hamburger menu, a circle, and a back arrow.

7. ábra A regisztrációs oldal mobilon

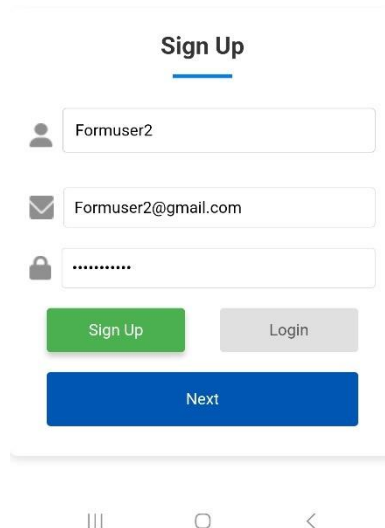
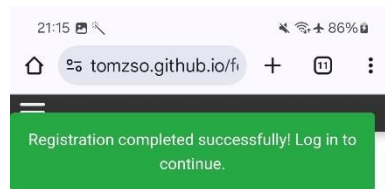
A kitöltött három adat után nyomjunk rá a kék színű Next gombra. Ezután várjuk meg az alkalmazás választ. Kis várakozás után az oldal tetején meg fog jelenni a válasz.

A zöld szín a sikerességről tájékoztat, míg a piros színű üzenet a sikertelenség okát mutatja.

A zöld üzenet után várjunk egy 1 vagy 2 másodpercet, mert a rendszer automatikusan átnavigálja a felhasználót a *Login* oldalra.

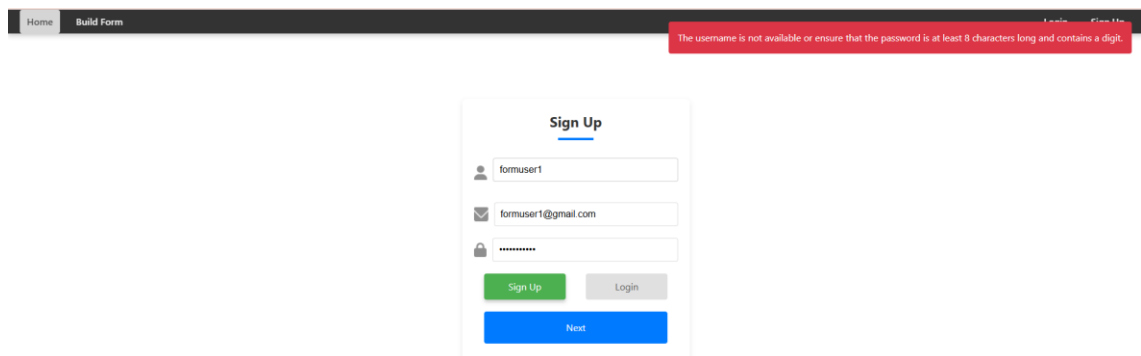
The screenshot displays a web application interface. At the top, a dark navigation bar contains links for 'Home', 'Build Form', 'Login', and 'Forgot My Password'. A green notification banner at the top right states 'Registration completed successfully! Log in to continue.' Below this, a 'Sign Up' form is centered. The form includes three input fields: a username field with 'formuser1', an email field with 'formuser1@gmail.com', and a password field with masked characters. Below the inputs are two buttons: a green 'Sign Up' button and a grey 'Login' button. At the bottom of the form is a blue 'Next' button.

**8. ábra. Sikeres regisztráció asztali felületen**

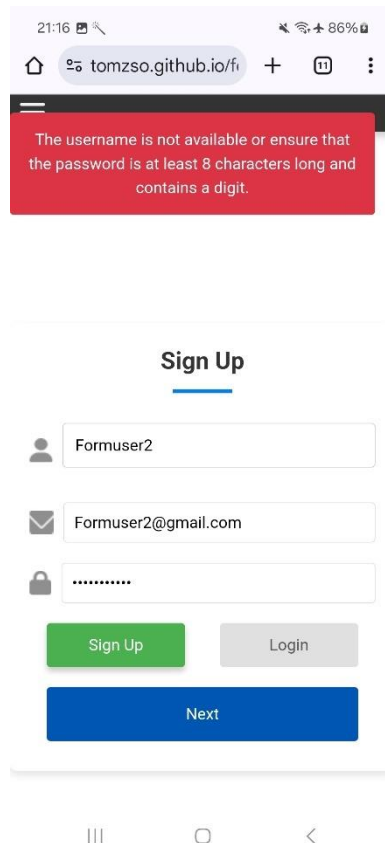


9. ábra. Sikeres regisztráció mobil felületen

Ha megpróbáljuk újra regisztrálni ugyanazokkal az adatokkal, hibát kapunk.

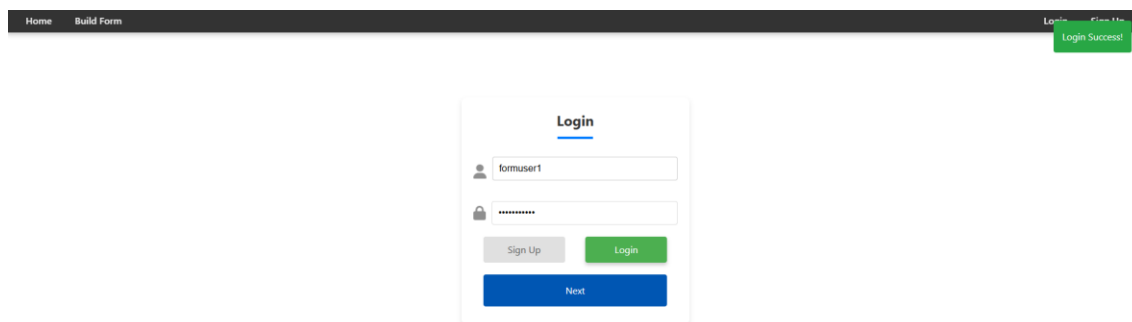


10. ábra. Sikertelen regisztráció asztali felületen

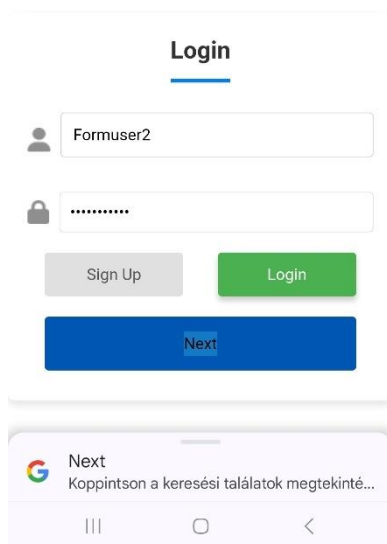
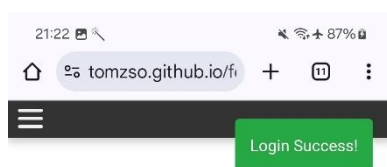


**11. ábra. Sikertelen regisztráció mobil felületen**

A bejelentkezés (Login) oldalára lépve újra megadjuk az adatokat, habár a regisztráció során automatikusan átnavigál minket ide. A navigálás során a beírt adatok nem vesznek el, tehát a felhasználónevet és a jelszót már kitöltve találjuk, ha az imént regisztráltunk. Ezért elegendő, ha csak a *Next* gombra kattintunk. Várjuk meg a zöld jelzést, és akkor, körülbelül két másodpercen belül, átnavigál a bejelentkezettek kezdőlapjára.



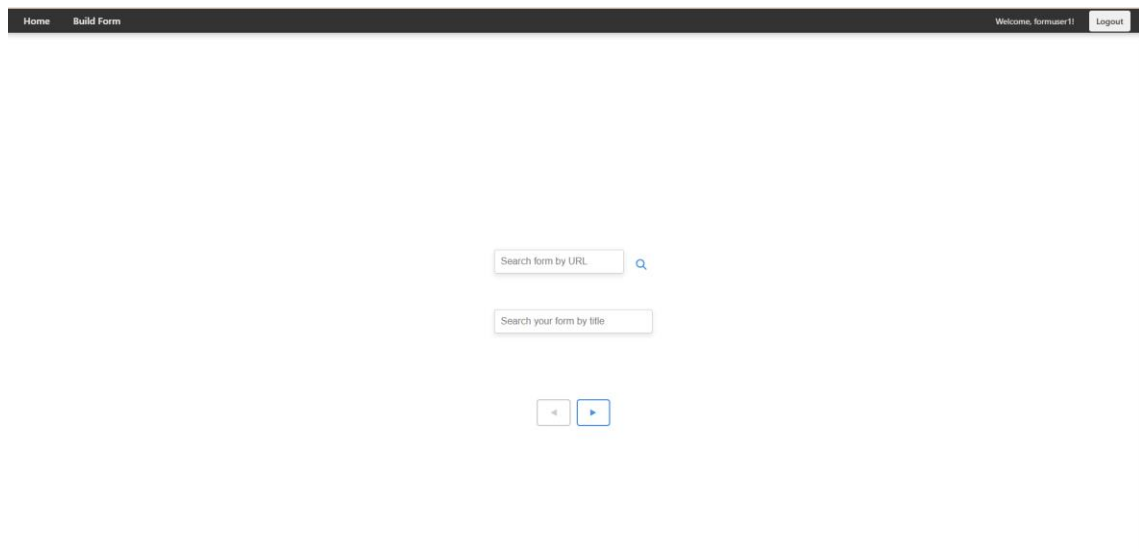
12. ábra. Sikeres bejelentkezés asztali felületen



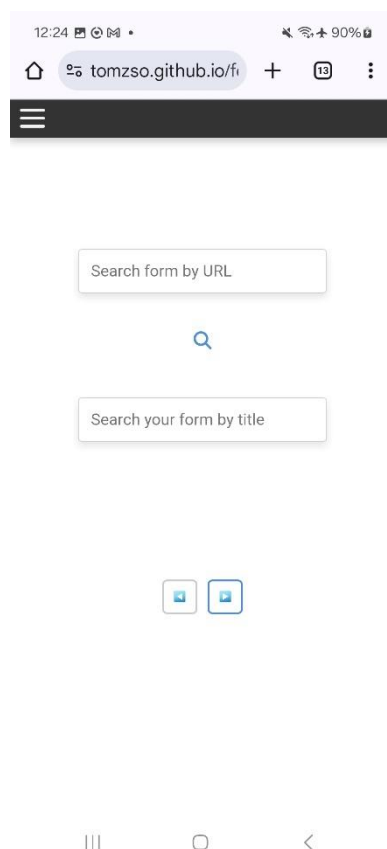
13. ábra. Sikeres bejelentkezés mobil felületen

A bejelentkezést követően a felhasználói kezdőlap (Home) fogad minket. Itt lehetőség van meglévő űrlapokra keresni az első adatbeviteli mező segítségével, valamint saját, már létrehozott űrlapokra is rákereshetünk a következő mezőben.

Az alábbi képen egy felhasználó által nemrég létrehozott űrlap látható:

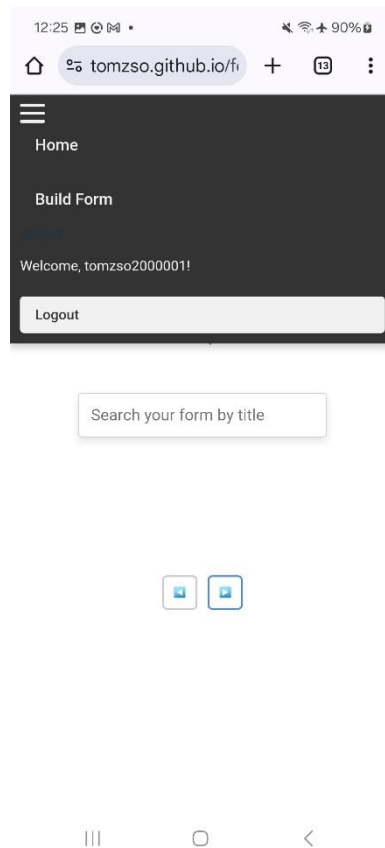


**14. ábra. Bejelentkezettek kezdőlapja asztali felületen**



**15. ábra. Bejelentkezettek kezdőlapja mobil felületen**

Az űrlap létrehozásakor a menüsorban található *Build Form* gombra kell kattintani. Mobil platformon a menüsor megjeleníthető a bal felső sarokban található, hamburger ikonhoz hasonló gomb segítségével.



**16. ábra. Menüsáv mobilon**

Az űrlapkészítő oldal így néz ki:



The desktop interface features a dark header bar with 'Home' and 'Build Form' on the left, and 'Welcome, formuser1!' and 'Logout' on the right. The main content area has a white background. At the top right is an orange 'Form' button. Below it, the text 'My title' is followed by a small edit icon. Further down, 'My description' is also followed by an edit icon. A section titled 'Add more question below:' contains a text input field with the placeholder 'Type your next question...'. Below this is a checkbox labeled 'This question is required'. Underneath are three green buttons: 'Textbox', 'Radio Buttons', and 'Checkbox'. A blue 'Save Question' button is at the bottom of this section. A final blue 'Save Form' button is located at the bottom right of the main content area.

17. ábra. Az űrlap létrehozási oldal asztali felületen

The mobile interface is shown within a browser window with a status bar at the top displaying '21:44', signal strength, Wi-Fi, and '88%' battery. The app's header is dark with a hamburger menu icon on the left and an orange 'Form' button on the right. The content area is white. It includes 'My title' and 'My description' with edit icons. The 'Add more question below:' section features a text input field, a 'This question is required' checkbox, and three green buttons: 'Textbox', 'Radio Buttons', and 'Checkbox'. A blue 'Save Question' button is at the bottom of this section. A blue 'Save Form' button is at the bottom of the screen. Mobile OS navigation icons (three vertical bars, a circle, and a back arrow) are visible at the very bottom.

18. ábra. Az űrlap létrehozási oldal mobil felületen

Az űrlap készítése és szerkesztése során lehetőség van az űrlap címét és leírását szerkeszteni a ceruza ikon segítségével. A narancssárga *Form* gombbal meghatározhatjuk, hogy milyen típusú legyen az űrlap. Alapértelmezetten kérdőív

(Form) van beállítva, de lehet kvíz (Quiz) is. Az *Add more question below:* rész alatt hozhatjuk létre a konkrét kérdéseket. A *Type your next question* mezőben fogalmazzuk meg a kérdést.

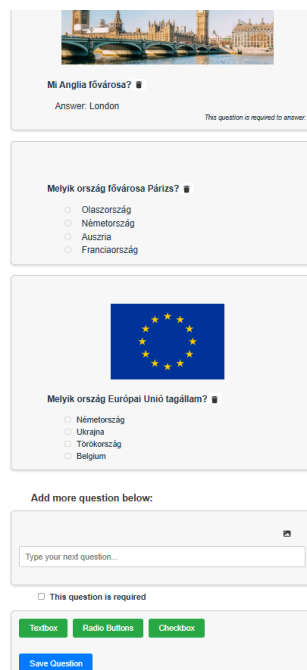
A *This question is required* melletti jelölőnégyzettel meghatározhatjuk, hogy kötelező-e kitölteni az adott kérdést az űrlapot kitöltő felhasználóknak. A *Textbox*, *Radio buttons* és *Checkbox* gombokkal beállíthatjuk, milyen jellegű választ várunk. A kép ikon segítségével képet csatolhatunk a kérdéshez.

A *Save Question* menti a kérdést, míg a *Save Form* az egész űrlapot menti el. A válaszopciók megadása nem kötelező a *Form* típusú űrlapoknál.

*Checkbox* és *Radio button* típusú kérdéseknél a válaszopciók számát a plusz és mínusz gombokkal állíthatjuk be (legalább 1 és legfeljebb 4 opció adható meg).

Egy kérdés létrehozása után a cím melletti kuka ikonnal törölhetjük az egész kérdést a válaszopcióival együtt.

Az alábbi képen egy 3 kérdésből álló űrlapot hoztam létre, és elmentettem az egészet:

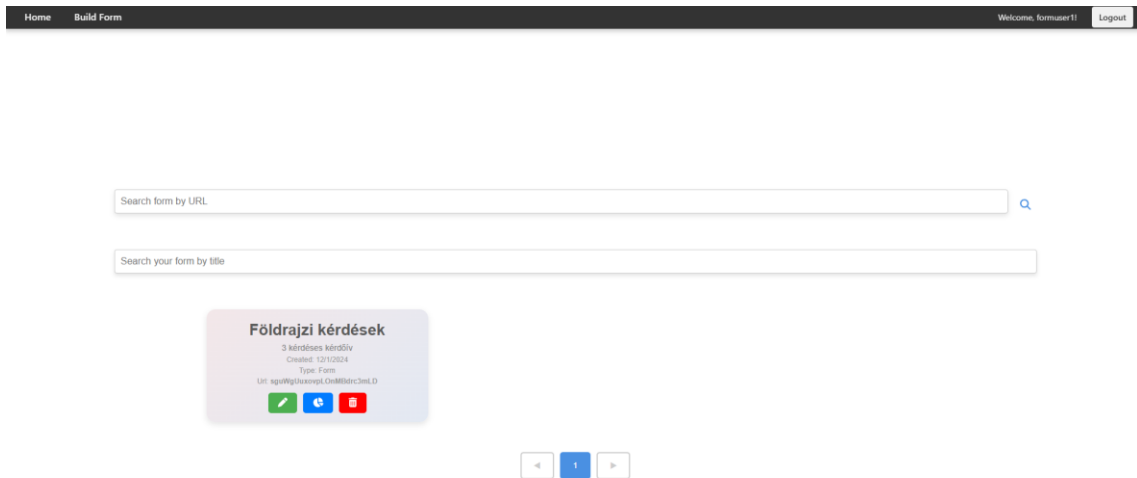


The screenshot displays a quiz creation interface. It contains three question cards. The first card has a picture of London and asks 'Mi Anglia fővárosa?' (What is the capital of England?) with a text input field and a 'This question is required to answer' checkbox. The second card asks 'Melyik ország fővárosa Párizs?' (Which country's capital is Paris?) with radio button options for Olaszország, Németország, Ausztria, and Franciaország. The third card features the European Union flag and asks 'Melyik ország Európai Unió tagállam?' (Which country is a member of the European Union?) with radio button options for Németország, Ukrajna, Törökország, and Belgium. Below the questions is a section titled 'Add more question below:' with a text input field for the next question, a 'This question is required' checkbox, and three buttons: 'Textbox', 'Radio Buttons', and 'Checkbox'. At the bottom is a blue 'Save Question' button.

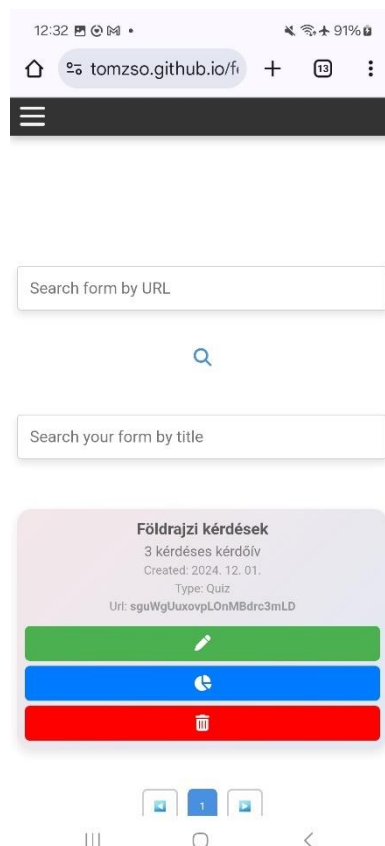
**19. ábra. 3 kérdést tartalmazó űrlap létrehozása asztali felületen**

A kezdőoldalra visszatérve már láthatjuk a korábban létrehozott űrlap információit. A legfontosabb információ az URL-kód, amely segítségével más felhasználók elérhetik az űrlapot. A ceruza ikonra kattintva szerkeszthetjük a kérdőívet, a kék kördiagram ikonra

kattintva pedig megtekinthetjük az űrlap válaszainak statisztikáit. A piros kuka ikonnal törölhetjük az egész kérdőívet.



20. ábra. A létrehozott űrlap megjelenik a kezdőlapon asztali felületen



21. ábra. A létrehozott űrlap megjelenik a kezdőlapon mobil felületen

A létrehozott kérdőívet minden felhasználó, beleértve annak létrehozóját is, csak egyszer tudja kitölteni. A kezdőoldalon lehet megnyitni az URL-t.

A *Textbox* típusú kérdéseknél a válaszokat szöveggként kell begépelni, míg a *Radio* és *Checkbox* típusú kérdéseknél egyszerűen be kell pipálni a megfelelő gombokat.

22:32 93%

### Földrajzi kérdések

3 kérdéses kérdőív

1. Mi Anglia fővárosa? \*

London

2. Melyik ország fővárosa Párizs?

☐ Olaszország

☒ Franciaország

☐ Németország

☐ Ausztria

3. Melyik ország Európai Unió tagállam?

☐ Ukrajna

☐ Törökország

☒ Németország

☒ Belgium

Back Next Submit

22. ábra. A kérdőív kitöltése mobil felületen

A kérdőív kitöltése után a *Submit* gombra kattintva a válaszok elmentésre kerülnek. Ezt követően a kérdőívet már nem lehet újra kitölteni. A gombok és a beviteli mezők nem reagálnak, és a rendszer halvány szürke színnel jelzi ezt.

22:40 93%

1. Mi Anglia fővárosa? \*

London

This question is required to answer.

2. Melyik ország fővárosa Párizs?

☐ Olaszország

☒ Franciaország

☐ Németország

☐ Auszria

3. Melyik ország Európai Unió tagállam?

☐ Ukrajna

☐ Törökország

☒ Németország

☒ Belgium

Back Next

23. ábra. A kitöltött sima kérdőív megjelenítése mobil felületen

A kvíz (*Quiz*) jellegű kérdőíveknél a program azonnal kiértékeli a válaszokat, és megadja a helyes megoldásokat.

12:23 90%

tomzso.github.io/fi

## Földrajzi kérdések

3 kérdéses kérdőív

Your Score: 2.5 out of 3



1. Mi Anglia fővárosa? \*

London

Correct answer: London

This question is required to answer.

field is required.

2. Melyik ország fővárosa Párizs?

☐ Olaszország

☒ Franciaország

☐ Németország

☐ Auszria

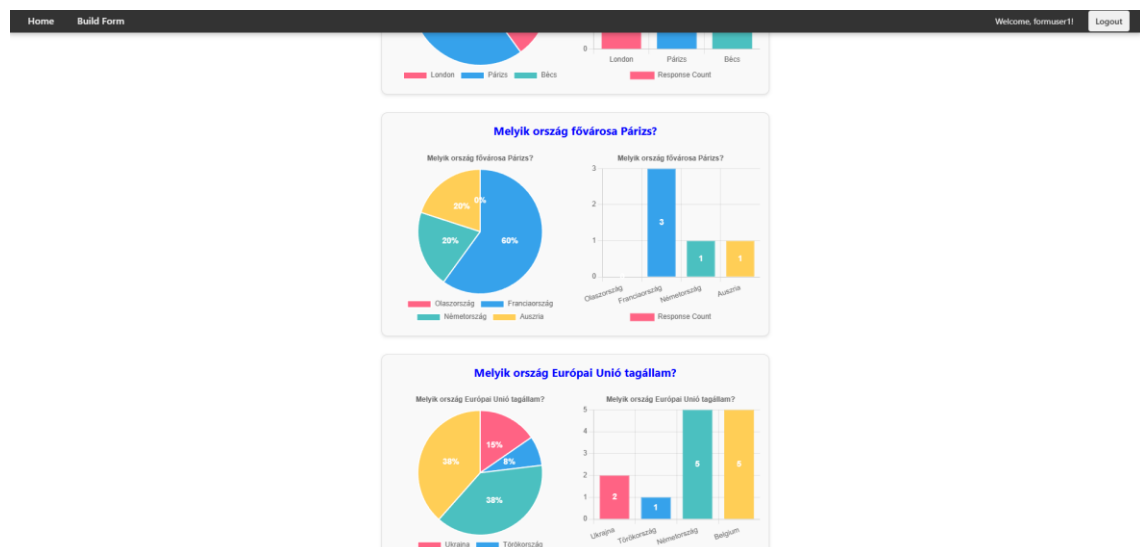
Correct answer: Franciaország

24. ábra. A kitöltött kvíz jellegű kérdőív megjelenítése mobil felületen

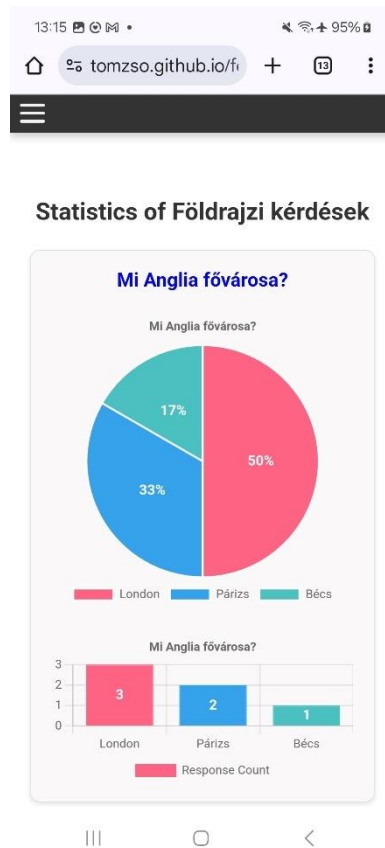
Ha nem töltöttük ki az összes kérdést, és úgy próbáljuk elküldeni a kitöltött űrlapot, akkor a rendszer visszajelez, és nem küldi el. Így csak akkor tudja elküldeni ha minden kötelező kérdésre válaszolt.

25. ábra. A kötelezően kitöltendő kérdés kihagyása során a rendszer visszajelez

A továbbiakban létrehoztam néhány újabb felhasználót, és kitöltöttem mindegyikkel az űrlapot különböző válaszokkal. Ezt követően bejelentkeztem az űrlapot létrehozó felhasználó fiókjába, és megnéztem a válaszok statisztikáit.



26. ábra. Egy űrlap válaszok statisztikái asztali felületen



27. ábra. Egy űrlap válaszok statisztikái mobil felületen



## 6. Összegzés

A program a felhasználók kezelését valósítja meg. A felhasználóknak joguk van űrlapokat létrehozni, módosítani és kitölteni. A belépés után a kezdőoldalon a felhasználók a keresősáv segítségével meg tudják találni a kívánt kérdőívet. Ezen az oldalon elérhetik a már létrehozott kérdőíveket is.

Az alkalmazásban űrlapokat lehet létrehozni a menüsáv alatt. A kérdések létrehozása során sok mindent lehet testreszabni. Be lehet állítani a címet, a leírást és a típust is. Láthatjuk, hogy Quiz esetén a rendszer ellenőrzi a megoldásokat, és kiértékeli a válaszokat. Forms esetén csak a bejelölt válaszokat mutatja. Az egyes kérdésekhez képet is lehet csatolni, és be lehet állítani a válasz opcióit (textbox, checkbox, radiobox). Be lehet állítani azt is, hogy a kérdést kötelező legyen kitölteni. Ha egy kérdés nem tetszik, azt törölni is lehet.

A kezdőoldalon lehetőség van arra, hogy a meglévő kérdőíveket töröljük vagy szerkesszük. Emellett van egy statisztika gomb is, amely megmutatja az egyes kérdésekhez tartozó válaszok arányait. Ezt kördiagram és oszlopdiagram formájában szemlélteti.

## 7. Továbbfejlesztési lehetőségek

A program fejlesztésének végéhez közeledve számos dolgot leegyszerűsítettem, és néhány dolgon még lehetne javítani.

A szoftverfejlesztés során nem volt időm teszteket írni. A teszt kódokkal a kódbázis karbantarthatóbbá és minőségibbé válna.

A webalkalmazás felhasználói felületét jobban és egyszerűbben is megvalósíthattam volna. A kérdőív QR-kóddal történő hozzáférés sokat javítana a felhasználói élményen. Az idő szűkössége miatt a használhatóságra törekedtem, ugyanakkor a felület javításra és továbbfejlesztésre szorul.

Ezenkívül a programból hiányzik néhány olyan funkció, amelyeket bár fontolóra vettem, de végül nem implementáltam. Ilyenek például az időzítő beállítása a kvízkérdésekhez, vagy az, hogy a kérdőívet többször is ki lehessen tölteni, vagy esetleg az, hogy minden opció válasznál be lehet állítani képet is.

## 8. Irodalomjegyzék

- [1] Biran Cline: Best Practices for Layered Architecture Development  
<https://blog.ndepend.com/layered-architecture-solid-approach/>
- [2] How Does That Help Me?: <https://www.brcline.com/blog/introduction-to-the-layered-architecture-n-tier-architecture>
- [3] Create the EntityFramework Core DbContext <https://codingsonata.com/build-restful-apis-using-asp-net-core-and-entity-framework-core/>
- [4] Tapes Mehta: Why RESTful APIs? <https://dev.to/wirefuture/how-to-build-restful-apis-with-aspnet-core-8-j5>
- [5] Evandro Gomes: I recommend you to use a code editor such as Visual Studio Code to develop the API. <https://www.freecodecamp.org/news/an-awesome-guide-on-how-to-build-restful-apis-with-asp-net-core-87b818123e28/>
- [6] Why Is React.js So Popular? 6 Key Factors Behind Its Wide Adoption  
<https://dev.co/react/why-is-react-so-popular>
- [7] Connect to and query Azure SQL Database using .NET and the Microsoft.Data.SqlClient library <https://learn.microsoft.com/en-us/azure/azure-sql/database/azure-sql-dotnet-quickstart?view=azuresql&tabs=visual-studio%2Cpasswordless%2Cservice-connector%2Cportal>
- [8] Tutorial: Deploy an ASP.NET app to Azure with Azure SQL Database  
<https://learn.microsoft.com/en-us/azure/app-service/app-service-web-tutorial-dotnet-sqldatabase>
- [18] Cloudinary felhő: <https://cloudinary.com/>

## 9. Kódrészletek jegyzéke

[9] A DbContext osztály implementálása, ahol kialakítottam az egyes táblák közötti kapcsolatokat.

```
public class ApplicationDbContext : IdentityDbContext<AppUser>
{
    public ApplicationDbContext(DbContextOptions options) : base(options)
    {
    }
    // Defining tables
    public DbSet<Form> Forms { get; set; }
    public DbSet<FormField> FormFields { get; set; }
    public DbSet<FormFieldOption> FormFieldOptions { get; set; }
    public DbSet<FormResponse> FormResponses { get; set; }
    public DbSet<FieldResponse> FieldResponses { get; set; }

    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        base.OnModelCreating(modelBuilder);

        // Defining relationships
        modelBuilder.Entity<Form>()
            .HasOne(f => f.User)
            .WithMany(u => u.Forms)
            .HasForeignKey(f => f.UserId);

        modelBuilder.Entity<FormField>()
            .HasOne(ff => ff.Form)
            .WithMany(f => f.FormFields)
            .HasForeignKey(ff => ff.FormId);

        modelBuilder.Entity<FormResponse>()
            .HasOne(fr => fr.Form)
            .WithMany(f => f.FormResponses)
            .HasForeignKey(fr => fr.FormId);

        modelBuilder.Entity<FormResponse>()
            .HasOne(fr => fr.User)
            .WithMany(u => u.FormResponses)
            .HasForeignKey(fr => fr.UserId)
            .IsRequired(false);

        modelBuilder.Entity<FieldResponse>()
            .HasOne(fr => fr.FormResponse)
            .WithMany(fr => fr.FieldResponses)
            .HasForeignKey(fr => fr.ResponseId)
            .OnDelete(DeleteBehavior.Restrict);

        modelBuilder.Entity<FieldResponse>()
            .HasOne(fr => fr.FormField)
            .WithMany(ff => ff.FieldResponses)
            .HasForeignKey(fr => fr.FieldId)
            .OnDelete(DeleteBehavior.Cascade);
    }
}
```

```

        modelBuilder.Entity<FormFieldOption>()
            .HasOne(fo => fo.FormField)
            .WithMany(f => f.FormFieldOptions)
            .HasForeignKey(fo => fo.FieldId);

        // Seed roles
        List<IdentityRole> roles = new List<IdentityRole>
        {
            new IdentityRole { Name = "Admin", NormalizedName = "ADMIN" },
            new IdentityRole { Name = "User", NormalizedName = "USER" }
        };
        modelBuilder.Entity<IdentityRole>().HasData(roles);
    }

```

#### [10] Az AppUser osztály implementációja

```

public class AppUser: IdentityUser
{
    // Navigation Properties
    public List<Form>? Forms { get; set; }
    public List<FormResponse>? FormResponses { get; set; }
}

```

#### [11] A Form osztály implementációja

```

public class Form
{
    public int Id { get; set; }
    public string Title { get; set; } = string.Empty;
    public string Description { get; set; } = string.Empty;
    public string Status { get; set; } = string.Empty;
    public string Type { get; set; } = string.Empty;
    public DateTime CreatedAt { get; set; } = DateTime.UtcNow;

    // URL Property
    public string Url { get; set; }

    // Foreign Key to User (IdentityUser)
    public string? UserId { get; set; }

    // Navigation Properties
    public AppUser? User { get; set; }
    public List<FormField>? FormFields { get; set; }
    public List<FormResponse>? FormResponses { get; set; }
}

```

#### [12] A FormField osztály implementációja

```

public class FormField
{
    public int Id { get; set; }
    public int FormId { get; set; }
    public string Label { get; set; } = string.Empty;
}

```

```

        public string FieldType { get; set; } = string.Empty;
        public string ImageUrl { get; set; } = string.Empty;
        public bool Required { get; set; } = false;
        //public string Options { get; set; } // This can be deprecated
        public int Order { get; set; } = 0;

        // Navigation Properties
        public Form? Form { get; set; }
        public List<FieldResponse>? FieldResponses { get; set; }
        public List<FormFieldOption>? FormFieldOptions { get; set; } // New
relationship
    }

```

#### [13] A FormFieldOption osztály implementációja

```

public class FormFieldOption
{
    public int Id { get; set; }
    public int FieldId { get; set; }
    public string OptionValue { get; set; } = string.Empty;
    public int Order { get; set; } = 0;
    public bool IsCorrect { get; set; } = false;
    public int ResponseCount { get; set; } = 0;

    // Navigation Property
    public FormField? FormField { get; set; }
}

```

#### [14] A DBContext osztály implementációja

```

public class FormResponse
{
    public int Id { get; set; }
    public int FormId { get; set; }
    public string UserId { get; set; } = string.Empty;
    public DateTime SubmittedAt { get; set; } = DateTime.UtcNow;

    // Navigation Properties
    public Form? Form { get; set; }
    public AppUser? User { get; set; }
    public List<FieldResponse>? FieldResponses { get; set; }
}

```

#### [15] A DBContext osztály implementációja

```

public class FieldResponse
{
    public int Id { get; set; }
    public int ResponseId { get; set; }
    public int FieldId { get; set; }
    public string Value { get; set; } = string.Empty;

    // Navigation Properties
    public FormResponse? FormResponse { get; set; }
    public FormField? FormField { get; set; }
}

```

## [16] Példa a repository minta alkalmazására

### *Repository interfész:*

```
public interface IFormRepository
{
    Task<List<Form>> GetAllAsync();
    Task<List<Form>> GetByUserIdAsync(AppUser user, FormQueryObject
formQueryObject);
    Task<Form> GetByIdAsync(int id);
    Task<Form> CreateAsync (Form form);
    Task<Form?> UpdateAsync (int id, Form form);
    Task<Form?> DeleteAsync (int id);
    Task<bool> FormExists(int id);
    Task<Form?> GetByUrlAsync(string url);
}
```

### *Repository implementáció:*

```
public class FormRepository : IFormRepository
{
    private readonly ApplicationDBContext _context;
    public FormRepository(ApplicationDBContext context)
    {
        _context = context;
    }

    public async Task<List<Form>> GetAllAsync()
    {
        return await _context.Forms
            .Include(f => f.FormFields)
            .ThenInclude(ff => ff.FormFieldOptions)
            .ToListAsync();
    }
}
```

### *Controller:*

```
[Route("api/form")]
[ApiController]
public class FormController : ControllerBase
{
    private readonly IFormRepository _formRepository;
    private readonly UserManager<AppUser> _userManager;
    public FormController(UserManager<AppUser> userManager,
        IFormRepository formRepository)
    {
        _formRepository = formRepository;
        _userManager = userManager;
    }

    [HttpGet("all")]
    public async Task<IActionResult> GetAll()
    {

```

```

        var forms = await _formRepository.GetAllAsync();
        var formDtos = forms.Select(form => form.ToFormDto());
        return Ok(formDtos);
    }
}

```

[17] A connection string kód

```

"DefaultConnection":
"Server=tcp:formwebserverserver.database.windows.net,1433;Initial
Catalog=formdatabase;Persist Security Info=False;User
ID=lengtho01;Password={password};
MultipleActiveResultSets=False;Encrypt=True;TrustServerCertificate=False;C
onnection Timeout=30;"

```

[19] A hibás oldalhoz tartozó kód részlet

Egyszerű példa *JSX* fájlra:

```

import React from "react";
import "../PageNotAvailable.css";
export const PageNotAvailable = () => {
    return (
        <div className="page-not-available">
            <h2>Page is not available</h2>
        </div>
    );
};

```

Egyszerű példa *CSS* fájlra:

```

.page-not-available {
    display: flex;
    justify-content: center;
    align-items: center;
    text-align: center;
    font-size: 1rem;
    font-family: Arial, sans-serif;
    font-weight: normal;
    background-color: #f8f9fa;
    color: #333;
    width: 100%;
    min-height: 100vh; /* Ensures it spans the full height of the viewport */
    padding-top: 60px; /* Adds padding to avoid overlap with a fixed navbar */
    /*
    box-sizing: border-box; /* Includes padding in the total height */
}

```

[20] Példa egy űrlap adatainak betöltéséhez

```

export const getFormByUrl = async (token,url) => {
    let link = `${BASE_URL}/url/${url}`;
    return await getApi(token,link);
}

```



```

export const getApi = async (token, url) => {
  try {
    const response = await fetch(url, {
      method: 'GET',
      headers: {
        'Authorization': `Bearer ${token}`
      }
    });

    if (!response.ok) {
      const contentType = response.headers.get('Content-Type');
      let errorBody;

      if (contentType && contentType.includes('application/json')) {
        errorBody = await response.json();
      } else if (contentType && contentType.includes('text/plain')) {
        errorBody = await response.text();
      } else {
        errorBody = await response.text();
      }

      console.error('Failed get:', errorBody);
      return {
        success: false,
        message: errorBody
      };
    }

    const json = await response.json();
    return {
      success: true,
      data: json
    };
  } catch (error) {
    console.error('Get error:', error);
    return {
      success: false,
      message: error.message || error.toString()
    };
  }
};

```

[21] Backend forráskódja. <https://github.com/tomzso/form-backend>

[22] Frontend forráskódja. <https://github.com/tomzso/form>