Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
Automatizálási és Alkalmazott Informatikai Tanszék

Bui Quanganh Krisztián

INGATLAN HÍRDETŐ PORTÁL MEGVALÓSÍTÁSA

KONZULENS

Benedek Zoltán

BUDAPEST, 2023

Tartalomjegyzék

HALLGATÓI NYILATKOZAT

Alulírott **Bui Quanganh Krisztián**, szigorló hallgató kijelentem, hogy ezt a önálló laboratórium dokumentációt 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, 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, 2023. 06. 05.

...…….

Bui Quanganh Krisztián

1 Felhasznált technológiák

1.1 .NET

A .NET [1] (korábban ismert nevén .NET Core) egy ingyenes és nyíltforráskódú szoftverkeretrendszer, amit 2016-ban mutatott be a Microsoft. A .NET Framework jelentősen átdolgozott cross-platform utódja, így futtatható Windows, Linus és MacOS rendszereken egyaránt.

A .NET alkalmazások magja a robosztus és felügyelt közös nyelvi futtatókörnyezet (Common Language Runtime - CLR) [2]. Többféle szolgáltatást nyújt, ilyen a memóriakezelés a szemétgyűjtő segítségével, kivételkezelés, típusellenőrzés és szálkezelés. A CLR biztosítja továbbá a különböző programozási nyelvek együttműködését, ezért egy alkalmazás könnyedén felhasználhatja a különböző .NET által támogatott nyelveken írt könyvtárakat.

A .NET környezet nyelvfüggetlen, C#, Visual Basic, illetve az F# mellett számos objektumorientált programozási nyelvet támogat, a különböző nyelveken készült osztályok egymásból örökölhetnek. Az alkalmazás fordításakor első sorban IL (Intermediate Language) kód készül, mely több architektúrára és operációs rendszerre fordítható, majd a platformtól függően natív kód áll elő, ami Just-in-time vagy Ahead-of-time módon fut le.

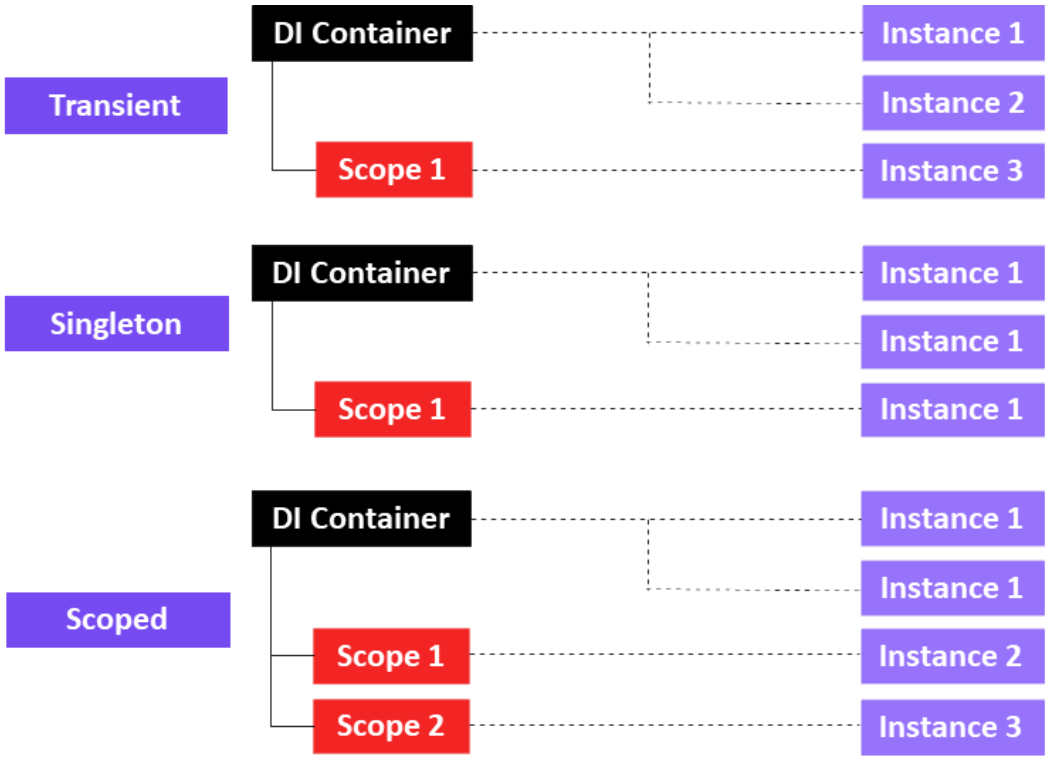
Ábra 1: Kód fordítása .NET környezetben.

A keretrendszer moduláris felépítésű és csak alapvető osztályokat tartalmaz, így a környezet kis méretű marad, az alkalmazás pedig csak a használt könyvtárakat tartalmazza. További komponenseket a NuGet csomagkezelő rendszeren keresztül tudunk hozzáadni.

1.2 ASP.NET Core

Az ASP.NET Core [3] egy ingyenes, nyíltforráskódú keretrendszer, melyet modern web alapú alkalmazások készítésére fejlesztettek ki. A keretrendszer segítségével készíthetünk szerver oldalon renderelt multi-page alkalmazásokat (Razor pages), single-page alkalmazásokat (React, Angular) vagy REST (Representational State Transfer) API (Application Programming Interface) szolgáltatást, amelyt használhatunk tetszőleges kliensoldali technológia alkalmazásához.

Az ASP.NET támogatja a függőséginjektálást (Dependency injection) tervezési mintát. A minta lehetővé teszi azt, hogy egy osztály és azon függőségei között laza csatolás jöjjön létre azáltal, hogy az osztály nem függ az implementációtól. Az osztály a függőség szolgáltatásaihoz egy interfészen keresztül tud hozzájutni. Az esetleges függőségeket a Program.cs fájlban tudjuk rögzíteni, és a keretrendszer az osztály konstruktorán keresztül adja át a megfelelő függőségeket. Az injektált függőségeknek többféle életciklust [4] tudunk beállítani: Transient esetén minden használatkor új objektum jön létre, ami többszálú alkalmazásban előny lehet; a Scoped objektum egy kérésen belül ugyanaz marad; a Singleton objektum esetén a kérések ugyanazt az egy objektumot használják.



Ábra 2: A életciklusok típusainak működése

1.3 ASP.NET Identity

Az ASP.NET Identity [5] a Microsoft által biztosított felhasználóikezelő rendszer, amely biztonságos hitelesítési és authorizációs funkciók kiépítésére szolgál az ASP.NET alkalmazásokban. Az ASP.NET Core keretrendszer része, amelyet a regisztráció, a bejelentkezés, a jelszókezelés és a szerepek kezelésére terveztek.

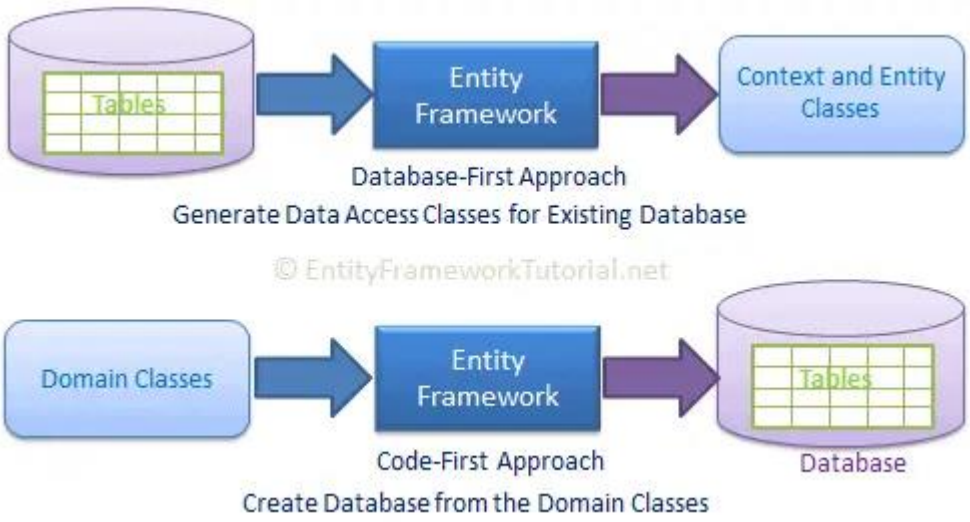
Az ASP.NET Identity több hitelesítési mechanizmust is támogat, használhatunk külső bejelentkezés szolgáltatókat, például Facebook, Google vagy Microsoft felhasználókat, vagy eltárolhatjuk a bejelentkezési adatokat az Identity rendszer segítségével és használhatunk cookie-alapú és token-alapú hitelesítést JSON Web Token (JWT) segítségével.

A hitelesítés és authorizáció egyik legfontosabb része a biztonság. Mivel én ebben a területben nem vagyok elég jártas, a felhasználókezelő saját implementálása helyett egy előre megírt és jól dokumentált megoldás mellett döntöttem. Az ASP.NET Identity és az Entity Framework Core szorosan integrálódik egymással, ezért ennek a könyvtár használatát választottam.

1.4 Entity Framework Core

Az Entity Framework Core (EF Core) [6] egy Microsoft által létrehozott objektum-relációs leképező könyvtár. A szoftver platformfüggetlen, használhatjuk Windows, MacOS és Linus operációs rendszeren, továbbá kompatibilis a legtöbb adatbáziskezelő rendszerrel, például a Microsoft SQL Server-hez, a MySQL-hez és a PostgreSQL-hez. Ennek használatának segítségével függetleníteni tudjuk alkalmazásainkat az adatbázismotortól.

Az EF Core egyik fő funkciója a code-first leképezési módszer. A fejlesztőnek van lehetősége arra, hogy elsődlegesen az alkalmazás objektummodelljét definiálja majd a keretrendszer segítségével hozza létre a kapcsolódó adatbázis sémát. A könyvtár egy migrációs rendszert is tartalmaz, ami az adatsémában való változtatások lebonyolítását segíti az adatok elvesztése nélkül.



Ábra 3: A database-first és a code-first megközelítés közötti különbség

Az EF Core és a LINQ (Language Integrated Query) rendszer együttműködésével könnyen tudunk olvasható adatbázis lekérdezéseket írni C# szintaxis használatával. A LINQ típusbiztonságot is biztosít, és olyan objektumokon is használhatjuk, amik implementálják az IEnumerable és az IQueryable interfészeket.

1.5 TypeScript

A TypeScript [7] egy magasszintű, objektum-orientált, böngészőkben használt, nyíltforráskódú programozási nyelv, ami a JavaScript kiterjesztése. A TypeScript fordító JavaScript kódot hoz létre, ezért minden JavaScript-et támogató környezetben futtatható, és minden JavaScript kód szintaktikailag érvényes TypeScript kód is. A nyelv funkciói jelentősen javítja a kód karbantarthatóságát és olvashatóságát.

Ábra 4: A fordító TypeScript kódból JavaScript kódot készít.

A TypeScript statikus típusosságot biztosít, így lehetővé teszi a típusellenőrzést fordítási időben. Ez jelentősen megkönnyíti a fejlesztést, mivel a legtöbb futási idejű hiba

kiderül fordításkor, továbbá valódi IntelliSense támogatást is nyújt. A nyelv segítségével saját osztályokat és interfészeket is létrehozhatunk, melyeknek adattagokat és függvényeket adhatunk. Strukturális típusosság jellemzi, vagyis egy objektum strukturálisan kompatibilis egy másik objektummal, ha tartalmazza a publikusan elérhető tagváltozóit és függvényeit.

Alapértelmezetten a változók felvehetik a null és undefined értékeket, melyek további futási idejű hibához vezethetnek. A fordító `--strictNullChecks` flag beállítása esetén a fordító típushibát dob, ha egy típusos változónak az előbb említett értékek egyiket adjuk meg.

A TypeScript modulok használatát is biztosítja, ezek segítségével összefüggő osztályokat, függvényeket, változókat tudunk összefűzni egy logikai fájlba. Alapesetben egy modul tartalma csak exportálás után láthatóak kívülről, és minden használni kívánt elem importálás után használhatóak.

1.6 React

A React [8] egy ingyenes, nyíltforráskódú, JavaScript keretrendszer, melyet a Meta (korábban Facebook) fejlesztett ki. A könyvtár segítségével felhasználói felületet tudunk létrehozni. A felület leírását deklaratív módon tudjuk leírni, a keretrendszer automatikusan frissíti és újrarendereli a komponenseket a fájlok változtatásakor, ez jelentősen felgyorsíthatja a fejlesztést és megkönnyíti a hibakeresést. A keretrendszerhez mellékeltek típus fájlokat is, ezért TypeScript projektben is könnyedén használható.

A React segítségével komponens alapú felhasználói felületet tudunk létrehozni. A komponenseket kétféleképpen tudjuk leírni: osztálykomponensként vagy függvénykomponensként. Az osztálykomponensek ES6 osztályokként jelennek meg, amelyek öröklik a React.Component ősosztályt és életciklus függvényeket alkalmaznak. Állapotok eltárolásához a ‘state’ objektumot használták, a külső paramétereket pedig a ‘props’ objektumon keresztül érjük el. A függvénykomponensek egyszerű függvényekként jelennek meg, a külső paramétereket a függvény paramétereiként kell átadni. Eleinte a függvénykomponenseknek nem volt állapotkezelő rendszere és életciklus függvényeik, de a könyvtár 16.8-as verziójában behozták a ‘hook’ függvényeket, melyek segítségével már lehetséges volt. A függvénykomponensek később népszerűbbek lettek az osztálykomponensekhez képest a hook-ok bevezetése óta az egyszerűségük miatt.

A komponensek leírásához a React új szintaxisokat vezetett be, amelyet JSX-nek (JavaScript HTML) neveztek el. A JSX a JavaScript bővítménye, amely segítségével HTML-szerű jelöléseket tudunk használni a React komponensek leírásához. Az alkalmazás fordításakor a HTML leírás függvény hívásokká alakulnak át, amelyek HTML elemeket hoznak létre, erre egy példa lent látható:

```
const hello1 = <div>Hello world!</div>;
const hello2 = React.createElement('div', null, 'Hello world!');
```

A React mellé célszerű további könyvtárakat is használni, amelyek kibővítik a keretrendszer funkcionalitását, ilyen például a React Router, a React Hook Forms vagy a React Query.

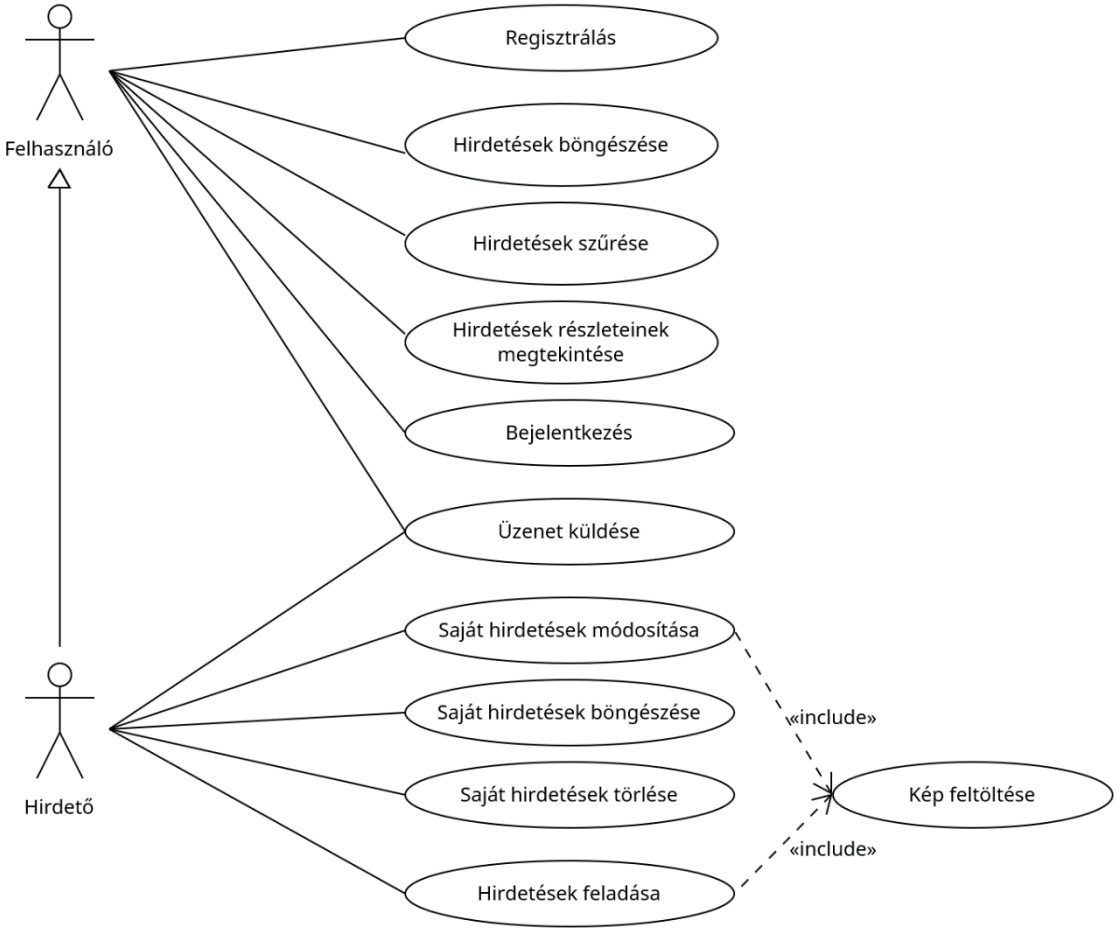
1.7 Chakra UI

A Chakra UI egy keretrendszer, amely egy rendszert biztosít saját stílusok definiálásához és komponensek testreszabásához. A könyvtár tartalmaz rengeteg előre definiált React komponenseket, amelyeket felhasználhatunk a felhasználói felület leírásához. A Chakra UI reszponzív tervezési funkciókat is nyújt, így könnyen tudunk a képernyőméretekhez alkalmazkodó felületeket készíteni.

A könyvtár komponenseinek stílus prop-okat tudunk megadni, amelyek megfelelhetőek inline CSS szabályoknak. Rövidítéseket is elfogadnak és ezzel gyorsabban meg tudtam tervezni a felületek kinézetét, de a projektemben a komponens véglegesítése után a szabályokat kiszerveztem egy külön fájlba könnyebb átláthatóság és újrafelhasználhatóság miatt.

Funkcionális követelmények

Ebben a fejezetben a funkcionális követelményeket fogom felsorolni, amelyek a megvalósítandó alkalmazáshoz nélkülözhetetlenek. A funkciókról készült felépítéséről készült ábra lentebb látható.



Ábra 5: A végrehajtható funkciók felhasználók szempontjából

1.8 Böngészés

Az egyik nélkülözhetetlen funkció az alkalmazásban a létrehozott hirdetésének böngészése. A felhasználó bejelentkezés nélkül csak ezt a funkciót éri el. A hirdetések egy listán keresztül tekinthetőek meg, melynek szűrési lehetőségeket kell biztosítania. Szűrési feltételnek megadható az ingatlan címe, ára, mérete és a szobák száma. A listanézetben a hirdetések száma korlátozott a könnyebb átláthatóság és navigálás érdekében, lapozással lehet a többi hirdetést megjeleníteni.

Ha egy hirdetés felkeltette a felhasználó érdeklődését, a listában lévő hirdetésre kattintva meg kell jelennie az ingatlan részletes leírásának. Ennek a nézetnek egy külön oldalon kell megjelennie, a részletek között meg kell jelennie az ingatlan adatainak, egy róla készült fotónak és a hirdető elérhetőségeinek. A bejelentkezett felhasználónak van lehetősége felvenni a kapcsolatot az alkalmazáson keresztül is.

1.9 Regisztrálás és bejelentkezés

Ahhoz, hogy elérje az alkalmazás további funkcióit, a felhasználónak rendelkeznie kell egy saját fiókkal, amit a regisztrációs ablakon tud létrehozni. Regisztráció esetén meg kell adni egy érvényes e-mail címet, felhasználónevet, megfelelő erősségű jelszót és opcionálisan egy telefonszámot. A regisztráció és bejelentkezés után elérhetővé válik a hirdetés készítése és az üzenetek küldése.

1.10 Hirdetés létrehozása és kezelése

A másik fontos funkció az alkalmazásban az ingatlan hirdetések létrehozása. A hirdetés létrehozásakor meg kell adni a hirdetés címét, a lakás címét, méretét, szobák számát, egy róla elkészült képet és az árát. A kliensnek és a szervernek validációt kell elvégeznie a kérés feldolgozása előtt. A képek eltárolása az adatbázisban történik, amit Base64 formájú szövegként tárolunk el. A hirdetőnek van lehetősége az saját hirdetéseit kilistáznia egy külön nézetben, itt eltávolíthatja a már nem aktuális hirdetést vagy módosíthatja a már elavult információkat.

1.11 Üzenetek küldése

Az üzenetváltás elkezdését az érdeklődőnek kell kezdeményeznie egy ingatlan hirdetés részletes nézetén keresztül. Az üzenet elküldése után egy másik nézet fog elénk tárulni, amin az összes üzenetet el tudjuk olvasni és további üzeneteket is tudunk küldeni. Ha egy felhasználónak vannak olvasatlan üzenetei, akkor a kliens menübárban kijelzi egy számláló segítségével. Ha az üzenetekhez navigál a felhasználó, akkor az összes vele kapcsolatos párbeszédeket látni fogja. A párbeszédre kattintva frissülnie kell a menübárban lévő számlálónak, meg kell jelennie az egymásnak elküldött üzeneteknek és azoknak az elküldési idejüknek.

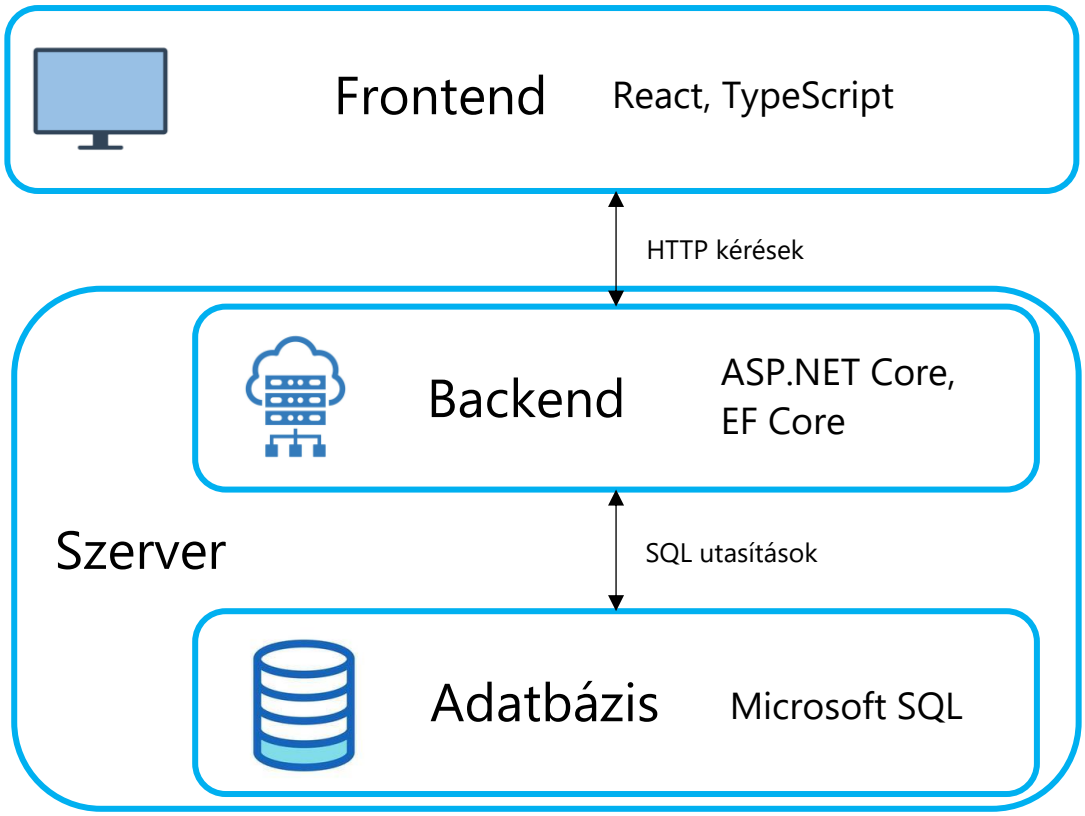
1.12 További követelmények

A felhasználó felületnek letisztultnak, reszponzívnak és intuitívnak kell lennie. A kliensnek a felhasználó legtöbb műveletére reagálnia kell, a hosszabb műveletek esetén aszinkron működést kell biztosítani, hogy ne fagyjon le a felhasználói felület. Az esetleges hibákat a kliensnek jeleznie kell.

2 Architektúrák

2.1 Rendszer architektúra

Az ingatlan hirdető alkalmazás három fő komponensből áll: az alkalmazást megjelenítő kliensből, amin keresztül a felhasználó tudja használni a szolgáltatást, a kiszolgáló szerverből, amely az alkalmazás logikáját valósítja meg, és az adatbázisból, ahol az adatokat tároljuk el. A felhasználó egy React alkalmazást futtató böngészőn keresztül tartja a kapcsolatot a kiszolgáló szerverrel HTTP kérések segítségével. A szerver egy REST interfészen keresztül szolgálja ki a klienseket, amit az ASP.NET Core technológiával valósít meg, és az Entity Framework Core könyvtár segítségével kommunikál az MSSQL adatbázissal.

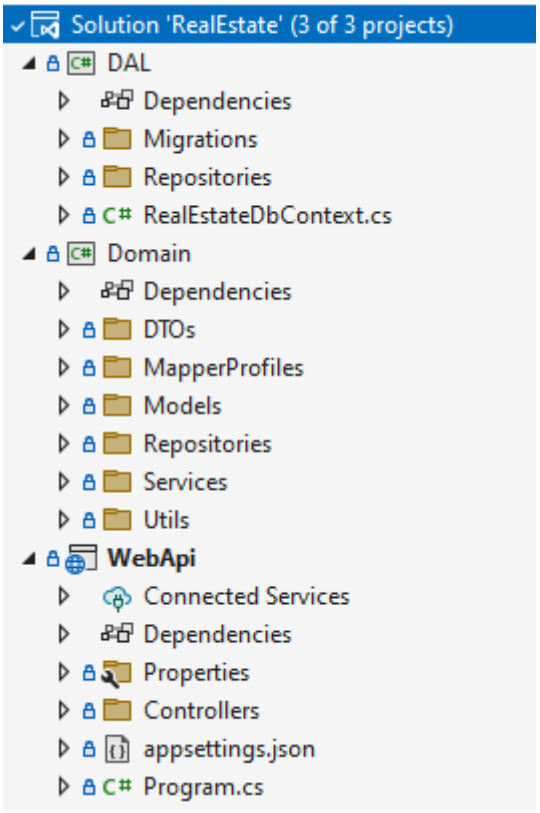


Ábra 6: Az alkalmazás komponensei

2.2 Szerveroldali architektúra

A szerveroldali komponens több részre osztottam, amelyeket a Domain Driven Design elv szerint definiáltam. A szerver magjában a Domain réteg áll, ez tartalmazza az

üzleti logikát implementáló szolgáltatásokat és a felhasznált entitásosztályokat, a külső erőforrások eléréséért a Data Access Layer (DAL) felelős, ami az adatbázissal kommunikál Entity Framework segítségével, és a WebApi teszi lehetővé azt, hogy egy REST API-n keresztül felhasználhassuk a Domain szolgáltatásait.



Ábra 7: A szerveroldali komponens mappaszerkezete

2.2.1 Domain réteg

A Domain réteget több bounded context-ekre bontottam, amelyek különböző szolgáltatásokra fókuszálnak (ilyenek a hirdetések kezelése, üzenetek kezelése és felhasználók kezelése), és egységbe zárják a funkciók megvalósítását. A szolgáltatásoknak el kell érniük az adatot ahhoz, hogy megfelelően tudjanak működni, de fontos az adatbázistól való függetlenség megtartása. A Repository tervezési minta használatával elérhetjük azt, hogy a szolgáltatások absztrakcióktól függjenek és ne az implementációtól, könnyedén le tudjuk cserélni az adatelérés implementációját és a tesztelést is megkönnyítheti. A szolgáltatások általában nem az entitás összes attribútumait adja vissza, mivel lehetnek benne olyan érzékeny adatok, amiket nem lenne érdemes elérhetővé tenni a felhasználó számára, ezért Data Transfer Object-eket (DTO) használtam az adatok, amikben csak a szükséges mezők láthatóak.

2.2.2 REST API

A REST API-t a WebApi nevű projektben valósítottam meg, ami az ASP.NET Core könyvtárt használja. Minden egyes szolgáltatáshoz tartozik egy Controller osztály, amik a Domain-ben létrehozott szolgáltatásokat használják, itt találhatóak a definiált REST interfészek és ezek kezelik a http kéréseket. A Program.cs fájlban a szerver konfigurációját írhatjuk le, a függőség injektáláshoz szükséges osztályobjektumokat itt tudjuk leírni. Ez a réteg felelős az authentikációért is, ellenőrzi, hogy az adott kéréshez tartozó access token érvényességét, és feloldja a hozzátartozó felhasználót.

2.2.3 Adatelérési réteg

Az adatelérési rétegben az adatbázissal való kommunikációt valósítom meg Entity Framework Core segítségével. A réteg több repository-kból áll, amelyek a Domain-ben definiált interfészeket implementálja, ezek kezelik a CRUD műveleteket (Create, Read, Update, Delete) az adatbázisban lévő entitásokon. Az adatbáziskapcsolat a RealEstateDbContext osztályban jön létre, amely a DbContext ősosztályt örökli, DbSet-eken keresztül lehet lekérdezni az entitásokat, amely támogatja a LINQ kifejezéseket is.

2.3 Adatbázis felépítése

Az adatbázis kezeléséhez Microsoft SQL adatbázisszervert alkalmaztam. A tervezéskor Code first megközelítést alkalmaztam az Entity Framework Core segítségével, a táblázat szerkezete követi a Domain rétegben definiált modellek szerkezetét. Az adatbázis szerkezet az 8. ábrán látható.

Az ASP.NET Core Identity több felhasználóval kapcsolatos táblázatot is generált, de azoknak a szerepe elhanyagolható, ezért nem szerepelnek. Az ábrán lévő táblázatokat tekintsük át.

2.3.1 AspNetUsers

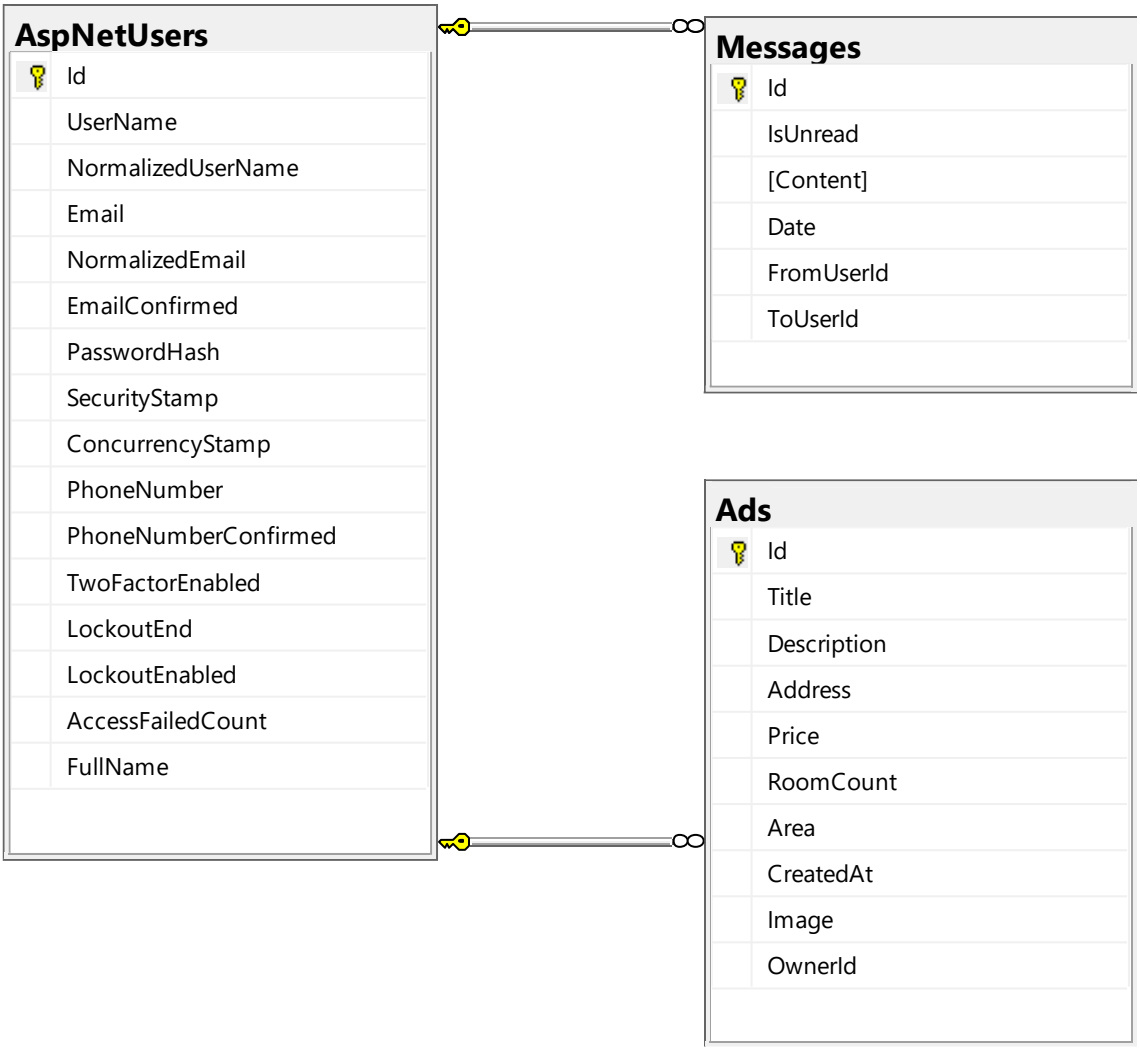
A felhasználó fontos adatait tartalmazó táblázat. Az adatok között szerepel a felhasználó e-mail címe, teljes neve, felhasználóneve, jelszava és opcionálisan a telefonszáma. A további, fel nem sorolt attribútumok az ASP.NET Core Identity működéséhez szükséges. A jelszó hashelt formában érhető el.

2.3.2 Ads

Ebben a táblázatban az ingatlanok hirdetésének adatait tárolom el. Szerepel benne a hirdetés címe, leírása, létrehozási ideje, tulajdonos idegen kulcsa, az ingatlan, címe, ára, mérete és a róla készült kép Base64 formátumban.

2.3.3 Messages

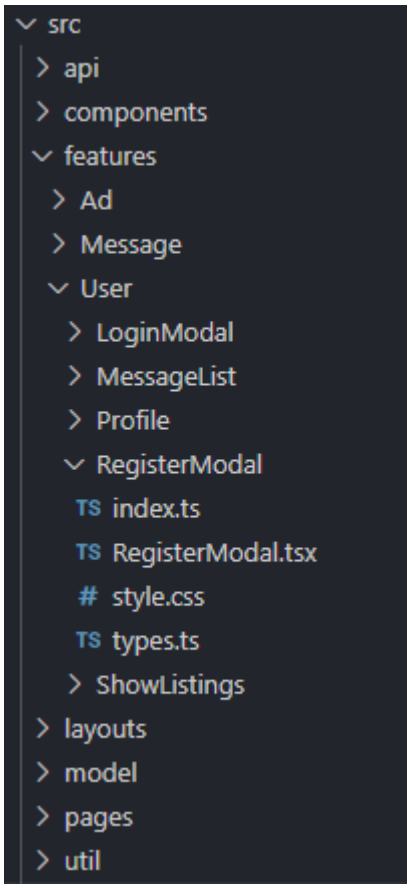
A felhasználók közötti üzenetváltásokat tárolom ebben a táblázatban. A küldött üzenet tartalmát, idejét, olvasottságát, feladóját és címzettét tárolja el.



Ábra 8: Az adatbázis sémája

2.4 Kliensalkalmazás

A kliensalkalmazás megvalósításához React keretrendszert használtam. A kliens megírása közben arra törekedtem, hogy a projekt mappaszerkezete átlátható és jól struktúrált legyen, külön mappát hoztam létre a REST API-t hívó függvényeknek; a komponenseknek; a modelleknek, amelyek a http kéréseknél kerülnek elő; és az adott oldal struktúráját leíró fájloknak. A komponenseket is több részre osztottam: a features mappában funkcióhalmazonként vannak rendezve az elemek, mivel ezek egy adott oldal részét képezik és nehéz őket általánosan újrahasználni őket, a components mappában pedig könnyen újrahasználható komponensek helyezkednek el.



Ábra 9: A kliens mappaszerkezete

Egy UI komponenshez tartozó mappán belül több fájl is található: az egyik fájl tartalmazza a felület deklaratív leírását és működését, a style.css-ben a komponens megformázása található, és a types.ts fájlban az erősen hozzákötethető típusdefiníciókat tartalmazza.

3 Irodalomjegyzék

[1] Microsoft, „What is .NET? Introduction and overview,” Microsoft, 15. 03. 2023.. [Online]. Available: https://learn.microsoft.com/en-us/dotnet/core/introduction. [Hozzáférés dátuma: 13. 05. 2023.].

[2] Microsoft, „Common Language Runtime (CLR) overview,” Microsoft, 25. 04. 2023.. [Online]. Available: https://learn.microsoft.com/en-us/dotnet/standard/clr. [Hozzáférés dátuma: 13. 05. 2023.].

[3] Microsoft, „Overview of ASP.NET Core,” Microsoft, 15. 11. 2022.. [Online]. Available: https://learn.microsoft.com/en-us/aspnet/core/introduction-to-aspnet-core?view=aspnetcore-7.0. [Hozzáférés dátuma: 13. 05. 2023.].

[4] Wade, „.NET Core Dependency Injection Lifetimes Explained,” .NET Core Tutorials, [Online]. Available: https://dotnetcoretutorials.com/net-core-dependency-injection-lifetimes-explained/. [Hozzáférés dátuma: 13. 05. 2023.].

[5] R. Anderson, „Introduction to Identity on ASP.NET Core,” Microsoft, 01. 12. 2022. [Online]. Available: https://learn.microsoft.com/en-us/aspnet/core/security/authentication/identity?view=aspnetcore-7.0&tabs=visual-studio. [Hozzáférés dátuma: 14. 05. 2023.].

[6] Microsoft, „Entity Framework Core,” Microsoft, 25. 05. 2021.. [Online]. Available: https://learn.microsoft.com/en-us/ef/core/. [Hozzáférés dátuma: 14. 05. 2023.].

[7] Microsoft, „TypeScript,” Microsoft, [Online]. Available: https://www.typescriptlang.org/. [Hozzáférés dátuma: 15. 05. 2023.].

[8] Meta, „React,” Meta, [Online]. Available: https://react.dev/. [Hozzáférés dátuma: 15. 05. 2023.].