ПРОФЕСИОНАЛНА ГИМНАЗИЯ ПО МЕХАНОТЕХНИКА, ЕЛЕКТРОНИКА, ТЕЛЕКОМУНИКАЦИИ И ТРАНСПОРТ „ХРИСТО БОТЕВ“ – ШУМЕН

ДИПЛОМЕН ПРОЕКТ

Тема: „Разработка на WEB сайт на фирмата BulBroker“

Разработил: Ивайло Ивайлов Тончев ……

(име, презиме, фамилия) *(подпис)*

професия код 481030 „Приложен програмист“

специалност код 4810301 „Приложно програмиране“

Ръководител-консултант: инж. Николай Христов ……

(име, фамилия) *(подпис)*

2022/2023 г.

С Ъ Д Ъ Р Ж А Н И Е

I. Предназначение на приложението и проблеми, които то решава.

1.Предназначение:

Приложението има за цел да предостави възможност на фирма занимаваща се с брокерство да представи на клиентите актуалните имоти както и информация за тях.

2.Проблеми, които приложението решава:

Софтуерната разработка предоставя възможност на клиента (възложител), да публикува съдържание с цел реклама на продукта недвижим имот, като това е в допълнение и/или към плакати, билборди , листовки и други средства за реклама. По този начин се постига повишение в посещаемостта на фирмата, а също и се спестяват разходи.

II. Функционалности и правата на достъп които приложението предлага:

1. Нива на достъп и правомощия в приложението:

1.1. Гост GUEST – това са всички потребители, които достъпват приложението без да имат регистрация в него. Тяхната анонимност е гарантирана. Има права да разглежда наличните имоти както и детайлите в тях. Може да вижда секцията контакти.

1.2. Потребител USER – представлява регистриран потребител, притежава всички права на гост потребител като има възможност да запази лични данни (имейл и парола) като и да ги променя в последствие.

1.3. Администратор ADMIN – Регистриран потребител със специфична роля. Притежава всички права на User, но има правото да добавя, редактира и изтрива и в навигационната лента вижда повече секции от всички други роли свързани с добавяне,детайли на адресите,изтриванеи тн.

III. Използвани технологии при разработката.

1. Обща информация за приложението:

Приложението е разработено посредством Model View

Controller(MVC Framework) на .NET 6. Представлява MPG (MultiPage Application) като характерно за него е че всеки адрес връща

нов HTML+CSS менажиран от сървъра.

2.Програмни продукти използвани при разработката на приложението:

- **MS Word** – За документацията към проекта.

MS Word е текстообработваща програма, разработена от Microsoft. Това е един от най-известните софтуерни продукти на света за създаване и редактиране на документи, като може да се използва за различни цели, като писане на писма, доклади, резюмета, статии и други.

С MS Word можете да форматирате текста, като добавяте заглавия, списъци, таблица и изображения. Можете да използвате различни шрифтове, размери на шрифта и цветове. Програмата съдържа и множество функции за корекция на правописни и граматически грешки.

Освен това, MS Word ви позволява да работите в екип, като можете да споделяте документите си с други хора и да работите върху тях едновременно. Програмата позволява и създаването на документи с различни формати като .docx, .doc, .pdf и други.

MS Word е една от най-използваните и популярни програми за текстообработка в света и е част от много офис пакети на Microsoft, включително Microsoft Office.

- **Visual Studio 2022**

Visual Studio 2022 е интегрирана среда за разработка на софтуер, която може да се използва за създаване на различни видове приложения, включително десктоп, уеб, мобилни и игри.

Програмата е разработена от Microsoft и се отличава с мощен набор от функционалности, които позволяват на програмистите да създават качествен софтуер по-бързо и по-ефективно. Visual Studio 2022 включва множество инструменти за управление на проекти, навигация в кода, дебъгване, тестване и много други.

Сред основните характеристики на Visual Studio 2022 са подобреният

редактор на код, който включва поддръжка на интелигентно попълване на кода, автоматично форматиране, проверка на грешки и други. Програмата също така включва инструменти за работа с бази данни, поддръжка на различни езици за програмиране и интегрирани системи за управление на версиите.

Visual Studio 2022 е налична в няколко издания, включително Community, Professional и Enterprise, като всяко от тях предлага различни функционалности и възможности в зависимост от нуждите на програмиста.

- Microsoft SQL Management Studio

Microsoft SQL Management Studio (съкращено SSMS) е интегрирана среда за разработка на бази от данни, която е създадена за управление на Microsoft SQL Server. Средата предоставя множество инструменти за създаване, модифициране и управление на бази данни, сървъри, таблици, процедури, функции и много други.SSMS позволява на потребителите да създават и изпълняват транзакционни заявки, да преглеждат и редактират съдържанието на базата данни, да създават резервни копия и да възстановят бази данни, както и да мониторират и анализират производителността на базата данни.SSMS е инструмент, който е насочен към програмисти, администратори на бази данни и други професионалисти в областта на базите от данни, които работят с Microsoft SQL Server.

- Git-Git е система за управление на версиите, която се използва широко в софтуерната индустрия за съхраняване на кодова база и проследяване на промените в нея. Това позволява на разработчиците да работят ефективно в екип и да поддържат ясност и контрол върху кода си.

Git позволява на потребителите да създават клонове (copies) на кодовата база и да работят върху тях отделно, преди да ги сливат (merge) обратно в основния кодов репозиторий. Това дава възможност на екипите да работят по различни функционалности и да се уверят, че техните промени не ще се конфликтуват.

Git позволява също така да се създават версии на кода, като се запазва историята на всички промени, които са направени върху него. Това прави възможно връщането на кода до предишна версия, ако е необходимо.

Всички тези функции на Git го правят мощен инструмент за сътрудничество и управление на кода.

- **TurtoiseGit**-TortoiseGit е безплатен софтуер за управление на версиите на изходния код, който може да бъде използван в средата на операционната система Windows. Той предоставя интуитивен и лесен за използване потребителски интерфейс, който позволява на потребителите да използват Git за управление на своите проекти.

TortoiseGit е интегрирано приложение за Windows Explorer, което позволява на потребителите да използват Git команди чрез контекстното меню на Windows. Това прави използването на Git по-достъпно и удобно за потребителите на Windows.С помощта на TortoiseGit, потребителите могат да извършват основни Git операции като клониране на репозитории, създаване на нови ветки, комитване на промени и теглене на промени от други потребители.

TortoiseGit предоставя и инструменти за сравнение и сливане на промени, които могат да помогнат на потребителите да решат конфликти между различни версии на своите проекти.

В допълнение към това, TortoiseGit предоставя и графичен интерфейс за преглед на историята на промените в Git репозитория, който може да помогне на потребителите да проследят различните версии на техните проекти.TortoiseGit е мощен и лесен за използване инструмент за управление на версиите на изходния код, който е особено полезен за разработчици на Windows платформа.

3. Програмни езици използвани при разработката на приложението:

- C# версия .NET 6 – За сървърна логика (back-end)

C# е обектно-ориентиран програмен език, който се използва за разработка на софтуерни приложения за множество платформи, включително уеб,

десктоп, мобилни и IoT. Езикът е създаден от Microsoft и е част от .NET платформата.

C# е силно типизиран език, което означава, че всички променливи и обекти трябва да бъдат декларирани с определен тип. Това гарантира по-голяма сигурност и надеждност на програмата.C# има синтаксис, който е подобен на този на други езици от семейството на C, като C++ и Java. Той използва фигурни скоби за маркиране на блокове от код и точки и запетаи за разделяне на инструкциите.Един от най-важните аспекти на C# е неговата поддръжка на обектно-ориентирано програмиране. Това означава, че програмистите могат да създават класове, обекти и наследяване, което позволява лесна и структурирана организация на кода.

C# включва множество вградени типове данни, като целочислени, десетични, символни и булеви типове, както и масиви, списъци и речници. Езикът също така има богата библиотека от класове и методи, които предоставят множество функционалности, включително работа с мрежи, файлове, бази данни, графика и много други.

.NET 6 е последната версия на .NET платформата, която включва множество подобрения и нови функционалности, свързани със сигурността, производителността и мащабируемостта на приложенията. Основната цел на .NET 6 е да предостави единно решение за разработка на приложения, което работи на множество платформи, включително Windows, Linux и macOS.

- **MS SQL** – За бази данни, като автоматично генериран код

през библиотеката Entity Framework.

Microsoft SQL Server е система за управление на релационни бази от данни, която използва структурния заявителен език (SQL) за манипулиране на данните в базата. SQL е език за заявки, който се използва за създаване на бази данни, манипулиране на данни и извличане на информация от базата данни.

При програмирането на SQL се използват заявки за взаимодействие с базата данни, като могат да се използват за създаване на таблица,

добавяне на данни в таблица, извличане на данни от таблица и много други.SQL Server поддържа и разширени процедури за съхранение на данни (stored procedures), които са програми, съхранени в базата данни и изпълнявани на страна на сървъра. Stored procedures могат да се използват за извършване на сложни операции с данни, като например извличане на информация от няколко таблица едновременно, обработка на данни и обновяване на данни.

SQL Server поддържа и транзакции, които са групи от заявки, които трябва да се изпълнят като едно цяло. Транзакциите гарантират цялостността на данните и предотвратяват проблеми с конкурентния достъп до данните в базата данни.

За програмиране на SQL Server могат да се използват различни програмни езици, като например C#, Java, Python и други. SQL Server предоставя множество API-та и драйвери за взаимодействие с базата данни от програмни езици.

- **HTML и CSS** – За визуалната част (front-end).

-**HTML**- HTML е скриптов език за маркиране на уеб страници. Това означава, че HTML използва етикети за да определи структурата и съдържанието на уеб страници. В HTML, етикетите се пишат вътре във файловете на уеб страниците и могат да бъдат използвани за да определят заглавията, параграфите, списъците, линковете, снимките, видеото и много други елементи на уеб страници.

HTML е стандартизиран от W3C (World Wide Web Consortium) и е много лесен за научаване и използване. Той може да бъде използван за създаване на прости или сложни уеб страници и може да бъде комбиниран с други технологии като CSS и JavaScript за да се създават интерактивни и атрактивни уеб приложения.

HTML не е програмен език, защото не позволява да се пишат функции или алгоритми за изпълнение на действия, като програмирането. HTML се използва за да се маркира съдържанието на уеб страници и да се определи начина, по който това съдържание ще се визуализира в браузърите на потребителите.

-CSS- CSS (Cascading Style Sheets) е програмен език, който се използва за определяне на визуалния стил на уеб страници. Той позволява на уеб дизайнерите да определят различни аспекти на изгледа на уеб сайтове, като например цветове, шрифтове, размери, позициониране, отстъпи, граница и т.н.

CSS се използва за разделяне на дизайна от съдържание, като позволява на дизайнерите да се фокусират върху визуалния аспект на уеб сайта, без да се притесняват за HTML структурата. Това прави по-лесно поддръжката на сайта и му позволява да бъде по-гъвкав при промяна на изгледа.

CSS правилата се пишат в блокове, като всеки блок започва със селектор, който определя къде ще се приложат правилата.

- **Java Script** – За логика при визуалната част (front-end)

JavaScript е програмен език, който се използва за създаване на динамични уеб страници и приложения. Той е високо ниво и интерпретируем език, който се изпълнява от клиента, т.е. от браузъра на потребителя.

JavaScript може да се използва за множество задачи, като например манипулиране на HTML елементи, създаване на анимации, валидация на форми, работа със сървърни заявки, изграждане на игри и много други.

Езикът е обектно-ориентиран, като използва обекти и методи за манипулиране на данни и елементи на страницата. Той поддържа също и функционални конструкции, като анонимни функции и затварящи функции.

JavaScript е един от най-широко използваните езици за уеб разработка и се използва както от начинаещи, така и от опитни програмисти.

4. Библиотеки използвани при разработка на приложението и тяхното предназначение.

4.1. Библиотеки при сървърната част (Back-End):

- **Entity Framework** – Entity Framework (EF) е библиотека за работа с бази данни в .NET среда. Тя предоставя ORM (Object-Relational Mapping)

функционалност, която позволява на разработчиците да работят с бази данни като обекти в .NET кода си, вместо да използват SQL заявки.

EF има две основни части - DbContext и DbSet. DbContext е главният клас в EF, който представлява контекста на базата данни. Той управлява връзката с базата данни, предоставя функционалност за заявки и съхранява данни от базата данни в обекти на C#.

DbSet представлява множество от обекти, които съответстват на записите в таблица в базата данни. В EF, DbSet предоставя функционалност за добавяне, изтриване, промяна и търсене на данни от базата данни.

EF поддържа различни модели на бази данни, включително Code First, Database First и Model First. Code First е метод, който позволява на разработчиците да дефинират моделите на базата данни чрез обекти на C# и да генерират базата данни от тях.

В Visual Studio, EF се интегрира като част от ASP.NET и предоставя множество инструменти за разработка на приложения, които използват бази данни. EF може да бъде инсталиран като част от Visual Studio, като пакет от NuGet или като част от .NET Core SDK.

-**Microsoft Identity** – Microsoft Identity е услуга, разработена от Microsoft за управление на идентичността и удостоверяването на потребителите. Това е интегрирана система, която позволява на потребителите да имат единен идентификационен профил, който може да бъде използван за достъп до различни приложения и услуги на Microsoft, като например Microsoft 365, Azure, Dynamics 365 и други. Microsoft Identity осигурява сигурност и защита на данните на потребителите, като използва различни механизми за удостоверяване и защита на идентичността. Това помага на организациите да управляват по-ефективно достъпа до приложенията и да гарантират сигурността на своята информация.

-**LINQ** – Набор от Extension методи върху колекции позволяващ

работа с тях. Използва се в съвкупност с Entity Framework като

работата с колекции класове се превежда от ORM до заявки към

базата на съответният език избран чрез Provider.

LINQ (Language Integrated Query) е технология в програмен език C#, която позволява да

се извършват заявки към източници на данни, като например бази данни, колекции от обекти и XML документи, използвайки език за заявки, вграден в езика C#. Това е мощен инструмент, който позволява на програмистите да извършват заявки и да обработват данни по много удобен и ефективен начин, като се избягват множеството повтарящи се

кодове и грешки. LINQ позволява на програмистите да работят със структури от данни по абстрактен начин , което ги освобождава от необходимостта да се занимават с детайлите на техните имплементации и позволява им да се фокусират върху решаването на проблемите в програмите си.

4.2. Библиотеки при визуалната част (Front-End):

Тези библиотеки се зареждат през CDN връзки (линкове) с цел по-

бързото им сваляне от клиента предвид разположението им на

множество сървъри.

-**Bootstrap** – Съдържа набор от CSS класове дефиниращи правила

за визуализация на html. Цели да позволи писане на HTML без да

се налага писане на CSS а вместо това се присвояват имената на

класовете от библиотеката на различни HTML елементи.

Bootstrap е популярен фреймуърк за уеб дизайн, който позволява на програмистите да

лесно и бързо създават респонсивни уеб сайтове. Респонсивен дизайн означава, че уебсайтът се приспособява към различните размери на екраните на устройствата, като например компютри, таблети и мобилни телефони.

Bootstrap предоставя готови компоненти и стилове за различни елементи на уеб сайтове, като например форми, бутони, менюта, таблиц и др. Това позволява на програмистите да

изградят стандартизиран и красив дизайн за своите уеб сайтове, без да имат нужда да измислят всеки елемент отново.

Bootstrap използва HTML, CSS и JavaScript, за да постигне своя ефект. Той се интегрира лесно с други фреймуърци и библиотеки, като jQuery и AngularJS. Bootstrap е с отворен код, който позволява на програмистите да го променят и да го използват по свой вкус.

-**Font Awesome** – FontAwesome е библиотека от безплатни икони, която може да се използва в уеб дизайна и приложенията. Тя предоставя над 7,000 векторни икони в различни категории като социални медии, технологии, спорт, наука, природа и други.

FontAwesome може да се използва с помощта на CSS класове или SVG код и е лесна за интегриране в HTML документи. Библиотеката съдържа много различни варианти на всяка икона, като размери, цветове и стилове, които могат да бъдат променяни по желание.

FontAwesome е съвместима с различни уеб браузъри и може да се използва с множество уеб технологии, включително HTML, CSS, JavaScript и много други. Тя е полезна за уеб дизайнери и разработчици, които искат да добавят визуални елементи към своите проекти, без да се налага да създават икони от самото начало.

IV. Структура на приложението и организация на кода.

1. Структура на приложението:

Приложението е разработено на база основа ASP.NET MVC като е

разширено и преструктуриране за да позволява скалиране и

менажиране на кода и отстраняване и локализиране на бъгове в

процеса на разработка и експлоатация. Кода в приложението е разделен и обособен във отделни папки и файлове със съответните наименования според тяхната дейност и функционалност.

2. Разделеняне на кода, функции и предназначение на всеки раздел:

2.1. DiplomenProekt представлява Console Application. Ролята му е да

стартира приложението и да управлява и използва всички останали

компоненти в целият Solution. В него се съдържат следните

компоненти:

-**wwwroot**:-"wwwroot" е основната директория в уеб приложенията, която съдържа всички файлове, свързани с уеб страниците и статичното съдържание. Тази папка се използва често в уеб разработката за съхранение на HTML, CSS, JavaScript и други файлове, които се използват за изграждането на уеб страници. В Visual Studio, "wwwroot" е стандартната папка за съхранение на статичните файлове в проекти за уеб разработка.

-**Areas**: Съдържа скафолднати Identity Razor Pages, които са

разширени допълнително за да прилягат на целите на

приложението.

-**Controllers**: Контролерът свързва модела и изгледа. Той приема заявки от потребителя и преобразува тези заявки в действия, които моделът и изгледът разбират. Контролерът се грижи за обработката на заявките, изпълнението на бизнес логиката и връщането на резултатите към потребителя.

-**Views**: Съдържат организирани в папки за да бъдат откривани по

конвенция *.cshtml файлове (Razor views) които се достъпват и

връщат през методите в контролерите.

Изгледът е това, което потребителят вижда и с което взаимодейства. Той използва данните, предоставени от модела, за да генерира HTML, CSS и JavaScript кода, който се показва на потребителя.

-**appsettings.json** – Съдържа информация до която има достъп

Program.cs (традиционно тук се съдържа ConnectionString към

съответната база както и друга чувствителна информация)

-**Program.cs** –Представлява стартируемият клас от който се

стартира приложението в него се съдържа „public static void

Main(string[] args)“ метода. Съдържа конфигурации, как да работи

приложението, в него се регистрират различни класове в

Dependency Injection контейнера, посочва се с кой DbContext да

се работи, какви изисквания има за паролите на потребителите,

дали да се изпозлват Midlewares и Filters.

2.2 Data.Models-в него са енумерациите и всички Entities,

които имат репрезентация в базата с данни.

Моделите в C# и MVC (Model-View-Controller) са ключови компоненти за създаването на софтуерни приложения.

Моделите са класове, които представят данните и логиката на приложението. Те могат да съхраняват данни във вътрешна памет или да изпращат заявки към база данни или уеб услуги за да вземат и обработят информация. Моделите могат също така да изпълняват валидация на данни, да предоставят функционалности за търсене и филтриране на данни и да реагират на събития в приложението.

MVC е архитектурен модел за разработка на софтуер, който разделя приложението на три компонента: модел, изглед (View) и контролер (Controller). Моделът представя данните и логиката, изгледът представя потребителския интерфейс, а контролерът обработва заявките на потребителя и ги изпраща към модела или изгледа.

Когато потребителят прави заявка, контролерът я обработва и избира подходящия модел, който да се използва за да се извлекат или обработят данните. Контролерът след това изпраща тези данни към изгледа, който ги представя на потребителя.

С помощта на този модел, разработчиците могат да разделят приложението на логически компоненти, което прави по-лесно поддръжката и разширяването на приложението в бъдеще.

-**Enumerations**– Съдържа енумерации които се ползват в класовете, на практика в тях се съдържа дефиниция (шифър), как да се интерпретират числовите стойности на отделните колони в базата със стойности от тип енумерация.

В C#, enumerations (или просто enums) представляват тип данни, който позволява да се дефинират именовани константи в рамките на определен набор. Тези константи могат да се използват като стойности на променливи и параметри.

Един енумератор се дефинира с помощта на ключовата дума **enum**, следвана от името на енумератора и набора на именованите константи, разделени със запетая. Всяка константа може да има своя стойност, като по подразбиране първата константа има стойност 0, втората - 1 и т.н.

-**Entities** – Съдържа всички модел класове нужни на приложението, тук са поместени базовите абстрактни класове, а в някои случаи разширяващи функционалност класове например AppUser.

В C# entities са обекти, които представят данни или обекти в една система. Те могат да бъдат дефинирани като класове, структури, интерфейси или енумерации.

Класовете са шаблони за обекти, които съдържат данни и методи за работа с тези данни. Те се дефинират с ключовата дума "class" и могат да наследяват други класове.

Структурите са подобни на класовете, но са по-леки и по-бързи за инстанциране. Те се дефинират с ключовата дума "struct" и не могат да наследяват други структури.

Интерфейсите са дефинирани като списък от методи, които трябва да бъдат реализирани от класовете или структурите, които ги наследяват. Те се дефинират с ключовата дума "interface".

Енумерациите са списъци от имена на стойности, които се използват за представяне на определен набор от константни стойности. Те се дефинират с ключовата дума "enum".

Всички тези типове entities могат да се използват за създаване на обекти и структуриране на данните в една програма на C#.

2.3-Migrations – Директория, в която се съдържат миграциите които

отразяват промените в структурата на базата при началното и

създаване както и в процеса на промяна по време на експлоатация на приложението.Migrations в C# са механизъм за управление на бази от данни в Entity Framework. Те позволяват да създавате и променяте базата данни по програмен път, без да се налага да я променяте ръчно.Когато използвате миграции, вие създавате класове, които наследяват **Migration** класа и представляват конкретни промени в базата данни. Класовете се именуват със символни имена, които описват каква промяна се прави в базата данни.С помощта на миграции можете да добавяте и премахвате таблиците, да променяте структурата на колоните, да добавяте индекси и външни ключове и т.н. Когато имате нови миграции, можете да ги приложите към базата данни с командите **Add-Migration** и **Update-Database** в конзолата на Visual Studio.Migrations предоставят удобен начин за управление на промените в базата данни, особено когато работите в екип, където различни разработчици могат да работят по различни части от базата данни. Те също така позволяват да се отменят или прилагат версии на миграции, когато е нужно да се върнете към предишно състояние на базата данни.

2.4-Data-– Съдържа ApplicationDbContext, наследник на DbContext.

Указва кои таблици ще бъдат създадени на базата на съответните класове от Models, също така указва в метода „protected override void

OnModelCreating(ModelBuilder builder)“чрез fluent API правилата които не могат да се дефинират чрез конвенционално именуване или атрибути.

-**ApplicationDbContext.cs**-Клас съдържащ метод който да

връща конфигуриран ApplicationDbContext (в него трябва да е

зададен ConnectionString за да се осъществи връзка с базата данни.

Класът ApplicationDbContext е част от Entity Framework и се използва за създаване на контекст за взаимодействие с базата данни. Този клас предоставя свойства и методи, които улесняват достъпа до данните, съхранени в базата данни.

2.5-DTO- съдържащ всички Data Transfer

Objects. Целта на DTO обектите е да излагат малка част от данните

намиращи се в базата към визуализационния слой, както и при създаване на нови обекти да транспортират само нужната информация за създаването или редактирането на конкретен обект. Съдържанието на библиотеката е групирано както следва:

-**AddressChoiseDTO.cs**-използва се при всички екшъни на Estate,без delete,тъй като Address си има отделен контролер и се контролира от там.

-**CreateEstateDTO.cs**-използва се за създаването на имотите с техните екстри и информация в допълнение

-**DetailEstateDTO_out.cs**-използва се за детайлите на имотите с техните екстри и информация в допълнение

2.6-Services-в него се намира класът DbSeeder.cs:

-DbSeeder-Този клас съдържа логика проверяваща дали базата е празна или не. Също и методи ,които да популират базата първоначално. Това е важно в процеса на разработка на приложението защото база без записи е трудно да бъде визуализирана на фронтенд от една страна, и от друга

често се налага да се повтарят операциите по популиране на база в процеса на разработка, което ако се случва ръчно би отнело много време и усилия.DBSeeder в C# е библиотека, която се използва за създаване на тестови данни и пълнене на бази данни с тях. Тя позволява на програмистите да дефинират различни модели на данни, които да се генерират автоматично и да бъдат добавени към базата данни. Това е особено полезно при тестване на софтуер, когато е необходимо да се използват реалистични данни, но ръчното им въвеждане е твърде

трудоемко.DBSeeder използва набор от техники за генериране на данни, като например случайно генериране на числа и низове, използване на базови стойности и т.н. Библиотеката може да бъде настроена да генерира данни в различни формати и да запълва различни типове бази данни, включително SQL, NoSQL и др.

2.7-Helpers-в него се съдържа класът GlobalConstants, неговата цел

е да съдържа променливи, които ще улеснят работата във Views,

като например currency= lv или dateFormat = dd-MM-yyyy г.

3. Достъп и зависимости на отделните проекти в Salution, видимост.

3.1. DimplomenProekt – Има право да ползва/вижда:

-Data,DTO,Models,Services

3.2Data– Има право да ползва/вижда: Models,Enumerations

3.3DTO- Има право да ползва/вижда: Models

3.4Models- Не ползва функционалност от други проекти.

3.5Services- Има право да ползва/вижда: Data

4. Съображения при избора на връзки между класовете:

При построяването на връзките “dependencies” се съблюдават две правила.

- Да може да функционира приложението тоест във всяка

библиотека-проект да има достъп до ресурси (класове)

намиращи се в други библиотеки-проекти.

- Да се избегне Circular Reference. Не се допуска две библиотеки

да са видими взаимно пряко или чрез посредник, вместо това

трябва достъпът да бъде иерархичен. По тази причина видно от

зависимостите може да се заключи, че на върха на йерархията на

проектите в Solution се намира Application конзолното

приложение от което се стартира цялото приложение.

V. База данни.

-База данни е организирана колекция от данни, които са съхранени и упоредени по определен начин, така че да може да се извличат и манипулират по лесен и ефективен начин. Това може да бъде компютърна

програма или система, която позволява на потребителите да въвеждат, обработват, съхраняват и извличат данни по удобен начин.

В базите данни данните са организирани в таблица, като всяка колона представлява определен тип данни, а всяка редица е запис с конкретни данни. Базите данни могат да се използват за съхраняване на различни типове информация, като например информация за клиенти, продукти, поръчки, фактури, книги и други.

Базите данни имат много приложения в различни области, като например бизнеса, науката, медицината, образованието и др. Те могат да се използват за анализиране на данни, вземане на решения, управление на проекти, автоматизация на бизнес процеси и много други.

Съществуват различни видове бази данни, като релационни, обектно-ориентирани, графови и др. Всяка от тях има своите предимства и недостатъци, които трябва да се вземат под внимание при избора на подходящ тип база данни за конкретно приложение.

1. Избор на база данни:

При разработката на приложението е избрана релационна база данни

приложение за управлението и Microsoft SQL

Management Studio, в което може да се визуализира базата както и да се

пишат Transact SQL заявки към базата.

2. Подход при разработката и внедряване на базата:

2.1. Подходи при създаване на нова база данни:

- DataBase First – При този подход базата се изгражда

посредством скрипт написан и изпълнен на Transact SQL, като

всички правила, Ограничения, процедури и т.н. се дефинират в

SQL. След като базата е готова на всяка таблица от нея трябва да

се създаде отразяващ я клас, който да репрезентира данните от

базата. Създаването на тези класове е възможно да се изпълни и

чрез Scaffold автоматично.

Основно предимство на този подход е, че базата може да се

дефинира по-оптимално и ясно в T-SQL. Могат да се използват

всички “features” на конкретният SQL език вместо само такива

валидни за всички SQL езици.

Недостатък на този подход е че базата дефинирана през T-

SQL е обвързана с конкретният избор на SQL сървър, изисква се

познание в избраният SQL език. По времеемка разработква.

-Code First – При този подход базата се изгражда след и въз основа на класове в C#. За да се създаде базата, се генерират миграции които се прилагат към базата. Всяка такава миграция съдържа автоматично генерирани T-SQL правила. Като могат да се извършват промени по класовете и да се допълват миграциите в последствие.

Основно предимство на този подход е, че ORM кореспондира с избраният SQL сървър, което позволява на програмиста да не използва чист SQL и предоставя възможност с много малко усилия да се смени SQL сървъра например с Postgre или MySQL в последствие.

Недостатъците на този подход са:

Понеже се разчита на Provider клас който да генерира SQL код, се изисква познание как да се укажат правила при самите класове които да доведат до правилна интерпретация на класовете. Пример за това са атрибути, FluentApi правила, или именоване на свойствата на класовете по конвенция.

Друг недостатък е че ако се изискват функционалности от базата каквито само конкретната база предлага, то това не е най удачният вариант защото Provider класовете за всеки тип база поддържат общи интерфейси тоест само обща функционалност.

2.2. Избран подход и обосновка:

За разработката на приложението е предпочетен подхода Code First понеже в процеса на разработка ще се променят класовете и ще се разчита на миграции за да се отразят тези промени, също така Code First предлага възможност да се разработи приложението без използване на T-SQL а фокусът се измества в посока на C# класовете, където може да се възползва програмистът от Intelisense функциите на Visual Studio 2022.

За неконвенционалните правила се разширява методът

protected override void OnModelCreating(ModelBuilder builder)
Тук се задават правила като композитни ключове и др.
Също така на места са използвани атрибутни обозначения в класовете дефиниращи таблиците модели* (Models).

3. Създаване на база данни и реализация в проекта.

Класовете репрезентиращи базата данни се намират в class Data.Models и могат да се виждат в приложението към дипломният проект.

3.1. Идентификатори:

Всеки клас ,който репрезентира конкретна таблица в базата данни, би следвало да има идентификатор (Id), този идентификатор може да бъде от числов или стрингов тип (GUID). Когато се цели по-голяма сигурност на данните и невъзможност да се предвиди идентификатор на отделни записи при зловреден достъп отвън се назначава string тип за идентификатор, а когато това не е нужно се използва int.
Това е така понеже ,когато

*За по нататъчно улеснение ще наричаме класовете съдържащи Entity типове (или репрезентиращи таблици в база данни) – Модели.

потребителят иска да достъпи своите данни или да види данни на конкретен потребител ,ако това е позволено (например в коментари секция ,ако има налична) от GDPR съображения е необходимо да не може да се достъпват данни на произволни потребители чрез for-цикъл например.

Някои от таблиците имат така наречените композитни ключове, вместо идентификатори от тип Int или string. Това се налага в случаите когато имаме т.нар Mapping таблица съдържаща „връзка“ между две таблици или 2 Foreign Key със типове отговарящи на съответната таблица. В този случай ако се цели уникалност на връзката за да се елиминира повторение на конкретната връзка се използва композитен ключ генериран на

базата на двата Foreign ключа от връзката, по този начин при последващ опит да бъде създаден нов запис със същите ключове това не се позволява от базата тъй като вече има наличен запис с такъв Id.

3.2. Обща функционалност във (за) всички модели:

За всеки Model съществува базова функционалност, която е сходна, и по тази причина са създадени два класа -родители, които да предоставят за наследяване общата функционалност на всички модели – деца.

[BaseEntityData] както и [BaseEntity<T>:BaseEntityData].

Класовете са дефинирани, като абстрактни*.

BaseEntityData е родител на класа BaseEntity<T>. По този начин всички класове „са BaseEntityData” но не всички “са BaseEntity”.

*Абстрактните класове не позволяват инстанциране, тяхната цел е да предоставят функционалност на наследниците им, и да позволяват наследяващите ги класове да бъдат представяни като абстрактният тип при дефиниране на данните (например работа с колекции и др.).

- BaseEntityData.

Тук е поместена функционалност абсолютно необходима за всеки модел. Пряко този клас се наследява само от модели репрезентиращи Mapping таблици, понеже идентификаторите в тези класове са композитни ключове генерирани на базата на ForeignKeys. Косвено този клас е наследен от всички останали чрез BaseEntity<T>. Този клас е в основата на всички модели. Функционалността която този клас поддържа състояние и поведение, както следва:

- При инстанцирането на кой да е негов наследник се попълва свойството DateOfCreation от тип DateTime.
- Съдържа се свойство от тип bool IsDeleted което отразява дали

даденият запис е изтрит от базата или не. Това позволява Soft-Deletion* на записи от базата. Освен това при промяна на това свойство сетера е модифициран така, че да запомня датата на изтриване на обекта, като в същото време позволява изтриването или задаването на този запис като изтрит само веднъж и при задаване на обект като изтрит след като той вече е изтрит не се променя датата на изтриване. По тази причина типа данни на DateOfDeletion е nullable* DateTime понеже неизтритите записи нямат дата на изтриване докато типа DateTime не поддържа null. Поддържа се функционалност и възтановяване на вече изтрит запис.

DateOfDeletion както и DateOfCreation се задават автоматично с текущата дата и час според GMT*.

***Soft-Deletion** – При базите данни когато има налични Foreign Key външни ключове, не е лесно даден запис да бъде изтрит ако неговият Id се явява ForeignKey за друга таблица. Например ако имаме Потребител и Коментар на потребителя и искаме да изтрием потребителя, възниква въпроса кой е оставил коментара? Подходите за справяне с този проблем са няколко. Каскадно, Задаване на Foreign Keys като null ако това е възможно или Ignore. Всеки един от подходите по горе, създава множество проблеми, затова решението е софтуерно да се зададе колона в таблиците от тип bool IsDeleted (репрезентирана в базата чрез 0-false и 1-true), като вместо да се изтриват данните просто не се визуализират и борави със данни имащи зададена стойност true.

***nullable** – По подразбиране типовете имат стойност която не винаги е null. Например типа int32 ако не бъде указана стойност по подразбиране е със стойност 0. Както и типа DateTime задава начална дата при инициализиране. C# позволява декларирането на типове които да имат възможност да заемат стойност null умишлено когато това не е тяхното поведение по подразбиране.

GMT – Приложението следва да се качи (deploy) в интернет и не е гарантирано, че то ще се ползва само от потребители в една и съща часова зона, затова е редно да се използва универсална общоприета часова зона, която при нужда да се конвертира към локалното време което се репрезентира различно за всеки отделен потребител.
BaseEntity<T>.
Представлява Generic абстрактен клас в който се указва типа на идентификатора и бива наследен пряко от класовете, които искаме да имат конкретен тип идентификатор. Тоест всички без mapping моделите където се търси уникалност на връзката. Типа е Generic защото се търси гъвкаво указване на типа на идентификатора (string, int) при самото наследяване.
Класът BaseEntityData<T> съдържа свойство Id от тип T като над свойството с ключа съдържа атрибут указващ на EntityFramework, че идентификаторите трябва да бъдат задавани от базата, като поредни а не да се указват от потребителя ръчно (Identity*).
***Identity** - при ключовете означава записите да бъдат създадени без да се указват изрично стойностите на техните ключове (Id) в базата. Съответно базата извършва проверка до кой пореден номер е достигнато при назначаването на последният ключ и присвоява идентификатор с пореден номер. Интересното е че при изтриване на записи и ключове не се обновява броячът и така е възможно да има липсващи Id-та (да не са поредни). Това е нормално, понеже основна задача на идентификаторите е те да бъдат уникални а не поредни.

4. Модели- описание на релации и организация на данните във всяка таблица (модел).

В приложението е използвана библиотеката Microsoft.AspNetCore.Identity. Тя предоставя базова имплементация на класа потребител както и на таблици управляващи неговите правомощия и достъп до приложенията. Тъй като операциите по регистрация, лог-ин автентикиране и роли са тривиална* задача

тази библиотека предоставя възможност да се преизползва това образцово решение и структура като при необходимост тя се разширява (extend)-ва с допълнителна функционалност.
Библиотеката съдържа както дефиниции на класовете IdentityUser и IdentityRole, така и съпътстващите работа с базата класове-Services съответно UserManager, SignInManager и RoleManager.

4.1. AppUser – Представлява клас наследяващ IdentityUser, като му добавя нови свойства и методи имащи отношение към конкретното приложение. От библиотеката интерес представляват (и се използват) класовете AspNetRoles и свързващата таблица AspNetUserRoles обединяваща конкретен User със съответна роля. Библиотеката предлага и други класове, но те нямат отношение към функционалността на приложението.

***Тривиална задача** в програмирането означава задача чието решение е ясно за имплементация но времеемко и може да се спести посредством готов код.

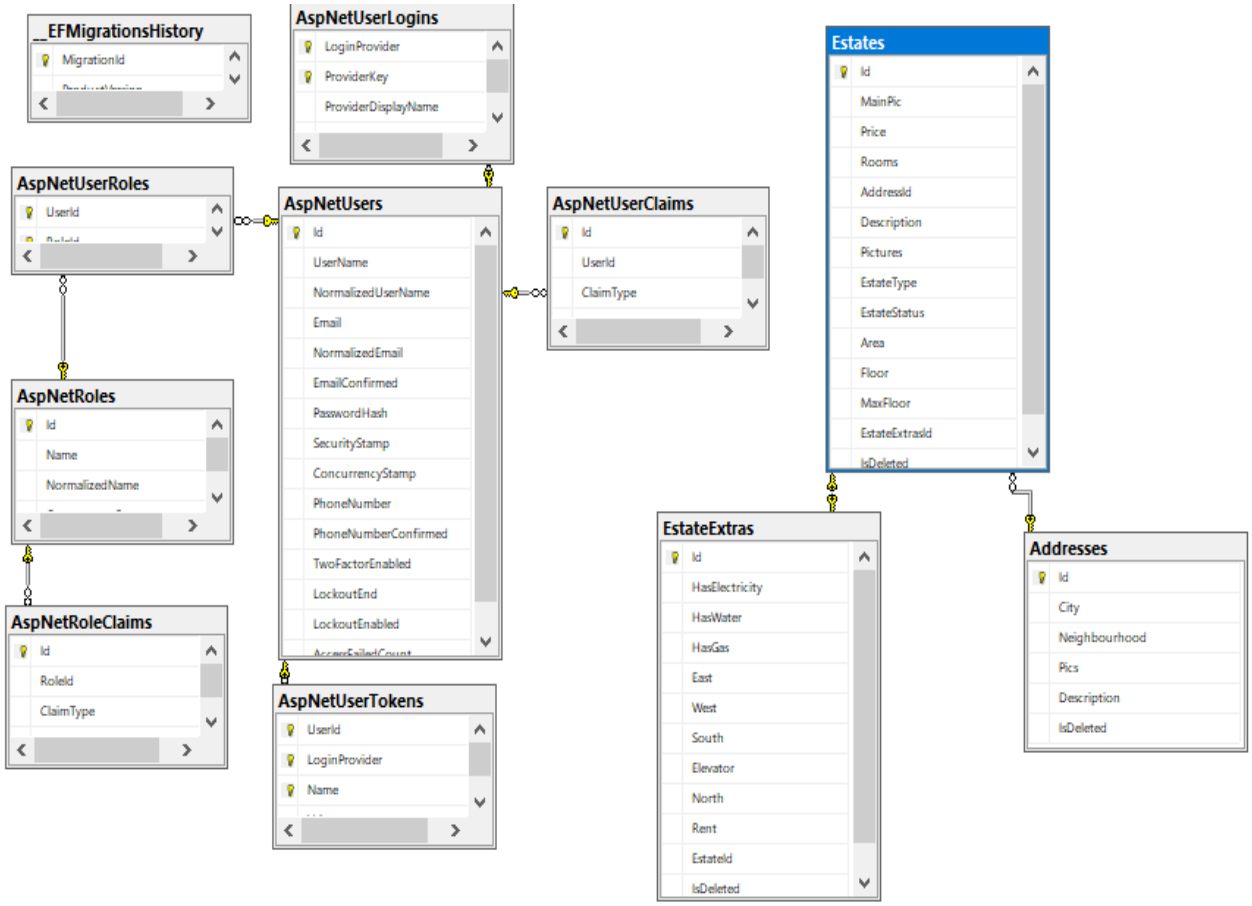
Тривиалната задача в програмирането е обикновено много проста и лесна за решаване. Тя обикновено не изисква специални знания или сложни алгоритми за решаване. Такива задачи се използват често като начални задачи за начинаещи програмисти, за да се запознаят с основите на програмирането.

Един пример за тривиална задача в програмирането може да бъде да се напише програма, която приема две числа и извежда тяхната сума. Това е много проста задача, която може да се реши с едно просто уравнение: сумата на две числа е равна на първото число плюс второто число.

Друг пример може да бъде да се напише програма, която извежда "Hello, World!" на екрана. Това е много стандартна задача, която се използва като начална точка за много програми, за да се провери дали компилаторът работи правилно и за да се убедим, че програмата е написана правилно.

Като цяло, тривиалните задачи в програмирането са много прости и не изискват специални знания или сложни алгоритми. Те са често използвани като начална точка за начинаещи програмисти, за да се запознаят с основите на програмирането и да се научат да пишат прости програми.

-Entity Relationship Diagram – Диаграма на връзките в базата данни.



Диаграмата на връзките в базата данни (Entity Relationship Diagram, съкратено ERD) е графичен инструмент, който се използва за представяне на връзките между съставните части на една база данни. Тя показва как същностите (entities) са свързани помежду си и как тези връзки са структурирани.

Същностите са обекти, които съдържат данни и информация в базата данни, като например клиенти, продукти, поръчки и други. Връзките между тях са представени чрез линии и символи.

Символите, използвани в диаграмата, могат да представят различни типове връзки, като например едно-към-много (one-to-many), много-към-много (many-to-many) и други.

ERD е полезен инструмент за проектиране на бази данни, тъй като показва как същностите са свързани помежду си, какъв е броят на връзките между тях и как те са структурирани. Това позволява на проектантите да проектират базата данни по начин, който да отговаря на нуждите на конкретната система или приложение.

4.2 Estates - Съдържа информацията за един имот .Тази таблица е свързана таблиците

за екстрите и адреса на всеки отделен имот.Всеки имот си има много екстри,но един адрес.

4.3 EstateExtras - Съдържа информацията за всички екстри на един имот.Един имот има много

Екстри,които могат да се редактират само за конкретен имот.

4.4 Addresses - Съдържа информация за адреса на един имот.Адресите могат да се редактират

от едно място за всички имоти в отделна страница,която само адмна може да види.

5. Особености и разлики в моделите и таблиците от базата.

5.1. Имена на таблици и класове .

В базата данни таблиците имат имена в множествено число,

защото съдържат записи и всеки ред от таблица е отделен запис със

собствен идентификатор, тоест в базата данни се описват

правилата за подредба на множество данни като всяка таблица е

контейнер за множество записи.

В моделите отделните класове имат имена в единствено число

защото всеки модел описва 1 конкретен тип обекти, тоест

индивидуални особености за всеки запис.

5.2. Разлики при моделите и таблиците в базата:

Някои данни се представят по различен начин в базите данни

и в кода, поради естеството на работа на базите данни.

***Тегло** – При теория на графите, представлява допълнителна

информация към дадено ребро или дъга отразяваща данни, дължина

на трасето, разход за транспорт, време за пътуване и т.н. В

конкретният случай се подразбира информация натоварваща

връзката на два записа от различни таблици с допълнителна

Информация.

-Енумерации: В езика C# енумерацията предсатвлява двойка от

[числова стойност + стрингова стойност], докато в базата това се

репрезентира само, като числова стойност. Следователно смисълът

от всяка числова стойност на енумерацията е дефиниран в

приложението в C# и само то може да разчете, какво означава.

- Представяне на сложни данни като стринг:

Понеже базата поддържа съхранение на текстови данни, за множество от данни, които се отнасят до само 1 обект(запис) може да се избегне създаването на допълнителни таблици .
-Пример за такъв случай са изображенията – Pictures в таблицата Estates и модела Estate(имот). В този конкретен случай имаме колекция от множество препратки към изображения, но те се отнасят само до 1 имот. Следователно не е нужно, да се създава таблица със записи за всяко изображение понеже информацията от тази таблица няма да се ползва от множество филми и това би натоварило базата допълнително релацията би била one – to – one. Вместо това множеството от изображения се съхранява като 1 стринг който се разчита от C# и се превръща в колекция от string, като всеки от тях представлява отеделна връзка към изображение. При нужда от сложно представяне на данни когато данните са индивидуални може да се разчита на JSON* сериализация* и десериализация*.

***JSON – Java Script Object Notation** – Този формат предоставя възможност да се представят данни като колекции и/или типове като стрингове с цел лесен пренос на информацията в WEB. Аналог на този стандарт е XML. Тези формати могат да пренасят информация за състояние но не и за поведение на класовете.

JavaScript Object Notation (JSON) е лек, текстуален формат за обмен на данни. Той е базиран на JavaScript обекти, но може да се използва от много езици за програмиране. JSON е изключително популярен формат за предаване на данни в мрежови приложения, особено във връзка с уеб разработката.

JSON форматът се използва за предаване на структурирани данни между приложения. Той е базиран на два основни типа данни - обекти и масиви. Обектите в JSON представляват колекция от ключове и стойности, където ключовете са низове, а стойностите могат да бъдат както низове, числа, булеви стойности, обекти или масиви. Масивите в JSON съдържат списъци от стойности, които могат да бъдат както низове, числа, булеви стойности, обекти или други масиви.

JSON е лесен за използване и четене от хора, както и от машини. Той е съвместим с много езици за програмиране и много приложения и библиотеки могат да го използват за обмен на данни. JSON форматът се използва широко в уеб разработката, където се използва за комуникация между клиентските и сървърните приложения.

За да използвате JSON, вие можете да го генерирате чрез своя код, или да го прочетете и обработите във вашия код. В повечето програмни езици има готови библиотеки за работа с JSON форматът.

***Сериализация** - Сериализацията е процесът на преобразуване на обект от програмен език във формат, който може да бъде запазен във файл или изпратен по мрежата. В основата си, сериализацията представлява преобразуването на данните на обекта в стрингово репрезентиране, което да може да бъде съхранено или предадено.

В процеса на сериализация, данните на обекта се конвертират в подходящ за съхранение формат, като често това е текстов формат, като например JSON или XML. Тези формати позволяват лесното запазване на данните във файл или тяхното предаване по мрежата. След като данните на обекта са били сериализирани, те могат да бъдат десериализирани обратно в обект, когато това е необходимо.

Сериализацията е полезна техника в програмирането, тъй като позволява на програмата да запазва и прехвърля данните на обектите си между различни приложения и системи. Това може да бъде полезно при работата с бази данни, при комуникацията между клиент и сървър, или при запазване на данни във файлове.

***Десериализация** - Десериализацията е процесът на преобразуване на сериализирани данни обратно в техния оригинален обектен или структурен вид. Сериализацията и десериализацията са важни техники за запазване на данни и обектни структури между различни приложения и устройства.

Когато данните се сериализират, те се конвертират във формат, който може да бъде записан или предаден като текстов низ, като например JSON, XML или бинарен формат. След това, когато данните се

десериализират, те се възстановяват в техния оригинален обектен или структурен вид.

По този начин, десериализацията може да се разглежда като обратен процес на сериализацията, който превръща данните от стрингово репрезентирани формати в обектен вид.

Виртуални свойства:

В класовете Модели има елементи, които нямат

репрезентация в базата данни във вид на колони. Това се налага в следните случаи:

- При нужда от допълнителен конструктор за по-лесно създаване или попълване на данни по подразбиране. Всички модели трябва да притежават безпараметричен конструктор, но неговото поведение или наличието на други конструктори за по-лесно създаване на обекти, не влияе на работата на моделите.
- При нужда от методи: Методите не се репрезентират в базите данни, но могат да предоставят допълнителна функционалност към всеки модел. Това може да се постигне разбира се и чрез разширение във външен клас например в библиотеката Common.
- При сложни обекти. Понеже приложението работи с ORM Ако в даден обект има външен ключ то, EntityFramework би могъл да намери в базата обект от типа към който сочи този ключ и да го предостави за по-лесна работа, в конкретният обект когато се изтегли от базата. Когато тези обекти са много при релация (one to many) може да се разчита на колекция от обекти със външен ключ сочещ към текущият идентификатор на обекта. Важно е да се отбележи, че при колекции те биват инстанцирани в безпараметричният конструктор за да позволяват добавяне на нови обекти в колекцията. Тези сложни типове не са задължителни при използването на моделите, но улесняват работата с моделите значително. Всички сложни типове (обекти

и колекции от обекти-модели) са отбелязани като virtual*.

***Virtual** обозначението позволява промяна на имплементация в наследник на класа на конкретният метод или свойство. Ако се цели използването на Lazy Loading трябва да се отразят методите и типовете като virtual.

***Lazy Loading** – Означава изтегляне на информация от базата при поискване на неизтеглено поле на конкретен модел. Позволява гъвкава работа с базата, но е възможно да доведе до прекалено много непланирани заявки към базата данни.

6. Типове релации между таблиците и моделите:

6.1. One-To-One. При тази релация в модел А има външен ключ към модел В и обратното, в модел В има външен ключ към модел А. Освен това тези външни ключове и в двете таблици са със зададено ограничение Unique. Това би могло да се използва в приложението при Estate-> Pcitures но е избегнато за да не се усложнява базата и затова няма пример за такава релация в настоящото приложение. Виж т.5.2 (представяне на данни като стринг).

6.2. One-To-Many. В C# връзката "one to many" се осъществява с използване на колекции от обекти. Колекцията от обекти на "едно" се съхранява в пропърти на обекта на "много", като се използва подхода на агрегация. Това означава, че обектът на "много" има референция към обекта на "едно", като обектите на "едно" могат да имат нула или повече референции към обекти на "много".

6.3. Many-To-Many. В програмирането на C# връзката "many-to-many" се постига чрез използване на междинна таблица, която свързва две други таблиците. Тази междинна таблица има два външни ключа, като всеки от тях сочи към запис в съответната таблица.

VI. Фронт-енд визуализация и контролери

1. Razor Pages:

Използват се Razor Pages за управлението на потребителските данни като log-in, log-out, register, достъп до данните на потребителя и Други.

Razor Pages е модел за уеб програмиране в ASP.NET Core, който позволява лесно създаване на уеб приложения с минимално количество

код. Този модел е създаден да бъде лесен за използване от начинаещи програмисти, но в същото време е мощен и гъвкав за да се справя и с по-сложните изисквания на професионалните разработчици.

Razor Pages използва синтаксис, който наподобява този на ASP.NET MVC, но се фокусира върху лесното създаване на уеб страници без необходимостта от разделянето на логиката и представянето в различни файлове. Вместо това, Razor Pages позволява лесното комбиниране на логиката и представянето в един файл.

Един от големите ползи на Razor Pages е възможността за лесно създаване на CRUD (Create, Read, Update, Delete) операции за работа с бази данни, като се използва Entity Framework Core или друг подобен ORM фреймуърк.

Като цяло, Razor Pages е много гъвкав и удобен модел за уеб програмиране, който позволява бързо създаване на уеб приложения с минимално количество код.

2. Razor Views:Използват се Razor Views за визуализиране на страниците от приложението със индивидуална бизнес логика. Всички Views се

подпъхват във общо _Layout.cshtml което също представлява View, като

е имплементирана логика по зареждане на индивидуален CSS и JavaScript за всяко View.Razor е маркиран език за шаблони, който се използва в ASP.NET, за да се генерират HTML страници. Той е създаден от Microsoft и предлага прост и лесен за използване синтаксис за генериране на динамичен HTML.Razor е смесица от HTML и C# код, която позволява на разработчиците да създават лесно и ефективно HTML страници, които са динамични и могат да взаимодействат с бази данни, данни на потребителя и други външни източници.За да използвате Razor в ASP.NET, трябва да създадете Razor файлове с разширение .cshtml. В тези файлове можете да използвате Razor синтаксиса за да генерирате HTML страници.

Razor е много популярен сред разработчиците на ASP.NET поради своя лесен за използване синтаксис и възможността да генерира HTML динамично, като използва данни, които са достъпни от приложението.

-Layout.cshtml- layout.cshtml е файл, който определя общия изглед на уеб страниците в ASP.NET MVC приложение. Този файл съдържа HTML, CSS и JavaScript код, който се използва за създаване на общия вид на страниците, като например хедъра, футъра, менютата, табовете, логото и други елементи на дизайна.**layout.cshtml** се използва като шаблон за всички страниците в приложението, които го наследяват. Това означава, че всички HTML елементи, които са дефинирани в **layout.cshtml**, ще бъдат включени във всички страници, които го използват. В **layout.cshtml** могат да се дефинират и специфични за всяка страница секции, като например съдържанието, което се различава за всяка страница в зависимост от изискванията й. Тези секции се наричат **@RenderSection** и са подходящи за динамично създаване на съдържание за страниците.Този файл се намира в папката **Views/Shared** в проекта на ASP.NET MVC приложението и може да бъде променен според нуждите на разработчика и дизайнера.

Достъпът до Views се ограничава в някои случаи само за потребители със роля администратор, това става със сървърна логика на ниво Action или на ниво Controller, освен при FrontEnd, което е за удобство а не с цел сигурност.

3. Статични ресурси.

Статичните ресурси като изображения, CSS и JavaScript са разположени в папката wwwroot, като с цел разграничение на кода ресурсите в тази папка са организирани в под-директории.

VII. Функционалност на приложението която, следва да се имплементира, надграждане.

1. Връзка потребител-администратор. Може да се имплентира чат система, която ще подобри комуникацията и ще улесни потребителите на приложението ,с което ще се подобри мнението на клиентите и за самата фирма .Секцията за чат може да се намира в навигационната лента, като това ще улесни допълнително потребителите,но може и да се показва например само в страницата за контакти.

2. Коментари от потребителите,които се следят от модератор.

Тук филтрация може да се извърши така, че само потребители

регистрирали се ,да имат право да изказват мнения под формата на коментари. Като всяко от мненията може да има съответен брой харесвания и нехаресвания от

останалите потребители. Като отново трябва да се имплементира логика,

кой ще има право да вижда тези коментари. Както и да се следи даден

потребител да не харесва повече от веднъж определен коментар. Също

така тези мнения от имоти могат да бъдат публикувани след удобрение

от модератор, или модератора да следи за лоши мнения. Възможно е да

се имплементира логика по маркиране на коментар като лош и

временното му скриване от останали потребители докато не се потвърди

от модератор.След ,което, ако той бъде потвърден да не позволява повече

репортване от потребители.

3. Превод на всичко в самото приложение на различни езици

Това ще повиши посещаемостта от гледна точка на това,че дори

ако хора от всякакви държави и националности посетят сайта,те ще

могат да го преведат на своя език,съответно да разберат всяка информация

за имотите ,като състояние,цена,местоположение и тн.

4. Анализ на данни - ако приложението работи с много данни, може да се имплементира функционалност за анализ на данни, като например генериране на статистики, диаграми и т.н.

5. Поддръжка на мобилни устройства - ако приложението трябва да се използва на мобилни устройства, може да се имплементира функционалност за поддръжка на мобилни устройства, като например оптимизация на изгледа за мобилни устройства или разработка на мобилно приложение.

6. Интеграция със социални мрежи - ако приложението е свързано със социални мрежи, може да се имплементира функционалност за интеграция с тях, като например споделяне на съдържание в социалните мрежи или вход със социален профил.

7. Изпращане на електронни съобщения - ако приложението трябва да изпраща автоматични електронни съобщения, като например потвърждения за регистрация или напомняния за изтекла регистрация, може да се имплементира функционалност за изпращане на електронни съобщения.

VIII. Използвана литература.

1. Боровска Пл., Компютърни системи, изд.Сиела, 2005.

2. Боянов Л., К. Боянов и др., Компютърни мрежи и телекомуникации,

изд. “Авангард Прима”, София, 2014.

3. Васил Георгиев. Съвременните технологии за конкурентна обработка.

Издателство „Св. Кл. Охридски”, 2013.

4. Владимир Димитров, Васил Георгиев. Съвременните модели за

информационно-технологично обслужване.

Издателство „Св. Кл. Охридски”, 2012.

5. Илиева С., В. Лилов, И. Манова, Изграждане на софтуерни

приложения, 2006, издателство СУ “Кл. Охридски” .

6. Илиева С., В. Лилов, И. Манова, Методи и подходи за разработване на

софтуерни системи, 2010, издателство СУ “Кл. Охридски”.