**Szegedi Tudományegyetem**

**Informatikai Intézet**

**SZAKDOLGOZAT**

**Mészáros Márton**

**2025**

**Szegedi Tudományegyetem**

**Informatikai Intézet**

**Interaktív weboldal fejlesztése falmászó közösség támogatására**

Szakdolgozat

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Készítette: |  | Témavezető: |  |
|  | **Mészáros Márton** |  | **Dr. Márkus András** |  |
|  | informatika szakos hallgató |  | egyetemi adjunktus |  |

Szeged

2025

# Feladatkiírás

Szakdolgozatom témája egy olyan weboldal létrehozása a cél, amely a falmászás sportjára épül, és elsődlegesen egy Szegeden működő mászótermet, valamint az ott elérhető falmászó lehetőségeket mutatja be. Az oldal tartalmazná a terem részletes leírását, a megmászható utakat, valamint egy funkciót, amely lehetővé teszi a felhasználók számára, hogy saját utakat töltsenek fel. Ehhez egy beépített képszerkesztő is rendelkezésre állna, amely segítene az utak vizuális megjelenítésében. Az egyes utakhoz hozzászólásokat lehetne fűzni, értékelni lehetne őket, és a nehézségi szintjüket is módosítani lehetne a közösségi visszajelzések alapján. Az oldalon egy beépített térképes megjelenítő is helyet kapna, amely a legrövidebb utat mutatná meg a teremhez. A kép- és hozzászólásfeltöltés kizárólag regisztrált felhasználók számára lenne elérhető, biztosítva ezzel a közösség szűrését és a tartalmak minőségét.

# Tartalmi összefoglaló

* A téma megnevezése:

Interaktív weboldal fejlesztése falmászó közösségi támogatására

* A megadott feladat megfogalmazása:

A weboldal, aminek célja, hogy segítse a mászókat egyfajta közösségi oldalként, ahol falmászó termeket és ottani falakat, valamint a falon lévő utakat megosszák egymással, ezzel segítve abban a döntésben, hogy a terem megfelelő- e.

* A megoldási mód:

A feladat megvalósítását egy modern, háromrétegű ASP.NET Core MVC alapú architektúra mentén végeztem, alkalmazva a Code First fejlesztési megközelítést. A perzisztencia réteget az Entity Framework Core (ORM) biztosítja egy SQLite adatbázis felett. A nagyméretű, vizuális adatok képek tárolására és gyors elérésére a Google Firebase Cloud Storage szolgáltatást implementáltam. A kliensoldali interaktivitást a Fabric.js JavaScript könyvtár beépítésével értem el, amely lehetővé teszi a mászóutak dinamikus, képalapú szerkesztését és a rajzolási állapotok kezelését. A felhasználói hitelesítést és a többszintű jogosultsági rendszert az ASP.NET Core Identity keretrendszerrel (Microsoft PasswordHasherrel) valósítottam meg

* Alkalmazott eszközök, módszerek:

Visual Studio 2022, ASP.NET Core, MVC, Razor View, C#, JavaScript, Entity Framework Core, SQLite, Google Firebase Storage, Fabric.js, Microsoft PasswordHasher, GitHub Flow

* Elért eredmények:

Létrehoztam egy működőképes, közösségi funkciókat támogató webalkalmazást (Csak mássz fel), amely kezeli a fő entitások (FalmászoHelyek, Falak, Utak) teljes CRUD műveletsorát. Kiemelt eredményként implementáltam egy Fabric.js alapú interaktív útvonalszerkesztőt (Undo/Redo funkcióval), valamint egy automatizált adatkezelési logikát, mely garantálja az adatbázis és a Firebase Cloud Storage konzisztenciáját (megakadályozva az árva fájlokat). A felhasználói adatok biztonságát a háromszintű szerepkör- alapú hozzáférés- szabályozás és a jelszavak biztonságos tárolása biztosítja. A fejlesztési folyamat és a kód karbantarthatóságát a GitHub Flow módszer alkalmazásával értem el.

* Kulcsszavak:

Falmászás, webalkalmazás, ASP.NET Core MVC, Fabric.js, Firebase Cloud Storage, adatintegritás

# Tartalomjegyzék

[Feladatkiírás 3](#_Toc215159049)

[Tartalmi összefoglaló 4](#_Toc215159050)

[Tartalomjegyzék 5](#_Toc215159051)

[1. Bevezetés 7](#_Toc215159052)

[1.1 Motiváció 7](#_Toc215159053)

[1.2. A weboldalam és a theCrag.com összehasonlítása 7](#_Toc215159054)

[1.3. A konkurensek analízise és a funkcionális rés 8](#_Toc215159055)

[2. Technikai háttér 10](#_Toc215159056)

[2.1. A fejlesztői környezet Microsoft Visual Studio 2022 10](#_Toc215159057)

[2.2. Entity Framework Core (EF Core) 10](#_Toc215159058)

[2.3. Razor View 12](#_Toc215159059)

[2.4. Kliensoldali technológiák 12](#_Toc215159060)

[2.5. Fabric.js 13](#_Toc215159061)

[2.6. Firebase Cloud Storage 14](#_Toc215159062)

[2.7. Az alkalmazás architektúrája és rétegződése 15](#_Toc215159063)

[2.7.1. Az MVC architektúra működése és a függőségkezelés 16](#_Toc215159064)

[2.8. Az adatbázis-struktúra és kapcsolatkezelés 16](#_Toc215159065)

[2.8.1. Adatvalidáció és adatintegritás 18](#_Toc215159066)

[2.9. Verziókövetés és a fejlesztési metodológia 18](#_Toc215159067)

[2.9.1. A GitHub Flow munkafolyamat 18](#_Toc215159068)

[2.9.2. A pull requestek szerepe a minőségbiztosításban 19](#_Toc215159069)

[2.10. Az alkalmazás futtatása 19](#_Toc215159070)

[2.11. Összegzés 19](#_Toc215159071)

[3. Az alkalmazás működése 21](#_Toc215159072)

[3.1. Felhasználói szerepkörök és jogosultságkezelés 21](#_Toc215159073)

[3.1.1. Jogosultságkezelés 21](#_Toc215159074)

[3.1.2. Felhasználó azonosítása 24](#_Toc215159075)

[3.2. Felhasználói műveletek 24](#_Toc215159076)

[3.2.1. Regisztráció 24](#_Toc215159077)

[3.2.2. Bejelentkezés 25](#_Toc215159078)

[3.2.3. Jelszó módosítás 26](#_Toc215159079)

[3.3. Adatbázis műveletek (CRUD) 26](#_Toc215159080)

[3.3.1. Az adatbázis felépítése 26](#_Toc215159081)

[3.3.2. Lekérdezési műveletek 27](#_Toc215159082)

[3.3.3. Létrehozási műveletek 28](#_Toc215159083)

[3.3.4. Módosítási műveletek 30](#_Toc215159084)

[3.3.5. Törlési műveletek 30](#_Toc215159085)

[3.4. Képkezelési és interaktív szerkesztési funkciók 32](#_Toc215159086)

[3.4.1 Interaktív szerkesztés és állapotkezelés 32](#_Toc215159087)

[3.4.2. Mászóutak képkezelése: szerkesztés és adatintegritás 34](#_Toc215159088)

[3.4.3. Adatintegritás és árva fájlok megelőzése (Törlés) 34](#_Toc215159089)

[3.5. Biztonsági mechanizmusok 35](#_Toc215159090)

[3.6. Külső szolgáltatások integrációja 37](#_Toc215159091)

[4. Összegzés 39](#_Toc215159092)

[Irodalomjegyzék 41](#_Toc215159093)

[Nyilatkozat 43](#_Toc215159094)

# 1. Bevezetés

## 1.1 Motiváció

A falmászással már egészen kis gyerekként megismerkedtem, de nem űztem mint sportot, csak nagyon alkalmi szinten foglalkoztam vele néhány évente egyszer-egyszer. Majd egyetemi tanulmányaim alatt felvettem, mint kötelező testnevelési kurzust, ahol megszerettem, mint sportot és elkezdtem járni az esti edzésekre. Ott megismert barátoktól és saját tapasztalatból született meg az a gondolat a fejemben, hogy ezzel a taméval kapcsolatban írjam meg a szakdolgozatomat. A weboldal egyik fő inspirálómondata az volt, hogy „milyen citromsárga nehézségű út volt ezen a falon legutóbb” mivel kéthetente újracetlizik az utakat azaz új utakat hoznak létre de az új út még nincs feljelölve, és nem emlékeznek arra, hogy hogy nézet ki a legutóbbi út. Ezen felül ha van saját utad, azt is meg tud adni, hogy mások is lássák anélkül, hogy a meglévő utak cetlijeibe bezavarna (ugyan olyan nehézségű és nem lehetne elkülöníteni a két utat egymástól). Ez a spraywall típusú falakon hasznos, ahol az utak nem azonos színű fogásokból állnak. Másik számomra fontos rész, hogy ha egy másik mászótermet szeretnék meglátogatni, de a terem elérhetőségein nem találok olyan képet, ami az aktuális állapotot mutatja be, de én kíváncsi vagyok arra, hogy jelenleg hogy néz ki a fal és milyen nehézségű utak vannak. Ezen problémák megoldására és, hogy a falmászók megoszthassák egymással az információkat, hoztam létre a weboldalt, ami egyfajta online közösségi térként működik, és nem csak száraz adatokat tárol, hanem személyes véleményeket és tapasztalatokat is. A weboldalam célja, hogy egyfajta közösségi teret hozzon létre, hogy a mászók meg tudjanak osztani egymással személyes tapasztalatokat, véleményeket egy adott útról, vagy a meglévő fogásokból egy teljesen új utat hozzanak létre ezzel is színesítve a terem kínálatát.

## 1.2. A weboldalam és a theCrag.com összehasonlítása

Mind a két weboldal célja, hogy listázza az összes sziklamászó és falmászó helyeket. A legfőbb különbség, hogy az általam fejlesztett weboldalon az utakhoz kell képet is társítani, míg itt nem feltétel csak egy opció. A Csak mássz fel weboldalon van az útnak leírása is nem csak a nehézség és a név van megjelenítve, valamint az is látszik, hogy ki hozta létre. A theCrag weboldalon külön lapon, de csak az egyes falszakaszra van hozzászólási, fórum felület, míg nálam minden egyes útra van hozzászólási lehetőség. Külön falszakaszokra nincs hozzászólási lehetőség, mert a véleményt az útnál elég közölni a hibát, mert úgy is az esetek túlnyomó részében az úttal kapcsolatban fogalmaznak meg kritikát. A theCrag- en be lehet jelölni, ha már megmásztad az adott utat, míg az általam fejlesztett weboldalon ez a funkció nem érhető el. Ezen a weboldalon lehet a mászáshoz kapcsolódó könyveket is vásárolni, míg az általam létrehozott weboldal csak a falmászó helyeket és ezekhez kapcsolódó falakat, utakat és az utakhoz kapcsolódó hozzászólásokat jeleníti meg. A theCrag-en meg lehet tekinteni a felhasználókat is és néhány alap információt is megjelenít róluk. Nálam csak az admin jogosultsággal rendelkező felhasználó fér hozzá a felhasználók listájához. A bejelentkezetett felhasználó a Csak mássz fel weboldalon csak a jelszavát tudja módosítani, a profiljához nem fér hozzá, míg a theCrag weboldalon hozzáfér a felhasználó a profiljához és különböző statisztikákat is lát, mint kedvencek, megmászott utak [1].

A theCrag weboldal nem csak a teremben található utakat jeleníti meg, hanem a szabadtéri mászásra alkalmas sziklafalakat is. Erre az általam fejlesztett weboldalam is alkalmas, nem kell semmi különleges dolgot csinálni, nem kell külön megjelölni, hogy ez szabadtéren van, tehát mindent ugyan úgy kell csinálni, mint a falmászótermeknél [1].

Összességében a theCrag weboldal egy teljes körűen, minden apróbb részletre is kiterjedő weboldal, míg az általam létrehozott weboldal az egyszerűségre és csak a falmászóhelyek és az ezekhez szorosan kapcsolódó adatok megjelenítésére és minimális interakciókra a felhasználok között lett kialakítva [1].

## 1.3. A konkurensek analízise és a funkcionális rés

A theCrag.com részletes összehasonlítását követően elengedhetetlen a piacon elérhető további, mászókat támogató weboldalak rövid elemzése is. Számos online platform elérhető, amelyek közös jellemzője, hogy elsődlegesen sziklafalakat vagy mászótermeket listáznak, azonban funkcionalitásukban eltérések mutatkoznak a támogatott mászástipusok és a tartalom részletessége tekintetében.

A 27 Crags platform [2] kizárólag szabadtéri sziklamászásra fókuszál mind bulder, trad és sport mászástípusokban egyaránt. Ezen a weboldalon az úthoz tartozik kép, nehézség típus, továbbá biztosított a hozzászólási lehetőség az utak véleményezésére.

A természetjáró [3] weboldal szélesebb körű célközönséget szólít meg (túrázók és sziklamászók), de a mászással kapcsolatos szekciójában szintén csak a szabadtéri sziklafalak vannak megjelenítve. Ez a platform hosszabb leírás keretén belül adja meg a szükséges alapinformációkat, és kizárólag a magyarországi sziklamászóhelyeről nyújt részletes magyar nyelvű leírást.

Ezen és további fent nem említett rendszerek elemzése megerősítette, hogy a meglévő megoldások erőteljesen a szabadtéri helyszínek listázására és a statikus leírásokra koncentrálnak. A másik már létező weboldalak/alkalmazások pedig a spraywall-okra fókuszálnak. Ebből a piaci áttekintésből egyértelműen kirajzolódik a funkcionalitási rés: a Csak mássz fel weboldal az által biztosít egyedi értéket, hogy elsősorban a beltéri utak vizuális és interaktív szerkesztésére fókuszál, egy szorosan vett közösségi magot kiszolgálva, ami a konkurens rendszerekből hiányzik.

# 2. Technikai háttér

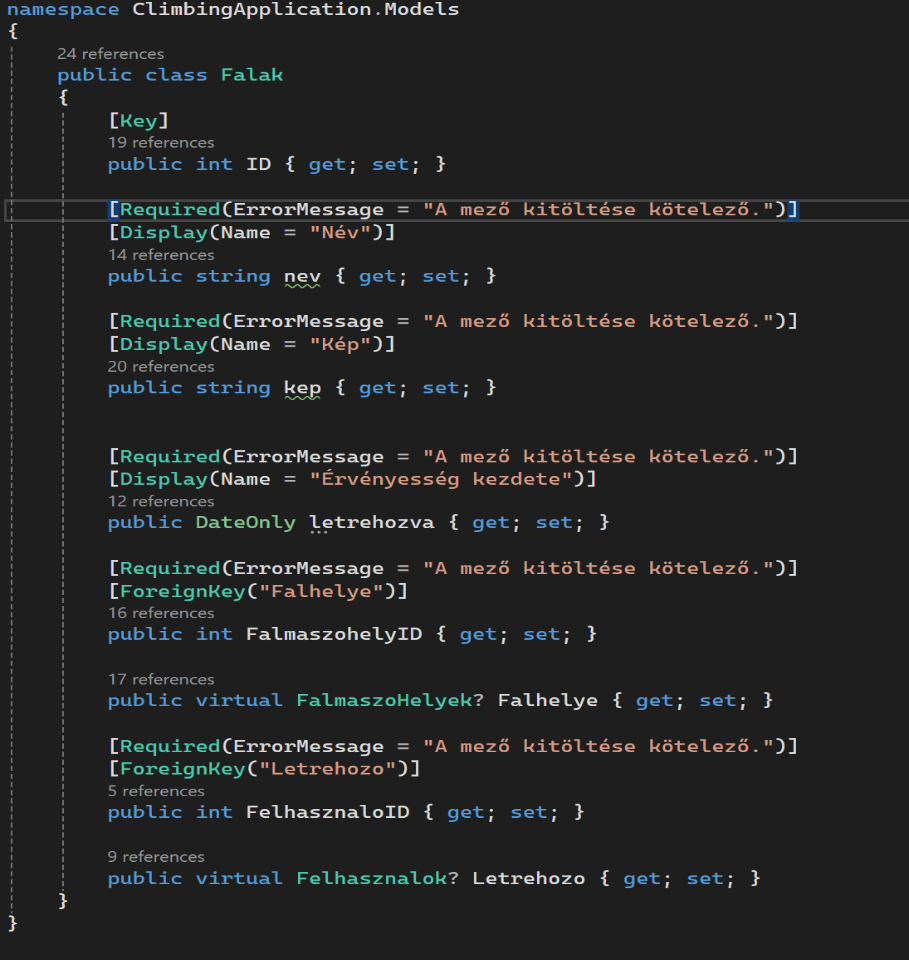
A projekt fejlesztése során egy modern, agilis, többrétegű architektúra és a Microsoft .NET Core ökoszisztémájára épülő technológiai stack került alkalmazásra. A választott eszközök és keretrendszerek - mit az ASP.NET Core, Entity Framework és a Fabric.js – kiválasztásának célja a magas szintű produktivitás biztosítása volt. A Dependency Injection elv alkalmazása és a GitHub Flow alapú verziókövetés biztosítja a kód minőségét és karbantarthatóságát. A rendszer Code First megközelítéssel készült, amely lehetővé teszi a modellek alapján történő rugalmas adatbázis-fejlesztést. A következő fejezetek részletezik a fejlesztés során alkalmazott eszközöket, elveket és a futtatási környezetet.

## 2.1. A fejlesztői környezet Microsoft Visual Studio 2022

A fejlesztéshez a Microsoft Visual Studio 2022 fejlesztői környezet került kiválasztásra [4]. A döntés hátterében az áll, hogy a VS 2022 biztosítja a legtöbb, beépített a fejlesztést segítő funkciót az ASP.NET Core projekthez. Ez a környezet teljes mértékben támogatja a Code First megközelítést, amely lehetővé teszi a modellek alapján történő adatbázis-generálást. Ezen túlmenően a scaffolding funkció automatikusan létrehozza a hozzá tartozó Controller és View részeket is, jelentősen gyorsítva a munkafolyamatot. A Visual Studio 2022 olyan modern funkciókat is kínál, mint a Hot Reload, amely lehetővé teszi a kód változásainak valós idejű alkalmazását az alkalmazás újraindítása nélkül, ezáltal drasztikusan gyorsítva a hibakeresést és a fejlesztési ciklust.

## 2.2. Entity Framework Core (EF Core)

Az Entity Framework Core (EF Core) egy nyílt forráskódú objektum-relációs leképező (ORM) keretrendszer a .NET platformra írt alkalmazások számára. A keretrendszer használatával a fejlesztők nagyobb termelékenyéget érnek el, mivel elkerülik a külön adatbázisspecifikus kódok írását. Az EF Core-t használók a tartományspecikikus osztályokra fókuszálnak, nem pedig az adatbázisban szereplő oszlopokra. Az EF Core használatával az adatbázis-kapcsolat kódok létrehozása feleslegessé válik, mivel az EF Core absztrakt szinten biztosítja azokat a fejlesztők számára. Nagyban csökkenti az adatspecifikus kódok méretét, és növeli a kód olvashatóságát is. Az Entity Framework végrehajtja a szükséges lekérdezéseket az adatbázisban, majd az eredményeket a domén modell példányaivá alakítja, így az alkalmazás feldolgozhatja azokat. Az EF Core néhány fontos jellemzője: platformfüggetlen; LINQ lekérdezéseket használ SQL lekérdezések helyett, és ezeket alkalmazza az adatmanipulációra is. Az EF Core az Ábra 2.1 – en látható modell osztályban lévő kódot veszi alapul az adatbázis adott táblájának létrehozására. A Code First megközelítés lehetővé teszi a kulcs és külső kulcshivatkozások, valamint a mezők kötelező és opcionális értékeinek beállítását közvetlenül a modellosztályokban [5].

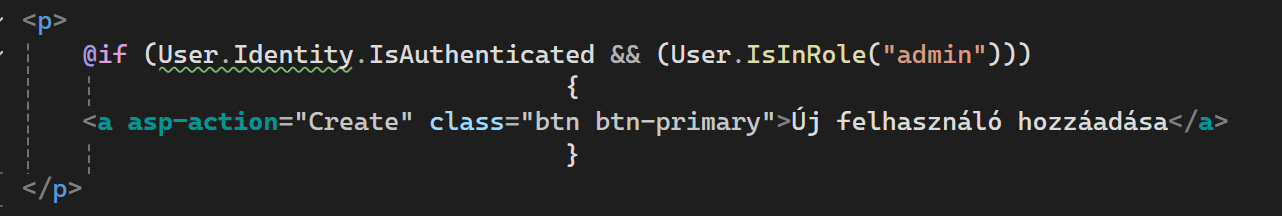


Ábra 2.1.Modell osztály példakód saját alkalmazásból

## 2.3. Razor View

A Razor egy jelölőszintaxis, amely az ASP.NET Core keretrendszerben a C# kód dinamikus beágyazására szolgál a weboldalakba. Ezt a technológiát a Microsoft fejlesztette ki, amely lehetővé teszi a C# és a HTML elemek zökkenőmentes kombinációját, ami a modern webalkalmazásokban a felhasználói felületeinek létrehozásának alapvető eszközévé válik. A Razor szintaxis leegyszerűsíti a szerveroldali logika és a kliensoldali megjelenítés elkülönítését, ezáltal növelve a kód olvashatóságát és karbantarthatóságát. Az ilyen nézetfájlok .cshtml kiterjesztést kapnak [6].

A @ karakter egy kulcsfontosságú karakter, ez jelzi a Razor számára, hogy ettől a karaktertől kezdve C# kódra kell váltani. Ez teszi lehetővé C# kifejezések, változók és utasítások beágyazását közvetlenül a HTML kódba. Implicit kifejezések esetén a @ szimbólum utáni C# kódot a Razor kiértékeli és HTML kimenetként jeleníti meg. Explicit kifejezések a @ utáni és kapcsos zárójelek közé írt C# kódot tartalmazzák, amely lehetővé teszi összetetteb műveletek beágyazását. A Razor teljes kódblokkot, például ciklusokat és feltételes utasításokat is értelmez. A Razor alapértelmezés szerint automatikusan kódolja a HTML és a C# kifejezések eredményét, ezzel kiküszöbölve a cross-site scripting (XSS) támadásokat. Az IHtmlContent típusú adatokat azonban kódolás nélkül jelennek meg, amely megengedi a szándékosan formázott HTML-tartalom beillesztését [7].



Ábra 2.2. Példa a Razora Ami egy feltételes utasítást tartalmaz nézet oldalon, hogy ki mit lásson

## 2.4. Kliensoldali technológiák

A modern webalkalmazások felhasználói felületei (UI) gyakran építenek a szerveroldali nézetmotorokon (mint Razor) túlmutató, fejlett kliensoldali technológiákra. A reszponzív megjelenés és a magas szintű interaktív felhasználói élmény biztosítása kiemelt cél. A felületek egységes kialakítását és a különböző eszközökön (például mobiltelefonokon és asztali gépeken) való megfelelő működést gyakran Bootstrap-hez hasonló CSS keretrendszerek biztosítják [8].

Az interaktív és dinamikus grafikus funkciók megvalósításához, különösen a canvas alapú alkalmazások esetében, speciális JavaScript könyvtárak alkalmazása szükséges, például a Fabric.js-t alkalmazzák. A projektben a Fabric.js könyvtár magas szintű objektumorientált API-t biztosít a grafikus elemek manipulálására [9]. Az általános viselkedésért, a DOM manipulációért és a dinamikus tartalomkezelésért a natív JavaScript felel, amely a webfejlesztés alapvető programozási nyelve.

## 2.5. Fabric.js

A fabric.js egy nyílt forráskódú JavaScript könyvtár, amely interaktív objektummodellt, biztosit a HTML5 canvas elem leegyszerűsítésére. A könyvtár magasabb szintű absztrakcióval kezeli a formákat, képeket és szövegeket, ezzel megkönnyíti az interaktív grafikus alkalmazások fejlesztését a böngészőkben. A fejlesztők ennek a segítségével nem közvetlenül az alacsonyabb szintű canvas API hívással dolgoznak, hanem egy jól meghatározott, tulajdonságokkal és metódusokkal rendelkező objektumokkal [9].

A könyvtár objektumorientált megközelítést alkalmaz minden grafikus elemet - például geometriai alakzat, kép vagy szöveg - önálló objektumként kezel. Ezen objektumok olyan attribútumokkal rendelkeznek, mint a pozíció, méret, szín, átlátszóság vagy a forgási szög. Az objektumok csoportosíthatók, így több elem együttesen is kezelhető, amely különösen hasznos a komplex grafikus jelenetek szerkesztésekor [9].

A fabric.js egyik legfontosabb ismérve az interaktív eseménykezelés. A könyvtár beépített eseménykezelővel lehetővé teszi a felhasználói műveletek (például egérkattintás, húzás vagy billentyűleütés) közvetlen kapcsolódását a grafikus elemekhez. Ezáltal a felhasználók intuitív módon mozgathatják, forgathatják vagy méretezhetik az objektumokat a vásznon, amely dinamikus és reszponzív felhasználói élményt biztosít [9].

A könyvtár nyílt forráskódú jellege, moduláris felépítése és aktív fejlesztői közössége révén stabil, jól dokumentált és rugalmas alapot, biztosít a modern, interaktív webes grafikai alkalmazások fejlesztéséhez [9].

## 2.6. Firebase Cloud Storage

A Google Cloud Storage for Firebase a Firebase platformon belül, a Google Cloud-alapú objektumtárolási szolgáltatás, amely kifejezetten a felhasználók által létrehozott tartalmak – például képek, videók vagy dokumentumok – biztonságos feltöltésére, tárolására és elérésére szolgál [10].

A rendszer a Google Cloud Storage infrastruktúrájára épül, ezáltal rendkívül jól skálázható, magas rendelkezésreállást biztosít, és képes petabájtos nagyságrendű adatok kezelésére is [11]. A fejlesztők a Firebase SDK-kon keresztül valósítják meg a fájlok feltöltését, letöltését vagy törlését anélkül, hogy közvetlenül a háttérrendszer kezelésével kellene foglalkozniuk [12].

A biztonság a szolgáltatás egyik legfontosabb eleme: a Firebase Authentication integráció révén a hozzáférések szabályozása felhasználói szinten is megvalósítható. A Firebase Security Rules segítségével pontosan meghatározható, hogy mely felhasználók milyen fájlokat érhetnek el, és milyen műveleteket hajthatnak végre rajtuk [13].

Az egyik legfőbb jellemzője az automatikus skálázhatóság: A Cloud Storage a terhelés növekedésével automatikusan alkalmazkodik, így nincs szükség manuális beavatkozásra [10].  
Egy másik jellemző a biztonság és titkosítás. Az adatok titkosítva vannak tárolva és továbbítva, a hozzáféréseket pedig szabályok kontrollálják [14].  
További fontos jellemzője az egyszerű integráció – Könnyedén kapcsolódik más Firebase-szolgáltatásokhoz, például Firestore-hoz, valós idejű adatbázishoz vagy Authentication-hoz [15].

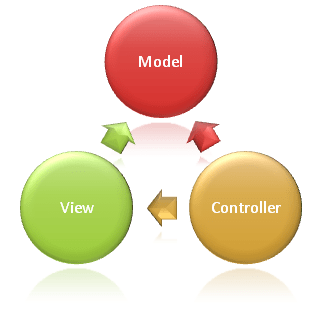
A fejlesztők úgynevezett bucket struktúrán belül dolgoznak, ahol minden fájl metaadatokkal együtt tárolható. A kliensoldali SDK támogatja a megszakított feltöltések folytatását, így a rendszer gyenge hálózati kapcsolatok mellett is megbízhatóan működik [12].

A Cloud Storage for Firebase ideális választás olyan web- és mobilalkalmazások számára, melyek nagy mennyiségű multimédiás vagy felhasználói tartalommal dolgoznak, például közösségi média-, oktatási vagy képszerkesztő rendszer esetében. A szolgáltatás rugalmassága, biztonsága és méretezhetősége révén hatékony háttértárat biztosít modern felhőalapú fejlesztésekhez.

## 2.7. Az alkalmazás architektúrája és rétegződése

A modern webalkalmazások fejlesztésében alapvető elv, a többrétegű architektúra alkalmazása. Ez a megközelítés funkcionálisan elkülöníti az egyes rétegeket, ezzel elősegítve a kód újrafelhasználhatóságát, a karbantarthatóságot és a tesztelhetőséget.

Az általánosan alkalmazott megközelítés az MVC (Model-View-Controller), amely három fő részre tagolja az alkalmazást: Prezentációs (View) réteg: ez a réteg biztosítja a felhasználói interakciót és az adatok megjelenítését. A megvalósítás leggyakrabban HTML, CSS, JavaScript és különböző keretrendszerek (például Bootstrap, React, vagy Angular) használatával történik.   
A következő réteg az üzleti (Controller) réteg, ez dolgozza fel a felhasználói kéréseket, koordinálja az adatok áramlását a View és a Model rétegek között.  
A harmadik réteg az adatkezelési (Model) réteg, ez a réteg az alkalmazás aktuális állapotának kezeléséért, valamint az ehhez kapcsolódó üzleti logikáért és műveletekért felel. Ebbe a rétegbe építjük be az üzleti szabályokat és a perzisztencia logikát (az adatok adatbázisban való tárolásának módját) egyaránt ebben a rétegbe kell beágyazni [16].



Ábra 2.7.1. MVC ábra [16].

A rétegzett architektúra alkalmazása lehetővé teszi a különböző rétegek egymástól független fejlődését [16].

### 2.7.1. Az MVC architektúra működése és a függőségkezelés

Az MVC architektúra célja a felelősségek szétválasztása (Separation of Concerns, SoC). A vezérlő (Controller) nem közvetlenül módosítja az adatbázist vagy a nézetet, hanem közvetítőként funkcionál. Ez a laza csatolás (loose coupling) elve. A modern ASP.NET Core alkalmazásokban ezt a függőségi injektálás (Dependency Injection, DI) elvével valósítják meg. Ez azt jelenti, hogy a magasabb szintű komponensek, mint például a Controller nem közvetlenül hozzák létre az alacsonyabb szintű komponenseket, mint például a Repository osztályt, hanem egy interfészen keresztül kérik be azokat. Ez a gyakorlat jelentősen megkönnyíti a tesztelést, mivel a valós adatbázis-hozzáférés helyettesíthető tesztadatokat szolgáltató hamis objektumokkal (úgynevezett mock objectekkel) [17].

## 2.8. Az adatbázis-struktúra és kapcsolatkezelés

A komplex alkalmazások a doménmodell és az adatbázis-hozzáférés részleteinek szétválasztása érdekében az absztrakciós rétegeket alkalmaznak, melyek célja a kódduplikáció minimalizálása és a karbantarthatóság növelése. A Repository tervezési minta (Design Pattern) kulcsszerepet játszik ebben a szétválasztásban, mivel egységes, gyűjteményszerű interfészt biztosít az adatok eléréséhez és kezeléséhez, elrejtve a mögöttes perzisztencia (adatbázis) logikát. Ez a minta lehetővé teszi a doménréteg számára, hogy deklaratív módon, objektumorientált módon kommunikáljon az adatokkal, anélkül, hogy ismernie kelljen az adatbázis fizikai felépítését vagy a lekérdezések pontos szintaxisát. Az adatmodell tervezése során az entitás-kapcsolat alapú modellalkotás jelentős szerepet játszik, ahol a relációs adatbázis táblái az üzleti domén entitásainak felelnek meg, közöttük egy-a-többhöz típusú vagy több-a-többhöz típusú relációk állnak fenn. Az ORM (például Entity Framework Core) használata lehetővé teszi, hogy a Code First megközelítéssel az entitásosztályból automatikusan létrejöjjön az adatbázisséma, és az adatmanipulációs műveleteket magas szinten, C# osztályokon keresztül történjenek [18].

Az Entity Framework Core használata során az alkalmazásban a Code First megközelítés egyre elterjedtebb. Ez a paradigma azt jelenti, hogy a C# nyelvű osztályok (entitások) jelentik az elődleges forrást a perzisztencia számára, és ezek alapján generálódik le az adatbázis sémája. Ez a módszer jellemzően rugalmasabb, mint a Database First megközelítés, ahol a fejlesztő már egy létező adatbázisból generálja le a kódokat [19]. A Code First szorosan illeszkedik a modern objektumorientált tervezési elvekhez, és jobban támogatja a folyamatos reflektálást és a verziókövetést a migrációs mechanizmusokon keresztül. A fejlesztők a kódban, C# nyelven definiálják a kapcsolatokat és megkötéseket (például [ForeignKey] attribútumokkal vagy a Fluent API-val), melyek az adatbázis szintjén is érvényesülnek [20].

A szoftverfejlesztési életciklus során az alkalmazások adatmodelljei folyamatosan fejlődnek, ami szükségessé teszi a mögöttes adatbázissémák frissítését. Az Entity Framework (EF) Core keretrendszer Migráció (Migration) funkciója kulcsfontosságú eszközt biztosít ezen inkrementális sémafrissítések kezelésére anélkül, hogy a meglévő adatok sérülnének. A folyamat során a fejlesztő az EF Core parancssori eszközeivel generálja le a C# kódot, amely a bekövetkezett modellváltozásokat írja le (migrációs forrásfájlok). Ezeket a fájlok a verziókövető rendszer részét képzik. Az Adatbázis frissítésekor az EF Core egy belső előzménytáblája alapján határozza meg, mely változtatásokat kell még végrehajtani a séma és az adatmodell szinkronizálása érdekében. A migrációs mechanizmus biztosítja, hogy az adatbázisséma változásai verziókövethetőek és bármikor visszavonhatóak legyenek. Ez a funkció különösen fontossá válik nagyobb projektek vagy csapatmunkában végzett fejlesztések során, amikor az adatszerkezetek dinamikusan módosulnak [21].

Az olyan könnyűsúlyú, fájlalapú adatbázis-megoldások, mint az SQLite, fejlesztési és tesztkörnyezetben különösen előnyösek, mivel nem igényelnek dedikált szervert, gyorsan beállíthatók, és platformfüggetlenül működnek [22]. Az alkalmazott ORM-rétegnek köszönhetően a későbbiekben akár más, robusztusabb adatbázis-motor is alkalmazható (például Microsoft SQL vagy PostgreSQL) anélkül, hogy az alkalmazás felsőbb rétegeit jelentősen módosítani kellene [23].

### 2.8.1. Adatvalidáció és adatintegritás

Az alkalmazás megbízhatóságának biztosítása érdekében a bemeneti adatok érvényesítése (validációja) kritikus fontosságú. Az ASP.NET Core MVC keretrendszer beépített mechanizmust kínál a System.ComponentModel.DataAnnotations névtérből származó attribútumok ([Required], [StringLength], [Range]) segítségével. Ez a deklaratív megközelítés lehetővé teszi, hogy a validációs szabályokat közvetlenül a C# entitásosztályok tulajdonságaihoz illesszük. A keretrendszer a validációt szerveroldalon automatikusan érvényesíti a ModelState.IsValid állapot ellenőrzésekor, megakadályozva a hibás adatok adatbázisba való kerülését, ezzel is biztosítva az adatok integritását [24].

## 2.9. Verziókövetés és a fejlesztési metodológia

A szoftverfejlesztés egyik alapköve a verziókövetés, ami lehetővé teszi a fejlesztők számára, hogy a forráskód különböző állapotait nyomon kövessék, és segítse az együttműködést több fejlesztő között, valamint a korábbi verziókhoz való visszatérést. A legszélesebb körben elterjedt eszköz a Git, míg távoli repozitóriumként gyakori a GitHub szolgáltatás alkalmazása [25].

### 2.9.1. A GitHub Flow munkafolyamat

A GitHub Flow egy agilis, elágaz-alapú munkafolyamat, melyet a modern szoftverfejlesztési projektekben a csapatmunka és a gyakori, zökkenőmentes üzembe helyezés támogatására alkalmaznak. A módszertan központi eleme a stabil, mindig telepíthető main (korábban master) ág (branch), melyből minden új funkciót vagy hibajavítás céljából külön fejlesztési ágat (feature branch) hoznak létre. A fejlesztés ezen az elkülönített ágon történik, a változtatásokat pedig gyakori commitekkel rögzítik, biztosítva a verziókövetést. Amikor egy funkció elkészül, a fejlesztő egy Pull Requestet nyit, amely felülvizsgálatra, automatikus tesztelésre és megbeszélésre kínálja a kódot a többi csapattagnak. A sikeres tesztelés és jóváhagyás után a módosítások beolvasztásra kerülnek a main ágba (merge), ahonnan azonnal vagy rövid időn belül automatikusan éles környezetbe telepíthetővé válnak, fenntartva ezzel a Continuous Integration (CI) és a Continuous Delivery (CD) elveit [26].

### 2.9.2. A pull requestek szerepe a minőségbiztosításban

A GitHub Flow munkafolyamatban a Pull Request (PR) nem csupán egy kérés a kód beolvasztására (merge), hanem a minőségbiztosítás (QA) központi eszköze [27]. A PR nyitásakor a csapattagok elvégezik a kódellenőrzést, javaslatokat tesznek a javításra, felismerik a potenciális hibákat vagy a stilisztikai problémákat, még mielőtt a kód bekerülne a stabil main ágba. Emellett a modern Folyamatos Integráció (CI) rendszerek automatikus teszteket (unit és integrációs teszteket) futtatnak minden egyes PR-ra, ezzel biztosítva, hogy az új kód ne törje meg a meglévő funkcionalitást [28]. Ezzel a többlépéses ellenőrzéssel növekszik a végtermékek stabilitása és hatékonysága.

## 2.10. Az alkalmazás futtatása

A szoftverfejlesztési életciklus során a fejlesztési és a tesztelési fázisban az alkalmazás helyi futtatása a Microsoft Visual Studio beépített Kestrel webkiszolgálóján keresztül történik. A Kestrel az ASP.NET Core alapértelmezett, keresztplatformos webkiszolgálója, melyet a fejlesztői környezet indít el a helyi gépen a gyors iteráció és hibakeresés érdekében [29].

A helyi környezet a localhost címen, egy előre meghatározott porton keresztül biztosítja a szolgáltatás elérését. Mivel a szoftver .NET Core alapokon épül, a végleges telepítés (deployment) lehetőségei rendkívül szélesek. Megvalósítható Docker környezetben, Azure App Service-en, vagy bármely más felhőszolgáltatónál (például AWS, Google Cloud), ami támogatja a .NET Runtime-ot, ezzel biztosítva a platformfüggetlen éles üzemet [30].

## 2.11. Összegzés

Összefoglalva, a projekt egy tudatosan megválasztott technológiai stackre épül, amely a Microsoft .NET Core rugalmasságát ötvözi a modern frontend megoldások interaktivitásával. A Code First és az Entity Framework használata gyors adatmodell-fejlesztést tesz lehetővé, míg a lazán csatolt, MVC alapú architektúra a Dependency Injection elvén keresztül biztosítja a könnyű tesztelhetőséget és a kód hosszú távú karbantarthatóságát. A Git/GitHub Flow metodológia garantálja a kódellenőrzést és a magas minőségi szabványokat. A választott technológiák összessége egy robosztus, skálázható és jól karbantartható webalkalmazás létrehozását szolgálja.

# 3. Az alkalmazás működése

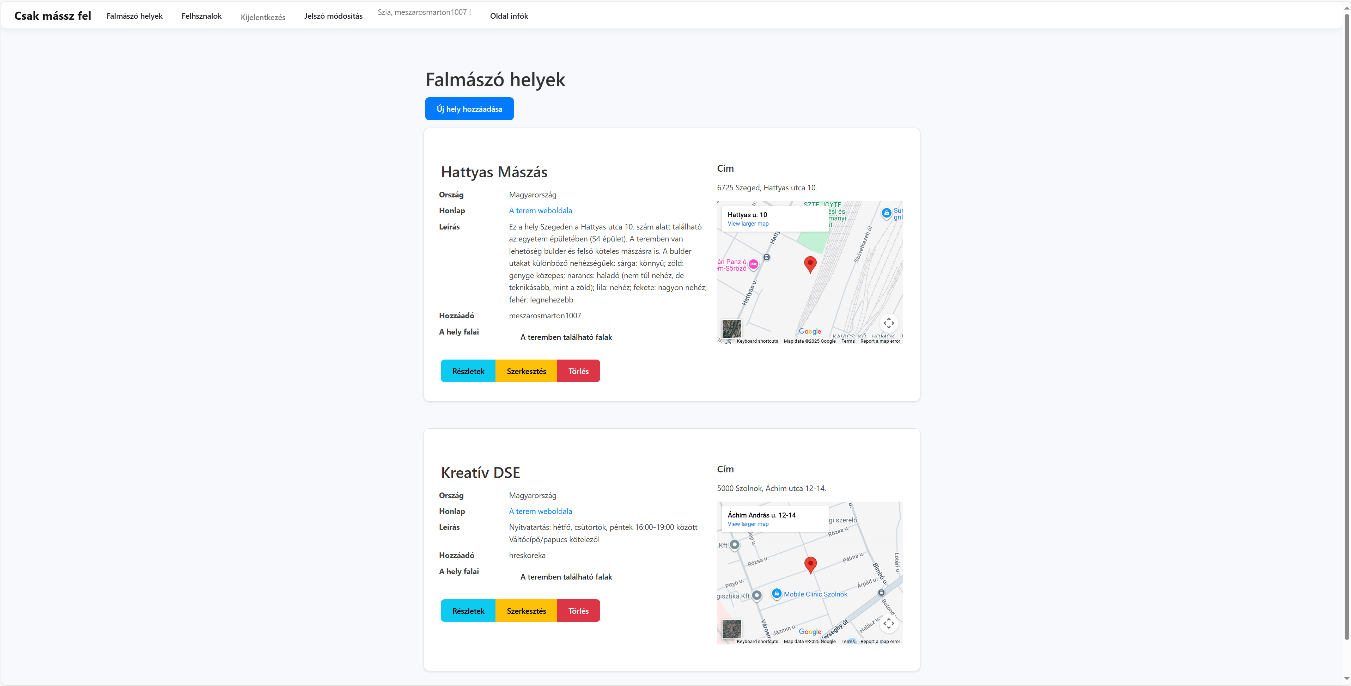
Ez a fejezet a Csak mássz fel névre keresztelt weboldal technikai működését mutatja be funkcionális szempontól, részletesen ismertetve az azalkalmazás architektúráját, az adatok kezelési folyamatát és a felhasználói interakciók technikai megvalósítását. A rendszer alapját az ASP.NET Core MVC keretrendszer adja, amely lehetővé teszi a logikai rétegek – a megjelenítés, az üzleti logika és az adatkezelés – világos szétválasztását. Az adatok perzisztenciája SQLite adatbázisban történik, az Entity Framework Core ORM technológiájának segítségével, amely biztosítja az egyszerű és biztonságos adatkezelést. A felhasználók által feltöltött képek tárolására a webalkalmazás a Firebase Cloud Storage szolgáltatást használja. A fájlokat biztonságosan helyezzük el a felhőben, míg az elérési útvonalak és a metaadatok azadatbázisban tároljuk. A felhasználói felületet a Razor nézetmotor, a Bootstrap és a JavaScript technológiák együttműködése biztosítja, ezzel a weboldal reszponzív, vizuálisan egységes és könnyen kezelhető marad. A dinamikus, interaktív elemek különösen a képszerkesztési funkciók megvalósításához a Fabric.js JavaScript könyvtár nyújt támogatást, amely lehetővé teszi a böngészőben, valós idejű rajzolást a képeken a canvas vásznon keresztül, valamint az undo és redo műveletek implementálását a vászonállapot mentésével.

## 3.1. Felhasználói szerepkörök és jogosultságkezelés

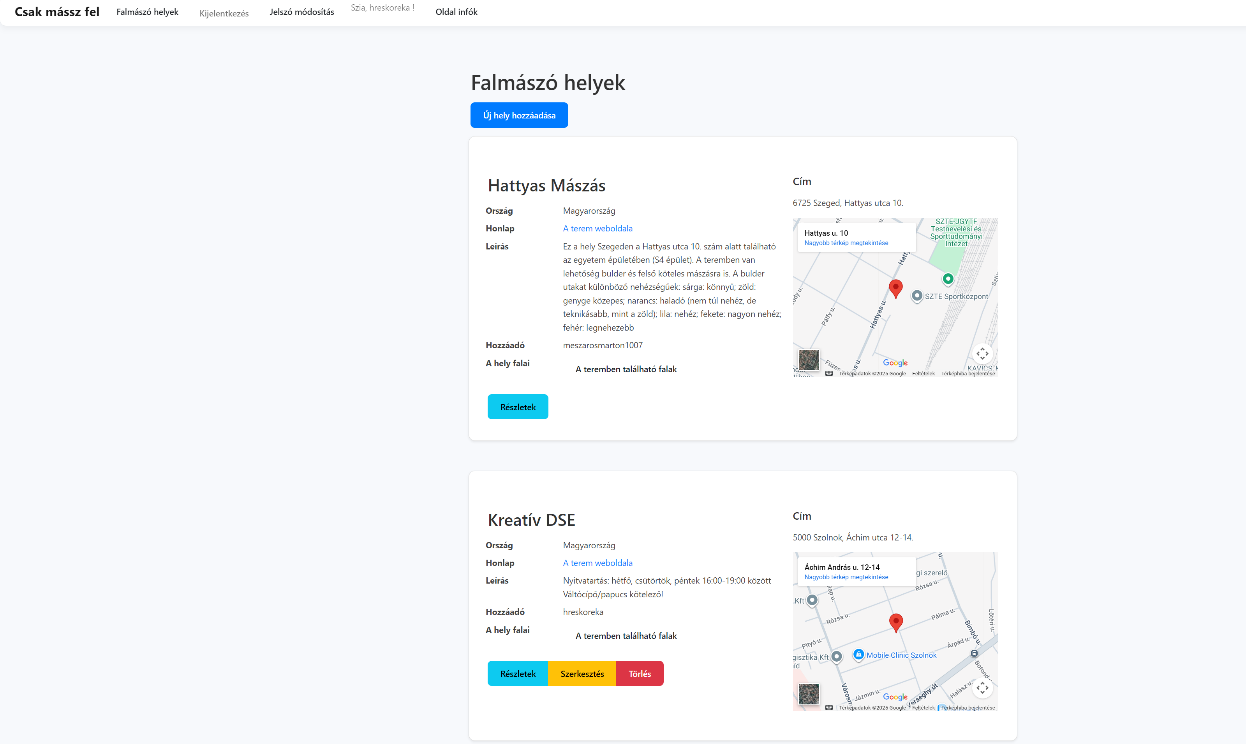
### 3.1.1. Jogosultságkezelés

Az alkalmazás három fő felhasználói szerepkört különböztet meg: az adminisztrátor, a regisztrált felhasználó és a vendég felhasználó. Mindhárom szerepkör különböző jogosultságokkal rendelkezik, így eltérő funkciókhoz fér hozzá az alkalmazáson belül.

Az adminisztrátor a legmagasabb jogosultsági szinttel rendelkezik, ezáltal teljes hozzáférést biztosít az alkalmazás minden eleméhez. Feladatai közé tartozik az új falmászóhelyek, falak és utak hozzáadása, a feltöltött utakhoz való hozzászólás és a hozzászólásokra adott válaszok, valamint mindezek szerkesztése és törlése. Emellett módosíthatja a már regisztrált felhasználók adatait és a felhasználók által feltöltött összes adatot. Így biztosítja a moderálást, megelőzve, hogy nem valós vagy sértő információ túl sokáig jelen legyen.

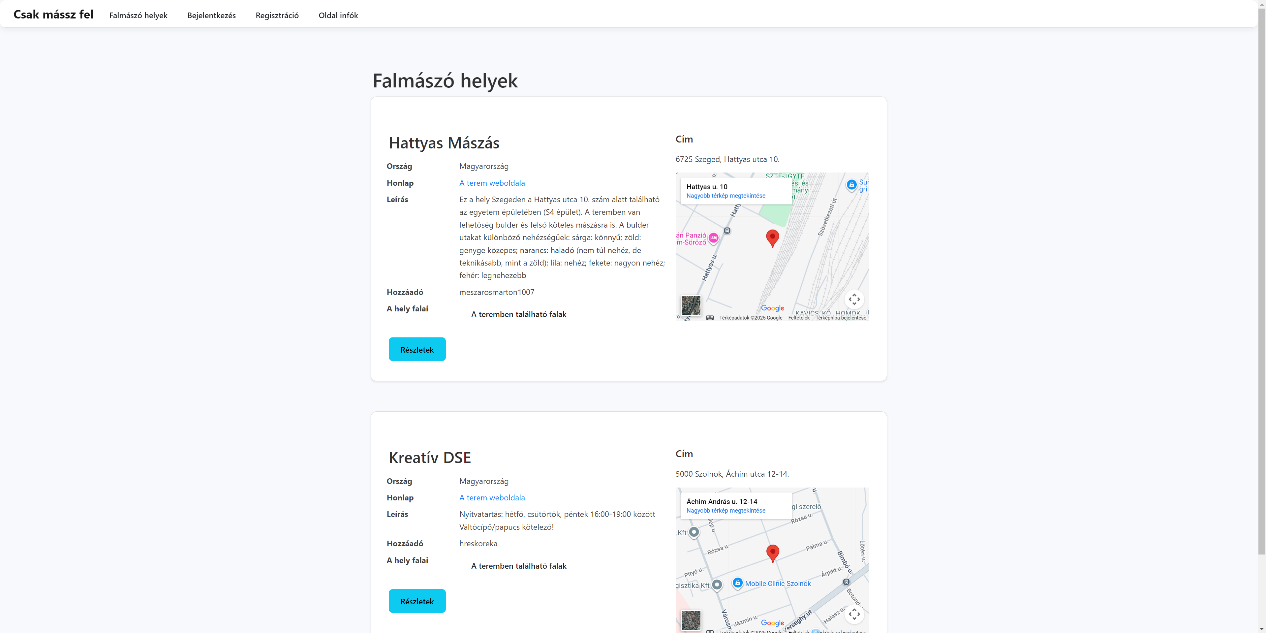


Ábra 3.1 Adminisztrátori nézet. Más által feltöltött adatot is tud szerkeszteni



Ábra 3.2 Regisztrált felhasználó nézet példa. Csak azt tudja módosítani, amit ő töltött fel

A regisztrált felhasználó rendelkezik saját profillal, amelyben saját adatai - felhasználónév, email, név, telefonszám és rang - nem módosítható, ezeket csak az adminisztrátor tudja módosítani. A regisztrált felhasználó a személyes adatai közül kizárólag a jelszavát tudja módosítani. A regisztrált felhasználó ugyanúgy, mint az adminisztrátor tud új falmászóhelyeket, falakat és utakat feltölteni. A feltöltött utakhoz hozzászólni és a hozzászólásokra válaszolni is képes. Az általa feltöltött adatokat tudja módosítani és törölni. Mások által feltöltött adatokat (például út, fal) nem módosíthatja, azokat csak megtekintheti.

A vendég felhasználó csak a nyilvánosan elérhető adatokat láthatja. Böngészhet a már feltöltött falmászóhelyek, falak és utak között, megtekintheti a hozzászólásokat és az ezekre adott válaszokat. Semmilyen adatmódosítást nem végezhet. Ahhoz, hogy bármilyen adatot fel tudjon tölteni, regisztrálnia kell, vagy ha már van felhasználói profilja, akkor be kell jelentkeznie, ezen műveletek elvégzése után egy magasabb jogosultsági körrel fog rendelkezni.

Ábra 3.3 Vendég nézet. Nem tud semmit sem szerkeszteni.

A felhasználók kezelését (bejelentkezés, regisztráció, jelszó módosítás) az AccountController végzi. Az adatok validálását a szerveroldalon hajtjuk végre. A jelszavakat titkosított formátumban tároljuk az adatbázisban, ezzel az adatbázis megfelel az alapvető adatvédelmi követelményeknek. A felhasználói adatok a Felhasznalok modellhez tartozó táblában tárolódnak, amely tartalmazza a személyes adatokat (név, e-mail, telefonszám, születési idő) valamint a felhasználónevet, a rangot és a jelszót.

A jogosultságok alapján való szűrést a nézetoldalakon keresztül valósítjuk meg. Itt a Razor motor segítségével a kérdéses részek megjelenítését egy feltételes utasítás igaz kimeneti ága esetén tesszük lehetővé, ezzel elkerülve, hogy külön adminisztrátori, regisztrált felhasználói és vendég nézetoldalakat kelljen létrehozni.

### 3.1.2. Felhasználó azonosítása

A felhasználók azonosítása a webalkalmazás működésének egy alapvető biztonsági és funkcionalitási eleme. A rendszerben az azonosítást a HTTP-sütik (cookies) segítségével végezzük, amelyek lehetővé teszik, hogy a szerver felismerje a böngészőt, és kapcsolatot teremtsen egy bejelentkezett felhasználó és a hozzá tartozó munkamenet között.

A sikeres bejelentkezést követően a szerver egy cookie-t generál, amelyet visszaküld a kliens böngészőjének. Ennek a sütinek nem az a célja, hogy a jelszót vagy bármilyen érzékeny adatot tároljon, hanem kizárólag egy titkosított munkamenet-azonosítót (token) helyez el, amely a szerveroldali session-hoz kapcsolódik. Ez a token lehetővé teszi, hogy a szerver az azonosító alapján minden további HTTP-kérésnél egyértelműen azonosítsa a kérést indító bejelentkezett felhasználót.

A felhasználó azonosítása minden kérésnél automatikusan megtörténik: a böngésző elküldi a sütit, a szerver összeveti az abban tárolt munkamenet- azonosítót a saját nyilvántartásával. Amennyiben az azonosító érvényes, a rendszer bejelentkezett felhasználóként kezeli azt. Ha a token érvénytelen, lejárt vagy manipulált, a rendszer visszavonja a munkamenetet és kijelentkezteti a felhasználót. Ez a sütialapú hitelesítési mechanizmus biztosítja, hogy a felhasználó folyamatosan egyedi módon azonosítható maradjon, miközben a webalkalmazás megőrzi a magas szintű biztonsági előírásokat és a felhasználói élményt.

## 3.2. Felhasználói műveletek

Ebben a fejezetben az alapvető felhasználókezelést fogom bemutatni: a regisztrációt, a bejelentkezést és a jelszó módosítást.

### 3.2.1. Regisztráció

A webalkalmazás teljes körű használatához felhasználói profil szükséges. A regisztráció fülre kattintva elérhetővé válik az űrlap, ahol meg kell adni a vezeték- és keresztnevet, az e-mail címet, a születési időt, a telefonszámot és egy egyedi felhasználónevet, valamint a jelszót. A rangot a felhasználónak nem kell megadnia, mivel minden regisztrációnál automatikusan az alapértelmezett "user" rang adódik ki.

Az adat-annotáció révén a bemeneti inputokba csak olyan adat kerülhet, amit a rendszer fogadni is tud, null (NULL) adatokat nem lehet továbbítani. Miután minden mezőt megfelelően kitöltöttünk, elindul a regisztráció folyamata. Első lépésként leellenőrizzük, hogy a születési idő nem mutat-e jövőbeli időpontra. Ezután ellenőrizzük az e-mail címet, ezzel elkerülve, hogy egy e-mailhez több felhasználó is tartozzon. Ezt követően ellenőrizzük a felhasználónevet is, amelynek szintén egyedinek kell lennie. A jelszavak ellenőrzését (validációját) az adat-annotációval a böngészőn keresztül végezzük el (kliens-oldali validáció). Az ID, az e-mail cím és a felhasználónév együttesen határozzák meg a felhasználót. Ha minden ellenőrzés sikeres, létrehozzuk a felhasználót a megadott adatokkal és a jelszavát hashelt formátumban továbbítjuk a Felhasznalok modellhez tartozó táblába. A sikeres regisztrációt követően a felhasználót automatikusan bejelentkeztetjük.

### 3.2.2. Bejelentkezés

Amennyiben a felhasználónak van már felhasználói profilja és hozzá szeretne férni a webalkalmazás teljes funkcionalitásához be kell jelentkeznie. Ehhez a bejelentkezés fülre kattintva, a felhasználó a dedikált bejelentkezési felületre jut.

Itt is az adat-annotáció révén nem szükséges az adatok ellenőrzésével külön foglalkozni, mivel a validáció már a LoginViewModel szinten megtörténik, és nem engedélyezi a hibás adat továbbítását. A bejelentkezéshez meg kell adni az e-mail címet és a hozzá tartozó jelszót. Ha a megadott e-mail létezik, és a jelszó megfelelő, a bejelentkezési folyamat sikeresen végbemegy. A jelszó ellenőrzése kétágú logikára épül, ami biztosítja a visszafelé kompatibilitást a régebbi felhasználókkal. Ez a mechanizmus kezeli azokat a felhasználókat, akik a jelszó hashelésének bevezetése előtt kerültek az adatbázisba, és akiknek a jelszava még nem került hashelésre. Felmerül a kérdés, hogy miként döntjük el, hogy az adott jelszó hashelt formátumban tárolódik-e. A beépített hash függvény sózást alkalmaz, mely "AQAAAA"-val kezdődik és az erre a kezdőkaraktersorra szűrünk rá. Amennyiben illeszkedik a jelszó erre a kezdőmintára, a bejelentkezéskor megadott jelszót hash-eljük és a már hash-elt változatával vetjük össze. Egyezés esetén a belépési funkció megvalósul. Amennyiben nem illeszkedik rá, akkor a sima szöveges (plain text) jelszavak egyezőségét ellenőrizzük. Amennyiben egyezik, a jelszót frissítjük az adatbázisban a hashelt változatra és megvalósul a bejelentkezési folyamat.

A bejelentkezési hibakezelés során kiemelt figyelmet fordítunk a biztonságra. A hibák két fázisban jelenhetnek meg a felhasználónál: a ViewModel szintű ellenőrzésén megbukik, például nem ad meg jelszót (adat-annotációs hiba), megkapja az alap „A mező kitöltése kötelező.” hibaüzenetet vagy ha hibás az e-mail akkor a „Hibás email!” üzenetet. Ezt követően a hitelesítési fázisban, ha hibás e-mail-jelszó párost ad meg a felhasználó, akkor a rendszer csak egyszerűen „Hibás email vagy jelszó” üzenetet adja vissza. Ezzel a megoldással megakadályozzuk annak lehetőségét, hogy a támadók információt szerezzenek az adatbázisban tárolt e-mail címekről, növelve az adatbázis biztonságát.

### 3.2.3. Jelszó módosítás

A bejelentkezett felhasználó egyetlen profilmódosítást tud végrehajtani, ez pedig a jelszó módosítása. A művelet végrehajtásához először be kell jelentkezni, majd a jelszó módosítása fülre kattintva megjelenik a jelszómódosítási felület. Ezen a felületen három beviteli mező áll rendelkezésre: a régi jelszó, az új jelszó és az új jelszó megerősítése mező. Itt is az adat-annotáció segítségével először ellenőrizzük, hogy minden mező helyesen van-e kitöltve. Amennyiben minden adat ki van töltve, még a kliens oldalon ellenőrizzük, hogy az új jelszavak megegyeznek-e, ha igen, akkor az adatokat elküldjük a szerver felé, ahol ellenőrizzük, hogy a régi jelszó illeszkedik-e az adatbázisban szereplő, hashelt változattal. Ha az ellenőrzés sikeres, végrehajtjuk a jelszómódosítást. Az új jelszó hashelt változatával frissítjük az adatbázist.

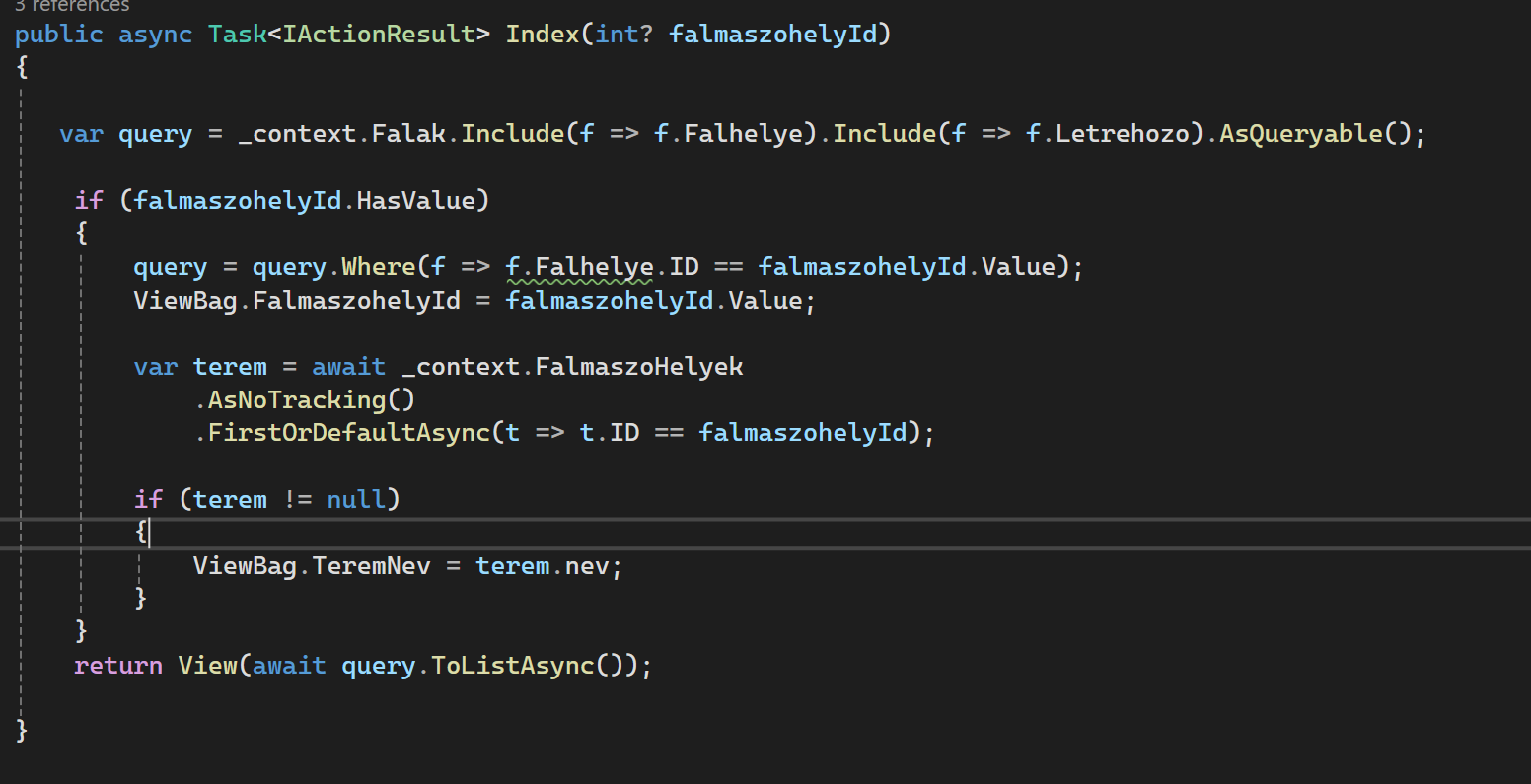
## 3.3. Adatbázis műveletek (CRUD)

### 3.3.1. Az adatbázis felépítése

A webalkalmazás adatperzisztenciáját a SQLite adatbázis biztosítja, amely a projekt igényeihez igazodva egyszerű, gyors és könnyen karbantartható adattárolást tesz lehetővé. Az adatokhoz való hozzáférést és a sémakezelést az Entity Framework Core (EF Core) ORM technológiája valósítja meg az EFContext osztályon keresztül. Ez az osztály felelős az összes releváns entitás, úgymint a FalmaszoHelyek, a Falak, az Utak, a Hozzaszolasok, a Valaszok és a Felhasznalok tábláinak kezeléséért. Az EFContext biztosítja az adatok létrehozásához, módosításához, lekérdezéséhez és törléséhez szükséges alapvető CRUD műveleteket. A projekt logikájában minden adatkezelési művelet a megfelelő kontrollerhez érkezik: a FalmaszoHelyekController felel a falmászó helyekkel kapcsolatos műveletekért, a FalakController a falakért, az UtakController az utakért. A HozzaszolasokController és a ValaszokController kezeli a hozzászólásokat és az azokra adott válaszokat. A felhasználói hitelesítési folyamatokat (bejelentkezés, regisztráció, jelszóváltoztatás, kijelentkezés) az AccountController látja el, míg a FelhasznalokController felel az egyéb felhasználói adatok kezeléséért. Ezek a controllerek nem csupán összegyűjtik a felhasználótól érkező adatokat, hanem elvégzik az érvényesítést, és meghívják az adatbázist módosító aszinkron műveleteket.

### 3.3.2. Lekérdezési műveletek

A webalkalmazás működésének egyik leggyakoribb folyamata az adatok lekérdezése, amelyet a felhasználók az oldal böngészése során folyamatosan igénybe vesznek. A navigáció hierarchikusan épül fel: a falmászó helyek listájáról kiindulva, a „A teremben található falak” gombra kattintva kerül sor az adott helyhez tartozó Falak listázásra. Tovább lépve, a falak oldalon lévő „A fal útjai” gombra kattintva az adott falhoz tartozó Utak kerülnek listázásra. Ez a szelektív betöltés biztosítja, hogy a lekérdezések minden esetben csak a szükséges adatokat töltsék be, ami gyorsabb oldalbetöltési időt és jobb felhasználói élményt eredményez.



Ábra 3.3.2.1. A fal szűrése falmászó hely alapján, C# kód a FalakControllerb-ből

Az utak listázása oldalon jelennek meg a hozzá tartozó hozzászólások és az azokhoz tartozó válaszok is. Az adatok betöltéséről a FalakController-ben lévő Utak(int id) akció során egyetlen adatbázis-lekérdezéssel történik. Az EF Core Eager Loading mechanizmusának (Include és a ThenInclude metódusoknak) köszönhetően az adatok több hierarchikus szinten is betöltődnek, biztosítva, hogy az egyes utakhoz tartozó felhasználói visszajelzések naprakészek legyenek.

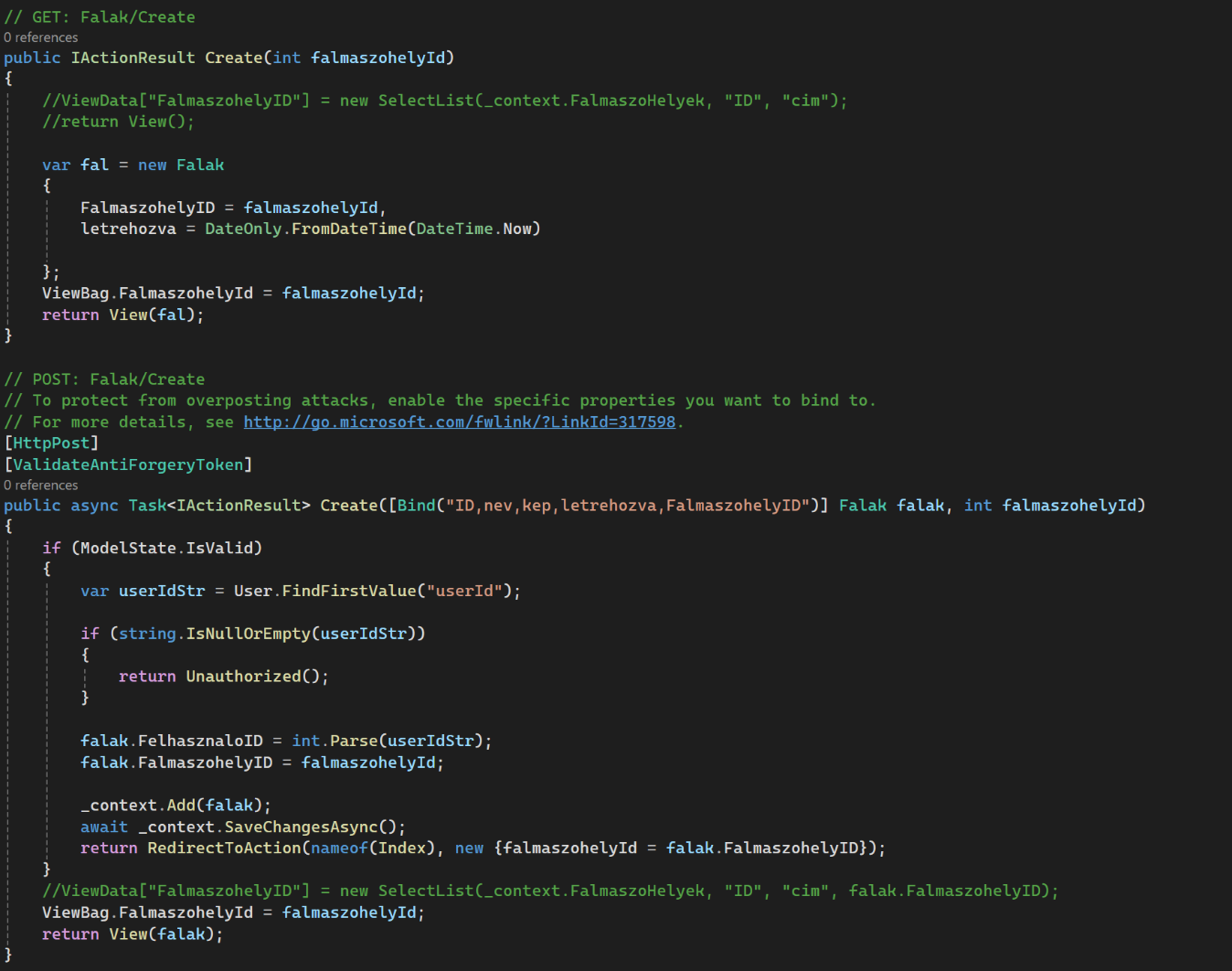
### 3.3.3. Létrehozási műveletek

Az új adatok adatbázisban történő rögzítését a felhasználó által kitöltött űrlapok elküldése indítja el. Mielőtt az alkalmazásban egy új entitást – legyen szó FalmaszoHelyek-ről, Falak-ról, Utak-ról, Hozzaszolasok-ról vagy Valaszok-ról – létrehoznánk, az alkalmazás elvégzi az adat-annotáción alapuló szerveroldali érvényesítést annak biztosítására, hogy a felhasználó az űrlapmezőket helyesen és hiánytalanul töltse ki. Amennyiben a bemeneti adatok megfelelnek a modell elvárásainak, az adott Controller meghívja a Create metódust, majd a sikeres mentést követően a rendszer lekéri a szülő entitást, és ez alapján visszatér a listanézet oldalra.

A létrehozás folyamatában kiemelt hangsúlyt kap a hierarchikus integritás és a jogosultság rögzítése. Például egy út hozzáadása esetén az UtakController először azonosítja a bejelentkezett felhasználót, akinek adatait a Claim-ekből olvassuk ki. Ha a felhasználó nincs hitelesítve, a webalkalmazás automatikusan elutasítja a kérést. Ezzel párhuzamosan ellenőrizzük a logikai kapcsolatot is: a rendszer azonnali elutasítással válaszol (kivéve a FalmaszoHelyek entitást), ha a létrehozni kívánt entitás nem rendelkezik érvényes szülő entitással, például ha egy fal nem kapcsolódik egy falmászó helyhez, vagy egy válasz nem kapcsolódik egy létező hozzászóláshoz.

Fontos eltérés a hozzászólások és a válaszok esetén az, hogy azokat nem a hozzájuk tartozó nézetoldalon rögzítjük, hanem egy dedikált myCreate metódus hívásával, amelyet a hozzászólások esetén a HozzaszolasokController-ben, míg a válaszok esetén a ValaszokController-ben definiáljuk. A metódus automatikusan hozzárendeli a felhasználót (Claim-ek alapján) és a megfelelő szülő entitást (utat vagy hozzászólást). A sikeres mentést követően a rendszer lekérdezi a szülő entitás azonosítóját, és erre hivatkozva visszatér az út listanézet oldalra. Ez a mentési logika csak abban tér el a többitől, hogy egy másik nézetoldalon belül valósítjuk meg a hozzáadást.

A létrehozás lépései az utaknál a következők: a bejelentkezett felhasználó az oldal tetején a címsor alatt található „Új út létrehozása” gomra kattintva átirányításra kerül egy új oldalra, ahol az új utat hozzá tudja adni, itt minden szükséges adat kitöltését követően a „Mentés” gombra kattintva elmenti az adatokat. Amennyiben vissza akarunk lépni az előző oldalra, az adatok mentése nélkül a „Vissza az utakhoz” gombra kattintva visszatérünk a listanézetre. A szigorú szülő- gyermek kapcsolat biztosítása után a Controller az entitást az EFContext segítségével hozzáadjuk az adatbázishoz, és a SaveChangesAsync aszinkron hívásával az új rekord véglegesen rögzítjük.



Ábra 3.3.3.1 FalakController Create metódusa új fal hozzáadása

### 3.3.4. Módosítási műveletek

A felhasználók által bevitt adatok módosítása a szerkesztőfelületeken történik. Egy falmászó hely, fal, út, hozzászólás, válasz szerkesztését az adott Controller kezeli, amely először betölti a módosítani kívánt rekordot, majd megjeleníti a jelenlegi adatait. A felhasználó ezeket átírhatja, majd a módosításokat visszaküldi a szervernek.

Az adatmódosítás az [HttpPost] Edit metódus hívásával történik. Itt az adatok érvényesítés után, az integritás megőrzése érdekében komplex biztonsági lépéseket hajtunk végre. Mivel az űrlapok csak a módosítható mezőket tartalmazzák, a kritikus külső kulcsok – a szülő entitás például FalmaszohelyID és a FelhasznaloID (létrehozó felhasználó) – manipulációjának megakadályozása érdekében a Controller lekéri az eredeti adatokat az adatbázisból AsNoTracking módban. Ebből az eredeti rekordból visszatöltjük a logikai kulcsok értékét a frissített entitásba, ezzel megelőzve, hogy a felhasználó az űrlapon keresztül módosíthassa a hierarchikus kapcsolatokat és a létrehozó személyt.

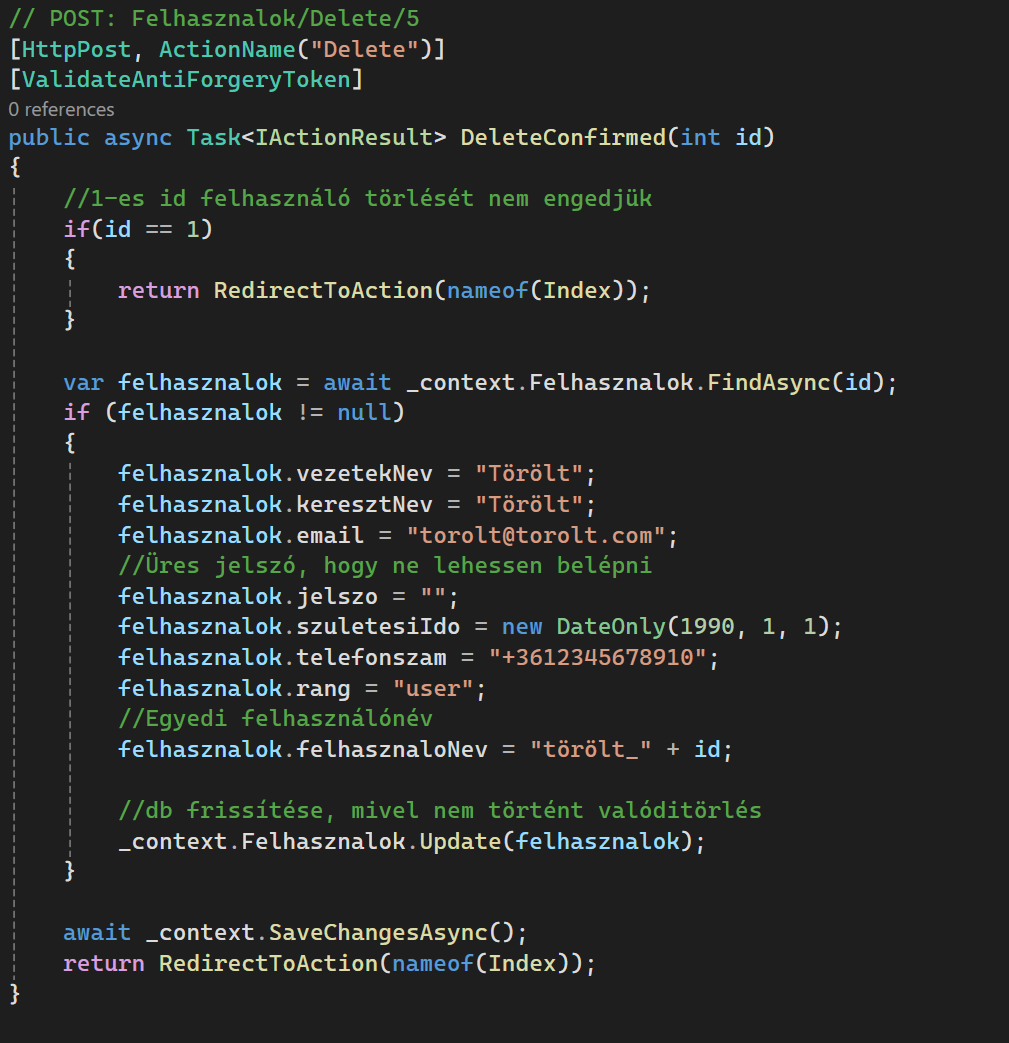
A módosított, és helyes logikai kulcsokkal rendelkező entitást az EFContext az Update metódussal nyomon követi. A változások véglegesítését a SaveChangesAsync metódus végzi. Ezt a szakaszt egy try-catch blokkal védjük, amelynek az a célja, hogy kezelje a *DbUpdateConcurrencyException*- t, biztosítva, hogy több felhasználó egyidejű módosítási kísérlete esetén a rendszer megfelelően reagáljon, és fenntartsa az adatok konzisztenciáját. A sikeres módosítást követően a felhasználót átirányítjuk a frissített listanézetre.

A frissítés menete: a felhasználó kiválasztja azt az entitást, amit módosítani szeretne, és rákattint a szerkesztés gombra, ez átviszi a szerkesztési oldalra, ahol a létrehozáshoz hasonló űrlappal találkozik. Ebben az űrlapban az adatok már előre ki vannak töltve, és a módosítani kívánt adatot csak át kell írni, ezután a ”Mentés” gombra kell kattintani, és a rendszer elvégzi a mentést. Hozzászólások és válaszok esetén eltérő, mert az utaknál rögtön szerkeszthetjük a hozzászólásokat és a válaszokat, anélkül, hogy egy külön oldalra lépnénk át.

### 3.3.5. Törlési műveletek

A törlési műveletek az adatkezelés legkritikusabb pontja, mivel a DeleteConfirmed metódusban nem csupán az adatbázis tartalmát, hanem a falak és az utak esetében a Firebase Cloud Storage-ban tárolt képeket is el kell távolítanunk. Erről a Firebase műveletről későbbi fejezetben fogok írni.

A törlési folyamatban először a képet távolítjuk el. Ez azért van, hogy abban az esetben, ha a képet nem sikerül törölni a Firebase-ből, akkor a rekord se törlődjön, ezzel megakadályozzuk, hogy egy nem használt kép felesleges helyet foglaljon el a Firebase Cloud Storage-ban. Miután a fal képét töröltük, az EFContext a Remove metódus hívással eltávolítja a rekordot az adatbázisból, melyet a SaveChangesAsync aszinkron metódus hívása véglegesít. Az EF Core kaszkád törlési szabályai eközben automatikusan biztosítják, hogy az egymáshoz tartozó gyermek entitásokat is eltávolításra kerüljenek, megelőzve az árva rekordok keletkezését és fenntartva az adatbázis integritását.



Ábra 3.3.5.1. Felhasználó törlése és az 1-es ID-val rendelkező felhasználó törlésének megtiltása

A felhasználók törlése másképpen zajlik, itt ugyanis nem konkrét törlést végzünk, hanem frissítést, a felhasználó minden személyes adatát átírjuk egy egységes mintára. A felhasználóneve a ” törölt” szó plusz az ID-ja lesz és ez fogja azonosítani az általa létrehozott entitásokat. Erre azért van szükség, hogyha a felhasználó sok adatot töltött fel, akkor az esetleges törlése esetén az adatbázis konzisztenciája megmaradjon. Egy másik biztonsági lépés, hogy az 1-es ID-val rendelkező admint nem lehet törölni, ezzel megakadályozzuk, hogy adminisztrátor nélkül maradjon az oldal.

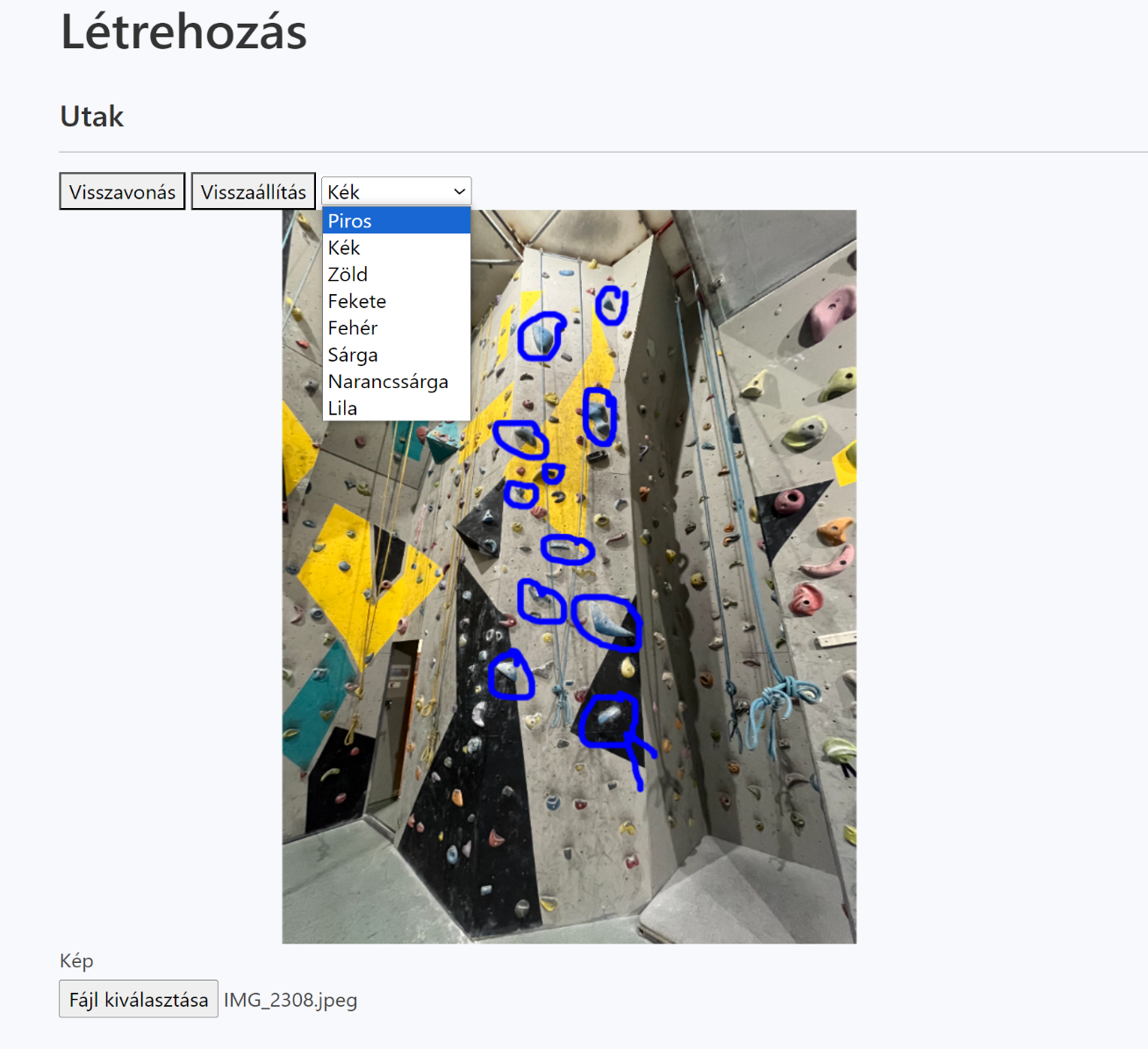
## 3.4. Képkezelési és interaktív szerkesztési funkciók

A webalkalmazás képkezelési logikája a moduláris FeltoltesController és a kliens oldali Fabric.js könyvtár szoros integrációjára épül, amely teljes körű CRUD műveleteket biztosít mind a Falak, mind az Utak entitásokhoz tartozó vizuális adatok számára. Az egész folyamat a Base64-kódolású képadatok aszinkron kezelésén alapul. A felhasználók által szolgáltatott képadatok stringként érkeznek a szerverre, ahol a FeltoltesController felelős az adatok dekódolásáért, egyedi azonosítók (GUID) alapján fájlnév generálásáért, majd a Google Cloud Storage Client segítségével történő hitelesített feltöltésért a Firebase Cloud Storage dedikált tárhelyére. A sikeres tárolás eredményeként a metódus a kép nyilvános URL-jét küldi vissza a kliensnek. Ezt az adatbázisban külső hivatkozásként rögzítjük, biztosítva ezzel egy skálázható és biztonságos megoldást a nagyméretű médiafájlok kezelésére.

### 3.4.1 Interaktív szerkesztés és állapotkezelés

Az alkalmazásban az Utak vizuális megtervezésére a Fabric.js Canvas interaktív képességeire támaszkodunk. Ezt az Utak/Create.cshtml és az Utak/Edit.cshtml nézetek implementálják. Míg a falak esetében a kép csak statikus háttérként szolgál, itt a funkcionalitás kiterjed a valós idejű szerkesztésre. Amikor egy felhasználó új képet tölt fel, vagy egy meglévő kép betöltődik, a Canvas aszinkron módon betölti a képet, és automatikusan beállítja azt háttérképként. Ezt megelőzően a képet arányosan skálázzuk, hogy optimálisan illeszkedjen a Canvas előre meghatározott méreteihez (width=”800” height=”600”), megakadályozva a torzulást és optimalizálva a rajzolási felületet. A Canvas ezután szabadkézi rajzolási módba kerül, lehetővé téve a felhasználók számára, hogy szabad kézzel felvigye a mászóutak vonalát vagy az útban használható fogásokat megjelölje.

A felhasználói műveletek nyomon követésére és a hibák kijavítására egy állapotkezelő mechanizmust valósítunk meg. Minden sikeresen befejezett rajzolási művelet után a vászon teljes állapota (a háttérképpel és az összes felvitt grafikus elemmel együtt) JSON formátumban mentésre kerül egy speciális JavaScript tömbbe, a *stateStack*-be. Ez a verem adatszerkezet biztosítja, hogy a felhasználó a Visszavonás (Undo) funkcióval bármikor vissza tud lépni egy korábbi állapotra. A Visszaállítás (Redo) funkció támogatására egy kiegészítő *redoStack* verem is használatban van. Az űrlap elküldésekor a JavaScript ellenőrzi, történt- e valódi módosítás a Canvas tartalmában. Ha az állapotváltozás detektálható, az aszinkron saveImage metódus végrehajtja a kép feltöltését a Cloud Storage-be. Ez a folyamat a Fabric.js canvas.todataURL(’img/png’) metódusát használja a vászon aktuális tartalmának egyetlen Base64 kódolású képfájlként történő kinyerésére, melyet aztán a szerver oldalra továbbítunk. A sikeres mentés után kapott új URL-t a kliensoldali logika beállítja a modellhez tartozó rejtett mező értékének, ezzel biztosítva, hogy a szerveroldali Create vagy Edit Action metódus már a frissített képhivatkozással dolgozzon.



Ábra 3.4.1.1. Képek szerkesztése az utak vizuális reprezentációjához

### 3.4.2. Mászóutak képkezelése: szerkesztés és adatintegritás

Az Utak entitás képeinek kezelése a Falak esetében alkalmazott, kétlépcsős aszinkron képkezelési logikát követi, kiterjesztve azt az útvonalrajzokat is tartalmazó Canvas-állapotra. A képfrissítés egy kritikus, aszinkron tranzakcióra épül, amelynek célja a tárhely optimalizálása és az adatintegritás fenntartása.

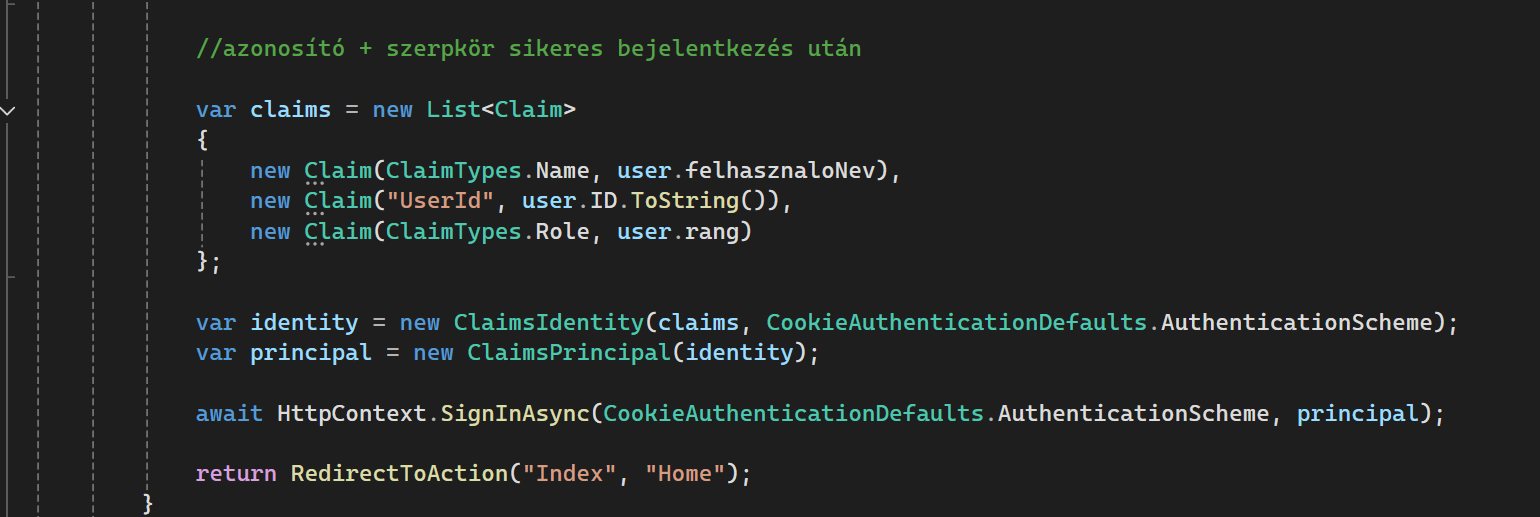
Amikor a felhasználó frissíti a falnak a képét, vagy az utat bemutató képet, a deleteOldImage aszinkron metódus első lépésben meghívja a szerveroldali api/Feltoltes/deleteimage végpontot. Ezzel a művelettel proaktívan eltávolítja a Firebase Storage-ból a lecserélt, korábbi képfájlt, amely tartalmazza a régi útvonalrajzot. A kód robosztussága érdekében a hibakezelés itt kiemelt szerepet kap: amennyiben a régi kép törlésekor hiba lép fel (például a kép már nem létezik a Storage-ben), a hiba naplózásra kerül, de a mentési folyamat nem szakad meg. Ezt követően a saveImage aszinkron metódus végrehajtja a Canvas aktuális rétegzett tartalmának feltöltését. A Canvas-ról kinyert új kép a Cloud Storage-ba kerül. A sikeres mentés eredményeként a kapott új URL beíródik a megfelelő rejtett mezőbe, garantálva, hogy a szerveroldali Edit Action metódusa már a frissített képhivatkozással frissítse az adatbázisrekordot. A Falak entitás képeinek frissítése csak abban különbözik, hogy ott nincs képszerkesztés, csak a régi kép törlése és az új kép feltöltése.

### 3.4.3. Adatintegritás és árva fájlok megelőzése (Törlés)

Az útvonal vagy a fal végleges törléséért felelős nézetek esetében az adatintegritás és a Google Cloud Storage tárhelyének tisztán tartása a legfontosabb szempont, elkerülve az árva fájlok kialakulását. A szerveroldali DeleteConfirmed Action metódus logikája ezért szigorúan előírja, hogy az adott úthoz vagy falhoz tartozó képfájl eltávolítása előzze meg az adatbázisrekord törlését. A logika az Utak vagy Falak entitás adatbázisából való eltávolítás előtt felhasználja a törlendő kép URL-jét, és a FeltoltesController törlési logikáját alkalmazva eltávolítjuk a képet a Google Cloud Storage-ból. Ez a sorrend garantálja, hogy felesleges, már hivatkozás nélküli képfájlok – amelyek az utakat és a falakat vizuálisan reprezentálják – ne halmozódjanak fel a tárolóban, biztosítva ezzel a rendszer erőforrás-hatékonyságát és konzisztenciáját.

## 3.5. Biztonsági mechanizmusok

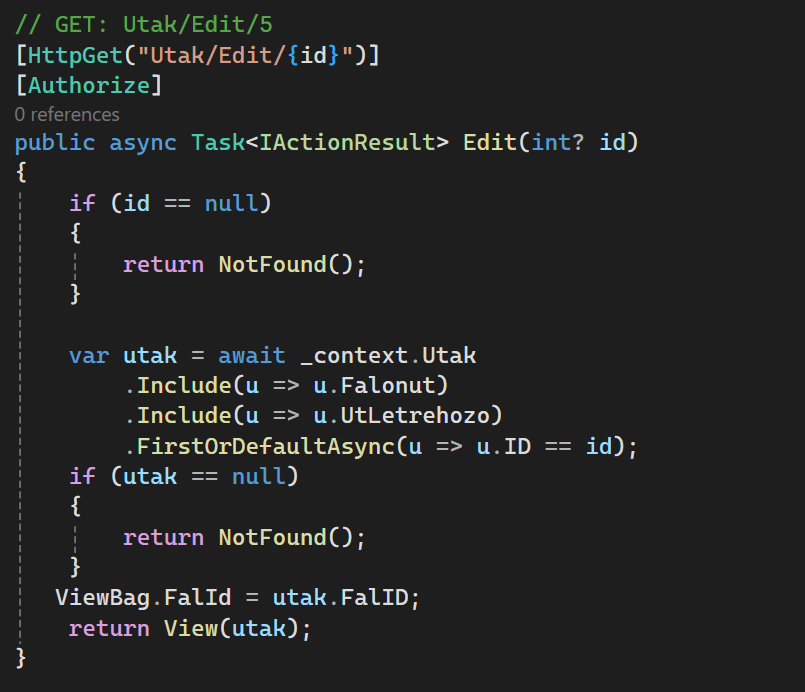
A webalkalmazás biztossági rétege a hitelesítési és a jogosultságkezelési folyamatokra épül, amelyek együtt biztosítják, hogy a felhasználók csak azokhoz férjenek hozzá, amelyhez megfelelő engedéllyel rendelkeznek. A bejelentkezési folyamatot az AccountController valósítja meg, ahol, a felhasználó által megadott adatok szerveroldali hitelesítése után létrejön az a titkosított munkameneti cookie, amely a teljes felhasználói munkamenetet állapját adja. A bejelentkezést követően a rendszer egy olyan cookie-t hoz létre, amely egy titkosított tokennel azonosítja a felhasználót, és a böngésző minden további kérése során automatikusan visszaküld. A szakdolgozat 3.1.2. fejezetében már megjelenő logikát a belépési metódus konkrét implementációja valósítja meg: a bejelentkezés után a SingInManager a saját kódbázisban szereplő AddClaimsIdentity eljárással hozza létre azt az Identity.Cookie munkamenetet, amely alapján a szerver minden kérésnél felismeri a felhasználót.



Ábra 3.5.1. Bejelentkezési süti létrehozása a felhasználók azonosítására

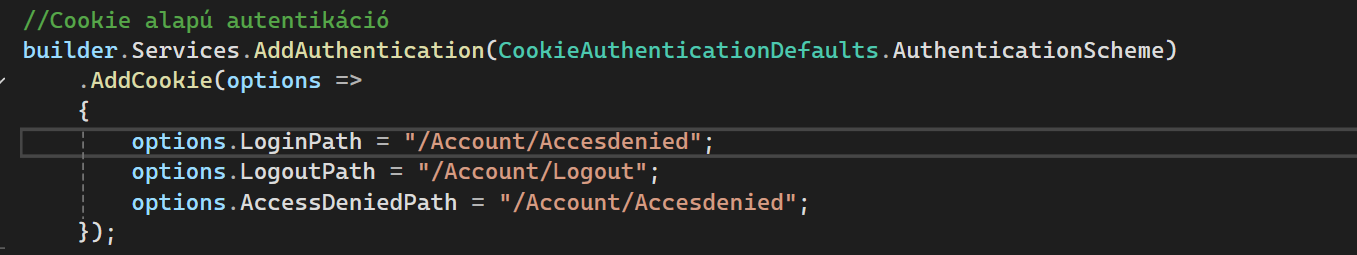
A jogosultágkezelés ehhez a hitelesítési réteghez társul. A rendszer a vezérlők és műveletek védelmét az ASP.NET Core beépített [Authorize] attribútumával valósítjuk meg. A forráskódban ez a jelölés több helyen is megjelenik, pédául az UtakController módosító műveleteiben, ahol a rendszer nem engedi végrehajtani az adatváltoztatást olyan felhasználónak, aki nincs bejelentkezve. A jogosultsági szabályok nem csak azt ellenőrzik, hogy a felhasználó, be van-e jelentkezve, hanem azt is, hogy a saját rendszerben szereplő rangja – amelyet a Felhasznalok modell tart nyilván – elegendő-e az adott művelet végrehajtásához. A szakdolgozat 3.1. fejezetében már bemutatott rangok kezelése itt válik ténylegesen működő biztonsági tényezővé: a rendszer például nem enged módosítani olyan adatot, amelyet nem az adott felhasználó hozott létre, vagy nem rendelkezik adminisztrátori szinttel.

Ábra 3.5.2. Jogosultság korlátozása az UatkController Edit Get metódusára



A jogosultsági hibák kezelésére a rendszer az AccessDeniedPath útvonalat alkalmazza, amelyet a Program.cs tartalmaz. Amikor a felhasználó egy olyan erőforrást próbál elérni, amelyhez nincs megfelelő jogosultsága, a rendszer automatikusan átirányítja erre az oldalra. Ez az oldal az AccesDenied.cshtm. Ezzel a megoldással biztosítjuk, hogy a jogosultsági hiba ne technikai kivételként jelenjen meg, hanem felhasználóbarát módon jelezze a hozzáférés megtagadását. Az implementáció lényegi eleme, hogy a rendszer a hitelesítést és a jogosultságkezelést teljesen külön kezeli: először azonosítja a felhasználót a cookie alapján, majd külön folyamtban vizsgálja, hogy az adott felhasználó jogosult-e a kért művelet végrehajtására.

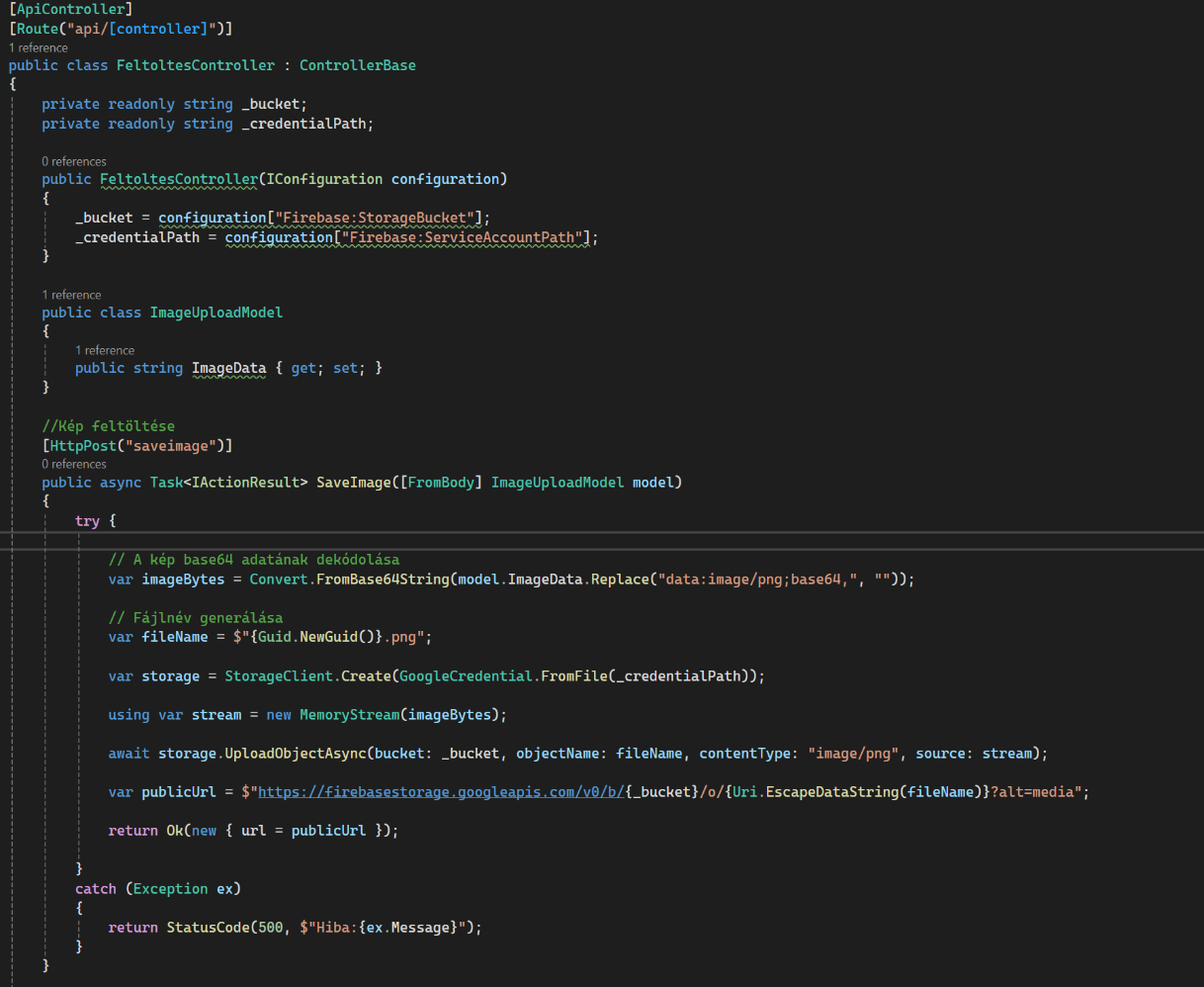
## 3.6. Külső szolgáltatások integrációja



Ábra 3.5.3. Jogosultság ellenőrzés és átirányítás

A képfájlok kezelése az alkalmazás egyik legkritikusabb funkciója, mivel a falak és az utak vizuális megjelenítése nagyméretű állományokat érint. Ezek tárolása nem a webalkalmazás saját tárhelyén történik, hanem a Firebase Cloud Storage szolgáltatásán keresztül, amelyhez a rendszer a Google Service Account hitelesítési mechanizmusát használja. A tároló eléréséhez szükséges hitelesítési adatok a projekt gyökérkönyvtárában található firabase-adminsdk.json fájlban szerepelnek. Ennek a fájlnak az elérési útját és a Firebase Bucket nevét a rendszer a konfigurációs állományból (IConfiguration) olvassa be a FeltoltesControllerv konstruktorába. Ezeket az adatokat felhasználva hozom létre azt a StogageClient példányt, amelyet az összes további fájlművelet alapjaként használunk.

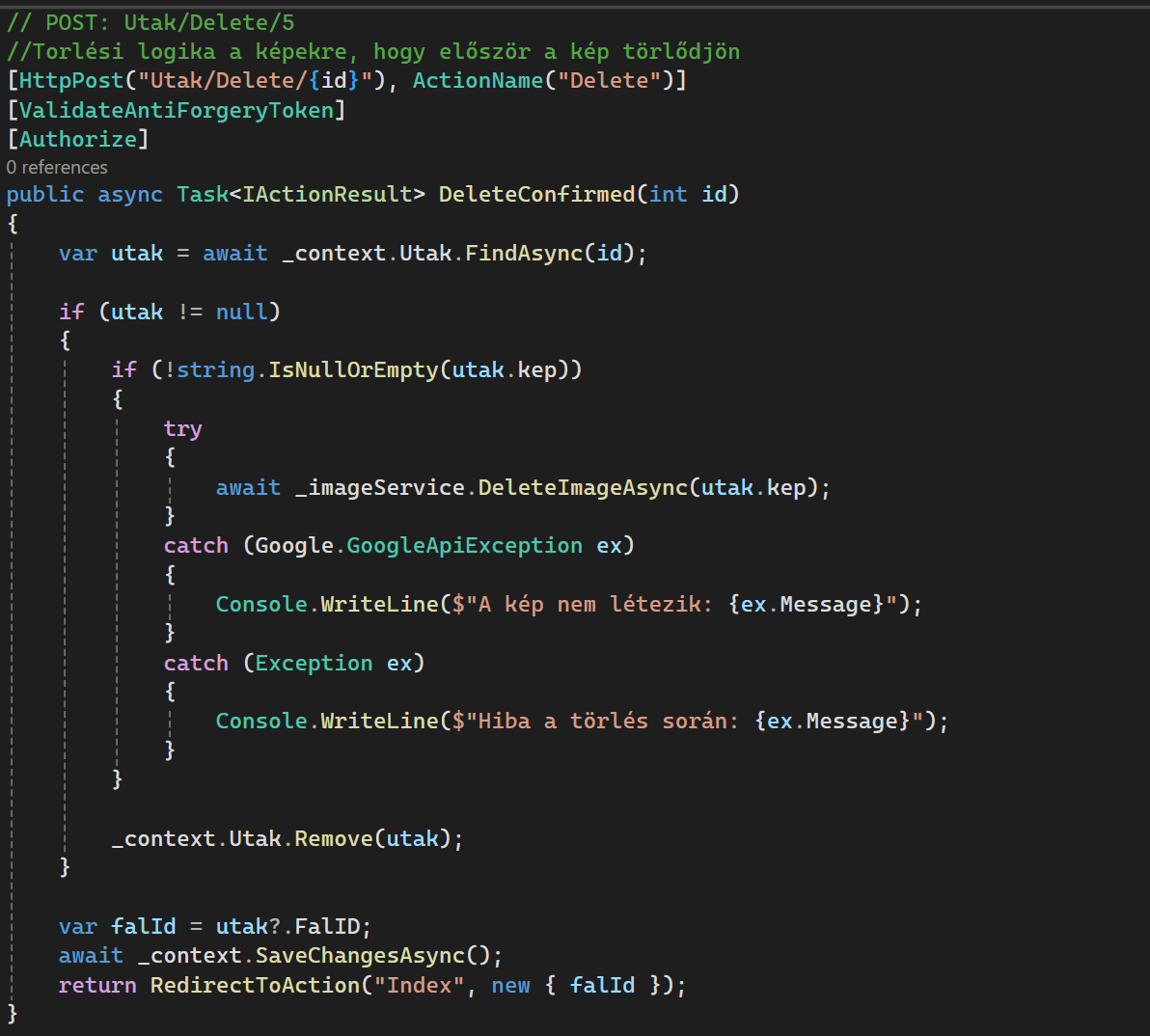
A képfeltöltési folyamat a FeltotesController API-vezérlőn keresztül zajlik, ahol a felhasználó által kiválasztott kép Base 64 kódolt streamként érkezik be. A Controller elvégzi a dekódolást, majd a létrejött adatfolyamot továbbítja a StorageClient UploadObjectAsync metódusába. A feltöltési folyamat szorosan illeszkedik a képkicserélési logikához: a rendszer először feltölti az új képet a Firebase tárhelyére, ezt követően pedig a szolgáltatás által generált publikus URL-t elmenti a megfelelő entitás (út vagy fal) adatbázismezőjébe. Ezzel biztosítjuk, hogy mindig a legfrissebb képet jelenítse meg.



Ábra 3.6.1. FeltotesContoller Kép feltöltése

A törlési folyamat ennél összetettebb, mivel a Firebase letöltési hivatkozásai jellemzően paraméterezett, escapelt URL-ek, amelyek nem tartalmazzák közvetlenül a törlendő fájl nevét. Ezt a problémát a saját implementációmban szereplő ExtracktFileNameFromUrl metódussal oldjuk meg. A metódus a hivatkozás azon részét vágja ki és dekódolja, amely a tárolóban szereplő tényleges állománynévnek felel meg. A fájlnév kinyerése után a törlési kérelem a StorageClient DeleteObjectAsync metódusával hajtódik végre.

A rendszer minden frissítésénél és törlésénél a DeleteImage API-végpontot hívja meg. A törlési logikát úgy alakítottam ki, hogy az entitás törlése előtt az eredeti URL alapján automatikusan eltávolítjuk az állományt a tárhelyről. Ez a logika megakadályozza, hogy felesleges árva fájlok maradjanak a Firebase-ben és hosszú távon optimalizálja a tárhelyhasználatot.



Ábra 3.6.2. Utak kép törlése először majd a teljes adatbázis rekord törlése

A Firebase Storage integráció ezzel olyan, a szerver saját erőforrásait tehermentesítő, skálázható megoldássá válik, amely teljes mértékben illeszkedik a rendszer működéséhez. A fájlműveletek biztonságos, hitelesített kapcsolaton keresztül zajlanak, a külső tároló használata pedig lehetővé teszi, hogy a rendszer nagy felbontású képekkel is hatékonyan működjön.

# 4. Összegzés

A szakdolgozatomban megfogalmazott célkitűzést –egy olyan webalkalmazás létrehozása, amely rendszerezetten támogatja a falmászó közösséget – sikeresen megvalósítottam. A projekt során kialakítottam egy modern, ASP.NET Core MVC alapú architektúrát, melyben az Entity Framework Core ORM-et, a SQLite adatbázist és a Firebase Cloud Storage szolgáltatást használtam az adatok tárolására és biztonságos kezeléséhez. A webalkalmazás fejlesztésekor kiemelt figyelmet fordítottam a felhasználói élmény optimalizálására, a személyes profilok kezelésére, egy többszintű jogosultsági rendszer kialakítására, valamint a Fabric.js könyvtár beépítésével a dinamikus, képalapú interakciók biztosítására.

A rendszerben kezelem a három különböző felhasználói szerepkört: a Vendég, a Regisztrált felhasználó és az Adminisztrátor számára eltérő hozzáférést biztosítok. Ezzel a szerepkörmodellel biztosítom, hogy a felhasználók kizárólag a számukra releváns funkciókat érhessék el, miközben fennmarad a rendszer biztonsága és adatainak integritása. A regisztrációs, bejelentkezési és jelszó módosítási funkciókhoz a Microsoft PasswordHasher biztonságos hashelési megoldásait rendeltem hozzá, ezzel biztosítottam a felhasználói adatok védelmét. A sütialapú hitelesítési folyamat révén tettem lehetővé, hogy a rendszer folyamatosan azonosítsa a bejelentkezett felhasználót anélkül, hogy érzékeny adatokat tárolnék közvetlenül a kliensoldalon.

Az alkalmazás egyik legfőbb funkciójaként megvalósítottam a képek kezelését és metódusait. A felhasználók által feltöltött képeket a Firebase Cloud Storage-ban tárolom, amely nagy rendelkezésre állást, skálázható és biztonságos objektumtárolási megoldást kínál. A képszerkesztési funkciót a Fabric.js alapokra építettem, lehetővé téve a felhasználók számára a szabadkézi rajzolást, továbbá a rajzolási műveletek Visszavonását (Undo) és Újraalkalmazását (Redo). Mindezen műveleteket teljes egészében a kliensoldalon valósítottam meg, így a felhasználói felület gyors és reszponzív. A képek cseréje, törlése vagy frissítése során az alkalmazás automatikusan kezeli a fájlok Firebase-ből való törlését és az új fájl feltöltését, miközben az adatbázisban frissülnek a megfelelő hivatkozások.

A projekt fejlesztése során a skálázhatóságot, a modularitást és a tiszta architektúra elveinek betartását helyeztem előtérbe. A rendszert a Code First megközelítéssel készítettem el, így a modellekből generált adatbázis-struktúra könnyen bővíthető és karbantartható. A GitHub Flow alapú verziókezelési stratégiának köszönhetően a fejlesztési folyamatot jól követhettem és visszakereshettem, a Pull Requestek pedig folyamatos minőségellenőrzést biztosítottak a kód számára.

A Csak mássz fel alkalmazással a célkitűzésnek megfelelő, működőképes, könnyen bővíthető és stabil alapot biztosítottam a jövőbeli fejlesztésekhez. Továbbfejlesztési lehetőségként szükségesnek látom beépíteni a mászóteljesítmények jelölésének funkcióját, amely lehetővé teszi egyfajta ranglista kialakítását arról, hogy az adott helyen ki mennyi utat mászott meg. Fontos a közösségi interakciók minőségbiztosítása is: implementálnék egy automatikus szűrő funkciót a kommentekhez, amely megakadályozza a sértő és nem megfelelő szavak használatát a hozzászólások és válaszok posztolásakor. Kiemelt feladatként megvalósítanék a webalkalmazás publikus elérését (hosting), valamint az ehhez kapcsolódó automatizált Continuous Deployment (CD) folyamatok kialakítását, amely biztosítja, hogy a kódtár (repo) frissülésekor az alkalmazás automatikusan kitelepüljön. Ezen túlmenően a térképi funkciók mélyebb Google Maps API integrációval történő bővítését tervezem. A jelenlegi robusztus alap lehetővé teszi, hogy a jövőben az alkalmazást egy, a mászóközösség számára még részletesebb, interaktívabb és funkciógazdagabb rendszerré fejlesszem tovább.

Forráskód: https://github.com/meszarosmarton1007/szakdolgozatFalmaszas/

# Irodalomjegyzék

|  |  |
| --- | --- |
| [1] | theCrag, „theCrag,” 2025. [Online]. Available: https://www.thecrag.com/. [Hozzáférés dátuma: 16 10 2025]. |
| [2] | 2. Crags, „27 Crags,” [Online]. Available: https://27crags.com/. [Hozzáférés dátuma: 20 10 2025]. |
| [3] | Természetjáró, „Sziklamászó helyek,” [Online]. Available: https://www.termeszetjaro.hu/hu/list/sziklamaszo-helyek/201908310/. [Hozzáférés dátuma: 20 10 2025]. |
| [4] | Microsoft, „What is Visual Studio?,” Microsoft Learn, 09 09 2025. [Online]. Available: What is Visual Studio?. [Hozzáférés dátuma: 11 11 2025]. |
| [5] | „Geeksforgeeks,” [Online]. Available: https://www.geeksforgeeks.org/c-sharp/what-is-entity-framework-in-net-framework/. [Hozzáférés dátuma: 20 10 2025]. |
| [6] | „Microsoft Learn,” Microsoft, [Online]. Available: https://learn.microsoft.com/en-us/aspnet/core/mvc/views/razor?view=aspnetcore-9.0. [Hozzáférés dátuma: 21 10 2025]. |
| [7] | „Razor Syntax Reference,” [Online]. Available: https://whosnailaspnetcoredocs.readthedocs.io/ko/latest/mvc/views/razor.html. [Hozzáférés dátuma: 29 10 2025]. |
| [8] | „Bootstrap Documentation,” Bootstrap Authors, 2024. [Online]. Available: https://getbootstrap.com/docs/5.3/getting-started/introduction/. [Hozzáférés dátuma: 09 11 2025]. |
| [9] | S. Bhattachergee, „Fabric.js: A Closer Look at Its Core Features,” Medium, 7 11 2023. [Online]. Available: https://medium.com/@syketweb/fabric-js-a-closer-look-at-its-core-features-a23aded4d61f. [Hozzáférés dátuma: 31 10 2025]. |
| [10] | Firebase, „Cloud Storage for Firebase Documentation,” Google, 2024. [Online]. Available: https://firebase.google.com/docs/storage. [Hozzáférés dátuma: 03 11 2025]. |
| [11] | GeeksforGeeks, „Introduction to Firebase Cloud Storage,” 2023. [Online]. Available: https://www.geeksforgeeks.org/firebase/introduction-to-firebase-cloud-storage/. [Hozzáférés dátuma: 03 11 2025]. |
| [12] | Firebase, „Get Started with Cloud Storage on Web,” Google, 2024. [Online]. Available: https://firebase.google.com/docs/storage/web/start. [Hozzáférés dátuma: 03 11 2025]. |
| [13] | Ashutec, „What is Google Firebase Storage and How to Use It for Application Development,” 2023. [Online]. Available: https://www.ashutec.com/blog/what-is-google-firebase-storage-and-how-to-use-it-for-application-development-4f9f462d1487. [Hozzáférés dátuma: 03 11 2025]. |
| [14] | A. P. Automate, „Firebase Cloud Storage Review,” 2023, [Online]. Available: https://www.appypieautomate.ai/blog/reviews/firebase-cloud-storage-review. [Hozzáférés dátuma: 03 11 2025]. |
| [15] | F. &. G. C. Platform, „Firebase and GCP Integration,” Google, 2024. [Online]. Available: https://firebase.google.com/firebase-and-gcp. [Hozzáférés dátuma: 03 11 2025]. |
| [16] | Microsoft, „Overview of ASP.NET Core MVC,” Microsoft Learn, 17 06 2024. [Online]. Available: https://learn.microsoft.com/en-us/aspnet/core/mvc/overview?view=aspnetcore-9.0. [Hozzáférés dátuma: 07 11 2025]. |
| [17] | Microsoft, „Dependency injection in ASP.NET Core,” Microsoft Learn, 2024. [Online]. Available: https://learn.microsoft.com/en-us/aspnet/core/fundamentals/dependency-injection?view=aspnetcore-9.0. [Hozzáférés dátuma: 10 11 2025]. |
| [18] | M. Fowler, „Repository,” Patterns of Enterprise Application Architecture, [Online]. Available: http://martinfowler.com/eaaCatalog/repository.html. [Hozzáférés dátuma: 08 11 2025]. |
| [19] | M. Ghanbari, „Code First vs. Database First: A Pragmatic Guide for Modern Software Engineers,” Medium, 12 07 2025. [Online]. Available: https://medium.com/@MatinGhanbari/code-first-vs-database-first-a-pragmatic-guide-for-modern-software-engineers-29ac04f1daa1. [Hozzáférés dátuma: 10 11 2025]. |
| [20] | Microsoft, „Creating and Configuring a Model - EF Core: Use fluent API to configure a model,” Microsoft Learn, 2024. [Online]. Available: https://learn.microsoft.com/en-us/ef/core/modeling/#use-fluent-api-to-configure-a-model. [Hozzáférés dátuma: 10 11 2025]. |
| [21] | Microsoft, „Migrations in EF Core,” Microsoft Learn, 01 12 2023. [Online]. Available: https://learn.microsoft.com/en-us/ef/core/managing-schemas/migrations/?view=efcore-7.0. [Hozzáférés dátuma: 08 11 2025]. |
| [22] | D. Richard, „Distinctive Features of SQLite,” SQLite.org, 31 05 2025. [Online]. Available: https://www.sqlite.org/different.html. [Hozzáférés dátuma: 08 11 2025]. |
| [23] | D. Cookies, „Understanding Object-Relational Mapping (ORM),” Medium, 03 09 2024. [Online]. Available: https://devcookies.medium.com/titleunderstanding-object-relational-mapping-orm-why-and-when-you-should-use-it-a18decc410ba. [Hozzáférés dátuma: 08 11 2025]. |
| [24] | M. Learn, „Model validation in ASP.NET Core MVC,” Microsoft Learn, 2024. [Online]. Available: https://learn.microsoft.com/en-us/aspnet/core/mvc/models/validation?view=aspnetcore-9.0. [Hozzáférés dátuma: 09 11 2025]. |
| [25] | GitHub, „What is Version Control?,” 2024. [Online]. Available: https://github.com/resources/articles/what-is-version-control. [Hozzáférés dátuma: 09 11 2025]. |
| [26] | GitHub, „GitHub Flow,” GitHub Docs, 2024. [Online]. Available: https://docs.github.com/en/get-started/using-github/github-flow. [Hozzáférés dátuma: 09 11 2025]. |
| [27] | G. Docs, „About pull requests,” GitHub, [Online]. Available: https://docs.github.com/en/pull-requests/collaborating-with-pull-requests/proposing-changes-to-your-work-with-pull-requests/about-pull-requests. [Hozzáférés dátuma: 10 11 2025]. |
| [28] | M. Learn, „About pull requests and permissions,” Microsoft Learn, 2024. [Online]. Available: https://learn.microsoft.com/en-us/azure/devops/repos/git/about-pull-requests?view=azure-devops. [Hozzáférés dátuma: 10 11 2025]. |
| [29] | C. R. S. H. T. Dykstra, „Kestrel web server in ASP.NET Core,” Microsoft Learn, 23 10 2023. [Online]. Available: https://learn.microsoft.com/en-us/aspnet/core/fundamentals/servers/kestrel?view=aspnetcore-9.0. [Hozzáférés dátuma: 11 11 2025]. |
| [30] | Microsoft, „Host and deploy ASP.NET Core,” Microsoft Learn, 2024. [Online]. Available: https://learn.microsoft.com/en-us/aspnet/core/host-and-deploy/?view=aspnetcore-9.0. [Hozzáférés dátuma: 11 11 2025]. |

# Nyilatkozat

Alulírott Mészáros Márton, Gazdaságinformatika BSc szakos hallgató, kijelentem, hogy a dolgozatomat a Szegedi Tudományegyetem, Informatikai Intézet Szoftverfejlesztés Tanszékén készítettem, Gazdaságinformatikus diploma megszerzése érdekében.

Kijelentem, hogy a dolgozatot más szakon korábban nem védtem meg, saját munkám eredménye, és csak a hivatkozott forrásokat (szakirodalom, eszközök, stb.) használtam fel.

Tudomásul veszem, hogy szakdolgozatomat a Szegedi Tudományegyetem Diplomamunka Repozitóriumában tárolja.

Szeged, 2025.11.22.

Mészáros Márton