

Budapesti Műszaki és Gazdaságtudományi Egyetem

Villamosmérnöki és Informatikai Kar

Bui Quanganh Krisztián

Ingatlan hirdető portál megvalósítása

Konzulens

Benedek Zoltán

BUDAPEST, 2023

Tartalomjegyzék

[Összefoglaló 5](#_Toc149406524)

[Abstract 6](#_Toc149406525)

[1. Bevezetés 7](#_Toc149406526)

[1.1 Témaválasztás 7](#_Toc149406527)

[1.2 Technológiai háttér 7](#_Toc149406528)

[1.3 A szakdolgozat felépítése 8](#_Toc149406529)

[2. Felhasznált technológiák 9](#_Toc149406530)

[2.1 .NET 9](#_Toc149406532)

[2.2 ASP.NET Core 10](#_Toc149406533)

[2.3 ASP.NET Identity 11](#_Toc149406534)

[2.4 Entity Framework Core 11](#_Toc149406535)

[2.5 SignalR 12](#_Toc149406536)

[2.6 Quartz.NET 13](#_Toc149406537)

[2.7 TypeScript 14](#_Toc149406538)

[2.8 React 15](#_Toc149406539)

[2.9 Chakra UI 16](#_Toc149406540)

[2.10 Tervezési minták 16](#_Toc149406541)

[2.10.1 Domain Driven Design 16](#_Toc149406542)

[2.10.2 Repository Pattern 17](#_Toc149406543)

[3. Funcionális követelmények 19](#_Toc149406544)

[3.1 Böngészés 19](#_Toc149406546)

[3.2 Regisztrálás és bejelentkezés 20](#_Toc149406547)

[3.3 Hirdetés létrehozása és kezelése 20](#_Toc149406548)

[3.4 Előfizetés 20](#_Toc149406549)

[3.5 Üzenetek küldése 21](#_Toc149406550)

[3.6 További követelmények 21](#_Toc149406551)

[4. Architektúrák 22](#_Toc149406552)

[4.1 Rendszer architektúra 22](#_Toc149406554)

[4.2 Szerveroldali architektúra 23](#_Toc149406555)

[4.2.1 Domain réteg 23](#_Toc149406556)

[4.2.2 Web API 24](#_Toc149406557)

[4.2.3 Adatelérési réteg 25](#_Toc149406558)

[4.3 Adatbázis felépítése 25](#_Toc149406559)

[4.3.1 AspNetUsers 26](#_Toc149406560)

[4.3.2 Ads 26](#_Toc149406561)

[4.3.3 Messages 27](#_Toc149406562)

[4.3.4 SubscriptionTiers 27](#_Toc149406563)

[4.3.5 Subscriptions 27](#_Toc149406564)

[4.4 Kliensalkalmazás 27](#_Toc149406565)

[5. Megvalósítás 29](#_Toc149406566)

[6. Irodalomjegyzék 30](#_Toc149406567)

Hallgatói nyilatkozat

Alulírott **Bui Quanganh Krisztián**, szigorló hallgató kijelentem, hogy ezt a szakdolgozatot 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. 12. 05.

...…………………………………………….

Bui Quanganh Krisztián

Összefoglaló

Az ingatlanpiac folyamatosan változik és fejlődik, és egyre többen választják az online platformokat a lakások vásárlásához és eladásához. Ebben az új digitális korszakban az ingatlan hirdető portálok váltak az elsődleges eszközökké az ingatlanok értékesítéséhez és bérbeadásához. Ezen portálok nemcsak a lakáshirdetések gyűjtését teszik lehetővé, hanem számos kényelmi funkciót is nyújtanak az eladók és a vevők számára. Azonban az ilyen alkalmazások fejlesztése és működtetése számos kihívással jár, amelyeket meg kell oldani ahhoz, hogy hatékony és sikeres legyen a platform.

A szakdolgozatomban egy single-page webalkalmazást készítettem, amely lehetővé teszi azt, hogy a felhasználók meghirdethessék saját ingatlanukat, kereshessenek saját igényeiknek megfelelő lakásokat, és megkönnyíti a kommunikálást a hirdető és az érdeklődő között egy valósidejű üzenetváltás funkcióval. A felhasználóknak van lehetőségük előfizetésért fizetni is, amellyel hirdetésüket feltűnőbbé tehetik, így több ember látja az ő hirdetésüket. Az elkészítés során törekedtem egy felhasználóbarát megjelenést készíteni. A szakdolgozatban részletesen ismertetni fogom az alkalmazás tervezésének és fejlesztésének technikai részleteit is.

Az ingatlanhirdető portál megvalósításához egy többrétegű webalkalmazást hoztam létre, amely egy React keretrendszerrel elkészített webkliensből és egy ASP.NET Core-t használó kiszolgáló szerverből áll, melyek REST API és SignalR könyvtár segítségével kommunikálnak egymással. Az alkalmazás implementálása közben több olyan könyvtárat és technológiát használtam, amelyek nem fordultak elő az egyetemi képzésem során, de népszerűek fejlesztők között, és szakdolgozatom célja volt ezeknek az eszközöknek a megismerése és elsajátítása.

Abstract

The real estate market is constantly changing and developing, and more and more people are choosing online platforms to buy and sell homes. In this new digital era, real estate advertising portals have become the primary means of selling and renting real estate. These portals not only enable the collection of housing advertisements, but also provide many convenient functions for sellers and buyers. However, the development and operation of applications comes with many challenges that must be solved in order for the platform to be efficient and successful.

In my thesis, I created a single-page web application that allows users to advertise their own property, search for apartments that meet their needs, and facilitates communication between the advertiser and the interested party with a real-time message exchange function. Users also have the option to pay for a subscription to make their ad more prominent, so more people see their ad. During development, I tried to create a user-friendly interface. In the thesis, I will also describe in detail the technical details of the design and development of the application.

To implement the real estate advertising portal, I created a multi-layered web application, which consists of a web client made with the React framework and a server using ASP.NET Core, which communicate with each other using the REST API and the SignalR library. While implementing the application, I used several libraries and technologies that did not occur during my university education, but are popular among developers, and the aim of my thesis was to get to know and master these tools.

# Bevezetés

Az ingatlanhirdetések módszerei drasztikusan változtak az elmúlt évtizedek során a technológiai fejlődések miatt. Régen a kiadó vagy eladó lakásokról ingatlanújságokból vagy hirdetőtáblákon keresztül tájékozódhattunk, illetve ingatlanközvetítőktől. Ezek a módszerek valamennyire korlátozottak voltak a potenciális bérlők és vásárlók elérésében, ezek a hirdetési módszerek általában a helyi közönséget érintették.

Manapság az internet, a számítógépek és a mobiltelefonok elterjedése jelentősen megváltoztatta az ingatlanhirdetéseket. Az online hirdetőoldalak segítségével bárhol és bárhonnan tudunk hirdetéseket böngészni elektronikus eszközeink segítségével, továbbá a kommunikálást is megkönnyítik.

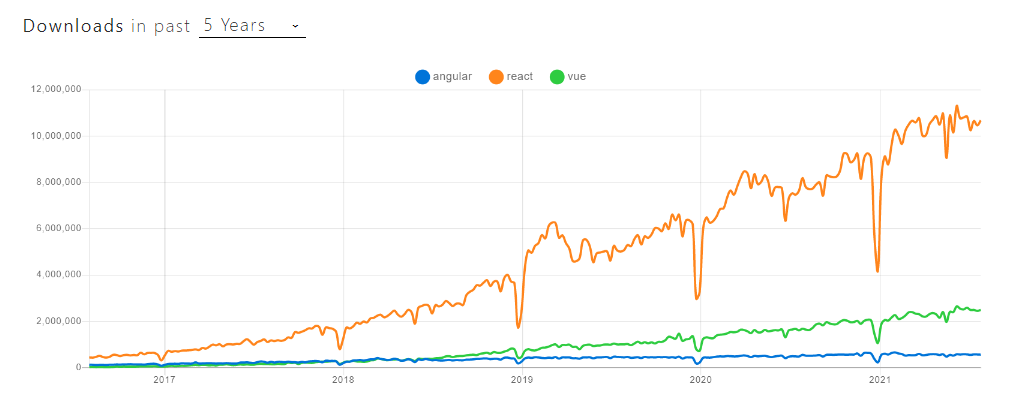
## Témaválasztás

A webalkalmazásfejlesztés egy izgalmas és dinamikus terület, amely folyamatosan változik és fejlődik. Az új technológiák és trendek folyamatosan megjelennek, és az alkalmazásoknak alkalmazkodniuk kell ezekhez a változásokhoz. A szakdolgozatom célja az volt, hogy mélyebben megértsem a webalkalmazásfejlesztés folyamatát, és hogy gyakorlati tapasztalatokat szerezzek ezen a területen.

## Technológiai háttér

Az utóbbi évtizedekben a webalkalmazások egyre nagyobb népszerűségnek örvendeznek. Már a legtöbb asztali alkalmazásnak megtalálhatjuk webalkalmazásbeli megfelelőjét, például a Microsoft termékei közül létezik a Word-nek, az Excel-nek és a többi Office 365 szolgáltatásnak böngészőből futtatható változata közel ugyanakkora (, ha nem ugyanakkora) funkcionalitással. Manapság az emberek rendszeresen használnak ilyen szolgáltatásokat, könnyedén el tudják érni őket telefonjaikon és asztali gépükön egyaránt.

Ezeknek az alkalmazások elkészítéséhez komplex UI keretrendszereket fejlesztettek ki a cégek, amelyeket nyilvánosan elérhetővé tettek. Ma az egyik legnépszerűbb a Meta (régebben Facebook) által fejlesztett React keretrendszer, amely egy rugalmas és könnyen megtanulható könyvtár, és az Angular, amely a Google terméke, és több beépített funkcióval kínál, de szigirúbb konvenciókkal rendelkezik. Mivel az Angular keretrendszer működését BSc tanulmányaim során már megismerhettem, így a React használata mellett döntöttem.



Ábra : Frontend keretrendszerek népszerűsége letöltések szerint [1]

## A szakdolgozat felépítése

A szakdolgozatom 2. fejezetében az alkalmazásom fejlesztése során felhasznált technológiákat és könyvtárakat fogom bemutatni.

A 3. fejezetben a webalkalmazáshoz kapcsolódó funkcionális és nem funkcionális követelményeket fogom részletezni.

A 4. fejezetben a portál architektúrális felépítését fogom ismertetni. Ki fogok térni a használt tervezési mintákra, és a rétegek felelősségeire.

Az 5. fejezetben az elkészült webalkalmazás funkcióit mutatom be, leírom a fejlesztés menetét, a kliens és a kiszolgáló szerver megvalósításának részleteit.

A 6. fejezetben összefoglalom a fejlesztés során szerzett tapasztalataimat, és leírom a továbbfejlesztési lehetőségeket.

# Felhasznált technológiák



## .NET

A .NET [2] (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) [3]. 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.

A képen szöveg, képernyőkép, Betűtípus, sor látható

Automatikusan generált leírás

Ábra : 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.

## ASP.NET Core

Az ASP.NET Core [4] 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 [5] 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.

A képen szöveg, képernyőkép, Betűtípus, szám látható

Automatikusan generált leírás

Ábra : A életciklusok típusainak működése

## ASP.NET Identity

Az ASP.NET Identity [6] 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.

## Entity Framework Core

Az Entity Framework Core (EF Core) [7] 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.

A képen szöveg, képernyőkép, tervezés látható

Automatikusan generált leírás

Ábra : 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.

## SignalR

A SignalR [8] egy ingyenes és nyíltforráskódú könyvtár ASP.NET keretrendszerhez, amelyet a Microsoft fejlesztett ki, és lehetővé teszi a valós idejű kommunikációt a szerver és a kliens között. Ez a technológia különösen hasznos azoknál az alkalmazásoknál, amelyek olyan funkciókat igényelnek, mint az élő csevegés, a valós idejű értesítések, az online játék stb. A könyvtár tartalmaz szerveroldali és kliensoldali JavaScript komponenseket.

A SignalR többféle módon módon tudja kezelni a valós idejű kommunikációt: WebSockets, szerver által küldött események (Server-Sent Events), vagy Long Polling segítségével. A rendszer automatikusan kiválasztja a körülményeknek legjobban megfelelő kommunikációs módszert a kliens képességei és a szerver konfigurációját figyelembe véve.

A könyvtár Hub-okat használ a kliens és a szerver közötti kommunikációhoz. A hub egy olyan interfészt nyújt a fejlesztőnek, ami lehetővé teszi azt, hogy a kliens a szerveroldali funkciókat tudjon meghívni paraméterekkel, és ugyanúgy fordítva, ezzel biztosítva a kétirányú kommunikációt. A SignalR eseménykezelést (pl. csatlakozás és lecsatlakozás), hibakezelést, felhasználó authentikációt és authorizációt is nyújt.



Ábra : A kliens és szerver közötti függvényhívás.

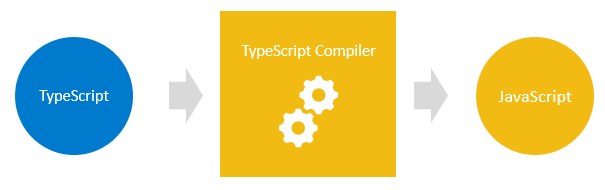
## Quartz.NET

A Quartz.NET [9] egy nyíltforráskódú feladat ütemező keretrendszer, melyet a .NET keretrendszerre fejlesztettek ki. A könyvtár segítségével könnyedén olyan feladatokat definiálni, amelyeket bizonyos időközönként vagy eseményekkor kell elvégezni. A keretrendszer erősen konfigurálható.

Az egyik funkciója könyvtárnak a Clustering, a rendszer több szerveren is képes futtatni a feladatütemezőt, amely nagy rendelkezésre állást biztosít. A Clustering segít abban figyel arra, hogy egy feladatot csak egy szerveren futtasson, ezzel megakadályozva a feladatok ismétlődését. A keretrendszer támogatja a bővítmények létrehozását is, lehetővé teszi különböző eseményekhez hallgatók hozzáadását, amelyek lehetővé teszik testre szabott műveletek vagy monitorozás hozzáadását.

## TypeScript

A TypeScript [10] 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 : 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.

## React

A React [11] 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.

## 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.

## Tervezési minták

### Domain Driven Design

A Domain Driven Design [12] (DDD, magyarul tartományalapú tervezés) egy szoftverfejlesztési területen használt tervezési irányelv. Az irányelv alapja az, hogy az alkalmazás középpontjában a domain (terület) szerepel, amely a probléma modelljei és szolgáltatásaiból áll, és a körülötte lévő struktúra vagy eszközök alkalmazkodnak hozzá.

A képen szöveg, képernyőkép, kör, Betűtípus látható

Automatikusan generált leírás

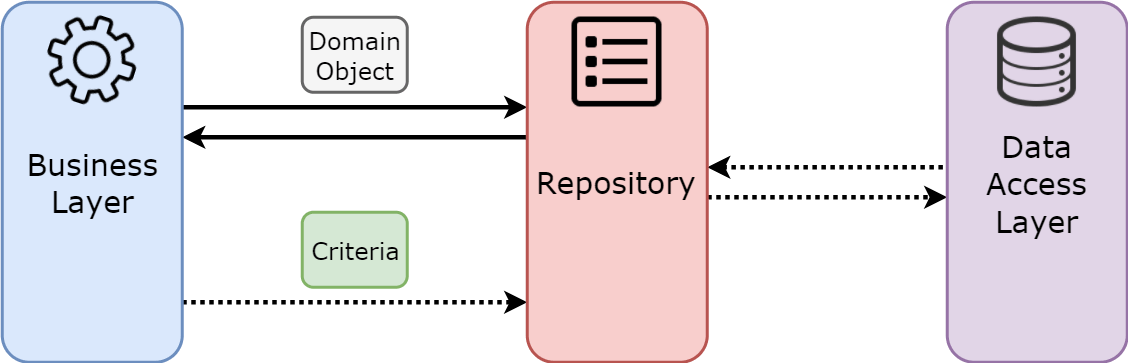
Ábra : A Domain Driven Design alapján elkészített alkalmazás felépítése

A Domain Driven Design egyik fő karakterisztikája a Bounded Context, azaz az üzleti logika felbontása olyan aldomainekre, amelyek adott problémákra fókuszálnak. Ezeknek az egységeknek általában saját modelljeik, funkcióik és szolgáltatásaik vannak, és más egységtől függetlenek. A minta használata csökkenti az elemek közötti csatolást és megkönnyíti a párhuzamos fejlesztést, különböző csapatok tudnak a saját elemein dolgozni anélkül, hogy konfliktusokba kerülnének egymással.

Az alkalmazások fejlesztésekor kulcsfontosságú a probléma többrétegű megközelítése. A belső rétegek erős összefüggése és a különböző rétegek közötti laza kapcsolatok segítik az alkalmazás karbantarthatóságát és bővíthetőségét. A DDD hasonlít a hagyományos három rétegű architektúrára, azonban az üzleti logikai réteg és az adatelérési réteg függősége megfordul, az üzleti logikai réteg függ az adatelérési rétegtől. A függőség irányának megfordításához a Repository tervezési mintát használtam.

### Repository Pattern

A Repository tervezési mintát gyakran használják a szoftverfejlesztésben, az alkalmazás adatelérési rétegében. A minta egy absztrakciós réteget definiál az alkalmazás üzleti logikája és az adatforrás között, amely segítségével laza csatolást érhetünk el a kettő réteg között, az üzleti logikának nem kell ismernie pontosan az adatelérés implementációját, hogy elérje az adatokat.



Ábra : A Repository tervezési minta általános sémája

A minta egységes interfészeket nyújt, amelyen keresztül az üzleti réteg manipulálni tudja az adatokat. Ezeket az interfészeket a különböző típusú adatforrásoknak implementálniuk kell, ezek az adatforrások lehetnek különböző gyártók adatbázismotorjait használó relációs adatbázisok, NoSQL adatbázis, fájlrendszer, vagy külső szolgáltatás is, és az egységes interfész miatt könnyedén tudunk váltani a megoldások között anélkül, hogy megváltoztatnánk az üzleti logikát.

Az interfészek használata megkönnyíti az alkalmazás részeinek tesztelését is. Mock repository osztályokat használhatunk tényleges adathozzáférés nélkül, amely segítségével függetleníteni tudjuk a tesztelendő részeket, és lehetővé teszik a tesztelést adatbázis nélkül is.

# Funcionális követelmények

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

A képen diagram, rajz, vázlat, szöveg látható

Automatikusan generált leírás

Ábra : A végrehajtható funkciók felhasználók szempontjából



## 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.

## 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.

## 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 hirdetőnek van lehetősége kiemelt hirdetést is létrehoznia, ebben az esetben a hirdetés a böngészőlista elején jelenik meg a létrehozási dátumtól függetlenül. A kiemelt hirdetés létrehozásához előfizetés szükséges. 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 a saját hirdetéseit kilistáznia egy külön nézetben, itt eltávolíthatja a már nem aktuális hirdetést vagy módosíthatja a már elavult információkat, a hirdetés kiemeltségét.

## Előfizetés

A felhasználónak van lehetősége előfizetnie különböző csomagokra az alkalmazásban. Az előfizetés előnye az, hogy az előfizető kiemelt hirdetéseket hozhat létre, amelyek a böngészési lista elején fognak megjelenni a létrehozási dátumtól függetlenül. Az előfizetés menete több lépésből áll. Az első lépésben a felhasználónak előfizetéskor ki kell választania a csomagot, amire elő szeretne fizetni. A csomagok a maximum kiemelt hirdetések számában és az előfizetés árában különböznek. A következő lépésben meg kell adni a használandó bankkártyaszámot, biztonsági kódot és a bankkártya lejárati dátumát. Az alkalmazás nem használ tényleges fizetési rendszert. A harmadik lépésben lehet átnézni a megadott információkat, és a „Subscribe” (Előfizetés) gombra kattintva tudja véglegesíteni a folyamatot. Ha a felhasználónak van előfizetése, akkor meg tudja tekinteni annak adatait.

## 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.

## További követelmények

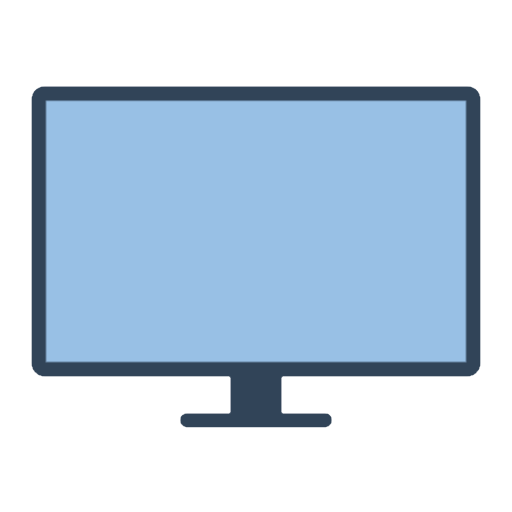
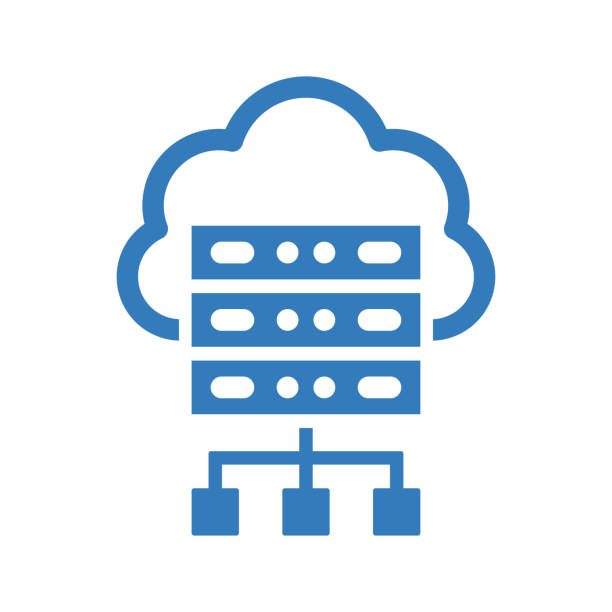
A felhasználó felületnek letisztultnak, reszponzívnak és intuitívnak kell lennie. A kliensnek a felhasználó legtöbb műveletére reagálnia kell, a hosszabb műveletek esetén aszinkron működést kell biztosítani, hogy ne fagyjon le a felhasználói felület. Az esetleges hibákat a kliensnek jeleznie kell.

# Architektúrák



## 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 a Microsoft SQL adatbázissal. Azokhoz a funkcióhoz, amelyek valós idejű kiszolgálás igényelnek, SignalR hub-okon keresztül valósulnak meg.



Frontend

Backend

Adatbázis

ASP.NET Core,   
EF Core

React, TypeScript

Microsoft SQL

SQL utasítások

HTTP kérések

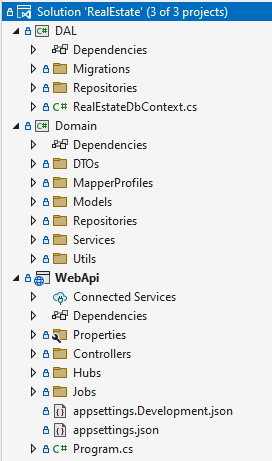
Szerver

Ábra : Az alkalmazás komponensei

Az ábrán a szerver komponensben lévő adatbázis és REST API réteget egybetartozónak jelöltem, de ezeket a részeket lehet külön fizikai gépeken vagy szolgáltatásokon futtatni megfelelő konfigurációval, ugyanez igaz a frontend architektúrára is.

## 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 Core segítségével, és a WebApi projekt teszi lehetővé azt, hogy egy REST API-on és SignalR hub-on keresztül felhasználhassuk a Domain szolgáltatásait.



Ábra : A szerveroldali komponens mappaszerkezete

### 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 továbbításához, amikben csak a szükséges mezők láthatóak.

### Web API

A WebApi nevű projektben valósítottam meg a kiszolgáló szervert, ami az ASP.NET Core könyvtárt használja. Ebben a rétegben találhatóak a REST API-hoz tartozó Controller osztályok és a SignalR-hez szükséges Hub osztályok.

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 Controller-ekben meg lehet adni, hogy mely végponthíváshoz kell authentikálnia magát a felhasználónak a hívása előtt, az elérési utat, és a paramétereket, amiket vár.

A Hubs mappában találhatjuk azoknak a funkcióknak a kiszolgálóit, amelyeknél valós idejű kommunikálást szeretnénk elérni. Ezeket a végpontokat a Controllerek-hez hasonlóan tudjuk felkonfigurálni, és eseménykezelőket is tudunk definiálni a Hub különböző eseményeihez, például a csatlakozáshoz és a lecsatlakozáshoz.

Az alkalmazás periodikusan elvégzendő feladatait a Jobs nevű mappában találhatóak meg. Ezeknek a feladatoknak az implementálásához a Quartz nevű könyvtár segítségét használtam, amivel könnyedén tudunk függvényhívásokat ütemtervezni.

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. A feladatok ütemtervét is itt kell felkonfigurálni, meg tudjuk adni, hogy mikor és milyen időközönként történjen a végrehajtása.

### Adatelérési réteg

Az adatelérési rétegben az adatbázissal való kommunikációt valósítom meg Entity Framework Core segítségével. A réteg több Repository osztályokból 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. Az alkalmazás jelenleg Microsoft SQL adatbázismotort használ, de az Entity Framework Core több, más cégek által létrehozott adatbázis-kezelő rendszert támogat, így könnyedén le tudjuk cserélni.

## 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.



Ábra : Az adatbázis sémája

### 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.

### 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, kiemelt-e, és a róla készült kép Base64 formátumban.

### Messages

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

### SubscriptionTiers

Ez a táblázat tartalmazza az összes olyan előfizetési csomagot, amire elő tud fizetni egy felhasználó. Egy sor megmondja az előfizetésről, hogy mi a neve, az ára, és azt, hogy hány hirdetést tud kiemelni.

### Subscriptions

Az összes felhasználóhoz kötött előfizetést tartalmazza ez a táblázat. Tartalmazza az előfizetéshez tartozó felhasználót, az előfizetési csomagot, amire előfizetett, és azt, hogy mikor fog lejárni. A felhasználók és az előfizetések között egy-egy kapcsolat van, ha a felhasználó megpróbál többször előfizetni, akkor hiba keletkezik.

## 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; a context-eknek és a hook-oknak; és az adott oldal struktúráját leíró fájloknak.

A valós idejű kommunikáláshoz SignalR könyvtárt használtam. Mivel a Hub-hoz tartozó kapcsolatot több komponensben is kell használni, a hosszú, több komponenseken keresztül áthidaló paraméterláncolás (prop drilling) helyett Context-et használok. Context definiálásával a szülő komponens könnyedén meg tudja osztani objektumokat a fában alatta lévő komponenseknek.

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 újra felhasználni őket, a components mappában pedig könnyen újrahasználható komponensek helyezkednek el.

A képen szöveg, képernyőkép, menü, Betűtípus látható

Automatikusan generált leírás

Ábra : 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 hozzáköthető típusdefiníciókat tartalmazza.

# Megvalósítás

# Irodalomjegyzék

|  |  |
| --- | --- |
| [1] | Existek, „Top Front-End Frameworks in 2023,” [Online]. Available: https://existek.com/blog/top-front-end-frameworks-2021/. [Hozzáférés dátuma: 27 10 2023]. |
| [2] | 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.]. |
| [3] | 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.]. |
| [4] | 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.]. |
| [5] | 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.]. |
| [6] | 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.]. |
| [7] | Microsoft, „Entity Framework Core,” Microsoft, 25. 05. 2021.. [Online]. Available: https://learn.microsoft.com/en-us/ef/core/. [Hozzáférés dátuma: 14. 05. 2023.]. |
| [8] | P. Fletcher, „https://learn.microsoft.com/en-us/aspnet/signalr/overview/getting-started/introduction-to-signalr,” Microsoft, 10 09 2020. [Online]. Available: https://learn.microsoft.com/en-us/aspnet/signalr/overview/getting-started/introduction-to-signalr. [Hozzáférés dátuma: 23 10 2023]. |
| [9] | M. Lahma, „Quartz.NET,” [Online]. Available: https://www.quartz-scheduler.net/. |
| [10] | Microsoft, „TypeScript,” Microsoft, [Online]. Available: https://www.typescriptlang.org/. [Hozzáférés dátuma: 15. 05. 2023.]. |
| [11] | Meta, „React,” Meta, [Online]. Available: https://react.dev/. [Hozzáférés dátuma: 15. 05. 2023.]. |
| [12] | „Domain-driven design,” [Online]. Available: https://en.wikipedia.org/wiki/Domain-driven\_design. [Hozzáférés dátuma: 28 10 2023]. |