Технически университет - София

Филиал Пловдив

Дипломна работа

Тема: Web-базирана информационна система за отдаване на помещения под наем

Студент: Деница Павлова Христева

Фак №: 367400

Специалност: КСТ

Образователна степен: ОКС бакалавър

Факултет: ФЕА

Ръководител: гл. ас. д-р Добринка Петрова

ТУ – София, Филиал Пловдив, 2020 г.

Съдържание

[Увод 4](#_Toc44915390)

[ГЛАВА 1. Обзор – състояние на проблема по литературни данни 6](#_Toc44915391)

[1.1 Анализ на особености и нужди за фирми и лица, занимаващи се с отдаване на обекти под наем 6](#_Toc44915392)

[1.2. Изисквания към Web-системата 6](#_Toc44915393)

[1.2.1 Нефункционални изисквания. 6](#_Toc44915394)

[1.2.2. Функционални изисквания. 7](#_Toc44915395)

[1.3. Структура на постъпващата информация. 9](#_Toc44915396)

[ГЛАВА 2. Описание на структурата на базата данни 9](#_Toc44915397)

[ГЛАВА 3. Избор на програмни езици, програмна среда, сървър за бази данни и архитектурен модел 14](#_Toc44915398)

[3.1 Програмни езици 14](#_Toc44915399)

[3.2 Програмна среда. 16](#_Toc44915400)

[3.3 Система от база данни 18](#_Toc44915401)

[3.4. Архитектурен модел. 18](#_Toc44915402)

[ГЛАВА 4. Описание на компонентите на Web-системата и тяхното взаимодействие 22](#_Toc44915403)

[4.1. База данни. 22](#_Toc44915404)

[4.1.1 Класове - модели в приложението (Models) 22](#_Toc44915405)

[4.1.2. Достъп до данните от базата 23](#_Toc44915406)

[4.2. Модели за изглед (View Models) 26](#_Toc44915407)

[4.2.1. Библиотека AutoMapper 27](#_Toc44915408)

[4.2.2. Валидиране на данни чрез анотации (Data Annotation Validations) 28](#_Toc44915409)

[4.3. Изгледи (Views) 28](#_Toc44915410)

[4.4. Контролери (Controllers) 37](#_Toc44915412)

[ГЛАВА 5. Приложимост. Изводи. 37](#_Toc44915413)

## Увод

Задача на настоящата дипломна работа е създаване на Web-базирана информационна система за отдаване на помещения под наем. В процеса на осъществяване на проекта са предприети действия, осигуряващи надеждни теоретични и софтуерни решения. Направени са проучвания за определяне както степента на приложимост, така и за целевата аудитория на бъдещата Web-система. Възоснова на тях е изградена представа за подходяща организация на базата данни и Web-приложението, идеи за потребителския интерфейс, визуализиране на графичните ресурси и статичната информация.

Избора на използваните технологии се базира на необходимостта от устойчива програмна среда за справяне със системните изисквания, характеристиките и планираните функционалности на Web-приложението. То ще е от тип клиент-сървър и като такова предвижда, наред с изграждането на модули с различни функции и логически издържано взаимодействие между тях, да се избере и подходящ дизайн, който не утежнява хода на действие на потребителя.

Цели се системата да отговаря на условията за бързодействие, увеличен капацитет, повишена надеждност, лесно откриваема информация и конкурентноспособност.

Приложението следва да поддържа функционалоност за регистриране на потребители, които ще имат опция както да добавят свое място за отдаване под наем, така и да наемат от вече наличните в базата данни на системата. Ще бъде изградена рейтингова система за поставяне коментари и оценки само от регистрирани потребители, които ще сформират обща оценка за дадено място. Приложението ще разполага и с контролна част. Тя ще разрешава на оторизиран потребител - администратор - да прави промени в съдържанието.

Съставена е база данни, която да съхранява информацията от споменатите дейности, а именно - данни за потребители, данни за обекти и техните характеристики, данни за резервации. За целта е използван Miscrosoft SQL Server. Програмната среда за разработване на Web-приложението е Microsoft Visual Studio 2019, .NET Core.

# ГЛАВА 1. Обзор – състояние на проблема по литературни данни

## 1.1 Анализ на особености и нужди за фирми и лица, занимаващи се с отдаване на помещения под наем

Tехнологиите са неизменна част от ежедневието на съвременния човек. Все повече обществени сфери се дигитализат – информацията и услугите стават все по-достъпни в Интернет пространството. Вследствие на това, дигиталната грамотност се превръща в жизнено важно условие за участие в съвременната икономика и общество. Веднъж създадена, информацията в Интернет, може да бъде посетена от всеки, който има интерес към нея. Сега съществуват електронни пазари, благодарение на които хората могат да закупуят определена стока или услуга от дома си, като това им дава възможност да спестят време и пари. В това число иновациите в технологиите довеждат до софтуерни решения за оптимизиране на процесите по наемане на обекти. Рационализират се бизнес операциите, автоматизират се определени задачи, премахва се ръчната работа за въвеждане и съхранение на данни, съответно се намалява или дори изключва риска от допускане на грешки, дублиране на информация и др.

Българските наемодатели все повече осъзнават потенциала на дигиталните технологии в подкрепа на техния бизнес и предприемат стъпки за използването им. Най-силно застъпените технологии към момента са мобилните и Web-приложенията.

## 1.2. Изисквания към Web-системата

### 1.2.1 Нефункционални изисквания.

**Безопасност на системата**

Системата трябва да отговаря на изисквания за сигурност на данните. Съобразено с това, че всички данни, които Web-приложението използва, се намират на сървърен компютър, безопасността във физическия смисъл е свързана с надеждността на самия сървър. Това включва предпазване от токов удар, сигурност и стабилност на базата данни.

**Системна производителност**

За да бъде пълноценна системата, всички страници, включително и тези, които се генерират при използване на търсачката в приложението, не трябва да отнемат повече от 5 секунди, за да се заредят. Това се налага, тъй като при по-бавното зареждане на страниците, е възможно да се появи загуба на интерес от страна на посетителя и той да напусне приложението неудовлетворен.

**Поддръжка на системата**

Системата трябва да е лесна за поддръжка – да еразработена така, че да се модифицира лесно с цел добавяне или изваждане на функции, подобряване на бързодействието и отстраняване на грешки.

### 1.2.2. Функционални изисквания.

За да бъде функционално, полезно и лесно за работа, Web-приложението трябва да притежава някои основни компоненти:

Клиентска част – това е частта, която се показва, когато потребител зареди URL-адреса на сайта в своя браузър. Началната страница трябва да бъде оформена по лесен за употреба от потребителя начин и той да добие ясна представа за целта и значението на изграждане на подобно Web-приложение.

Вход в системата - ще се допуска след извършване на регистрация на потребител, която се осъществява с email и парола, а email-a ще се подразбира и като username. Настройките на потребителския профил включват добавяне на мобилен телефон, смяна на email-а с друг, промяна на паролата и настройване на двуфакторно удостоверяване (two-factor authentication). Вход в контролната част на сайта ще се осъществява само от определена страница, достъпът до която е възможен само след правилно въведени email и парола на администраторския профил.

Добавянето на помещение за отдаване под наем трябва да се случва след попълване на форма за данни за мястото като: характеристики, местоположение, снимки, екстри, условия и цена. Всички съществуващи обекти в системата ще са достъпни за преглед от всеки посетител на приложението, но опцията за резервация ще изисква регистрация и вход.

При желание за наемане на даден обект, след избор на дати и потвърждение с бутoн за финализиране на дейността, ще се появява съобщение относно извършения процес и потребителят ще се пренасочва към списък с всички негови резервации.

Търсенето също е полезна функционалност в тази система. Ще се предостави възможност да се намерят обекти спрямо зададена локация, брой наематели и предпочитани дати за наемане. След избор на критериите и натискане на бутона за търсене, системата трябва да извежда всички резултати, които отговарят на аргументите. Друга опция ще предлага списък от всички помещения в дадена категория, като информацията за тях ще се поднася в съкратен вид, което допринася за бързо ориентиране на посетителя.

Всеки обект в системата ще притежава и Web-страница с подробна информация за характеристиките му, данни за собственика и календар, посочващ свободните дати за резервиране. В нея ще се предостави възможността за поставяне на коментари и оценки, които ще са достъпни за преглед.

## 1.3. Структура на постъпващата информация.

Релационната база данни ще съхранява информация за:

* регистрирани потребители в системата
* категории за обекти под наем
* характеристики за обект
* снимки за обект
* коментари и оценки за обект
* региони и градове в България
* резервации

# ГЛАВА 2. Описание на структурата на базата данни

Таблиците и връзките между тях са визуализирани на диаграмата по- долу (фиг.1), след което следват техните описания. Ще бъде използван следния формат на описание:

* (PK) – първичен ключ (Primary Key)
* (FK, Таблица) – външен ключ, релативна таблица (Foreign Key)



фиг. 1 - Схема на базата данни

Таблица **AspNetUsers** – съдържа информация за потребителите:

* Id(PK) – идентификационен номер
* UserName – потребителско име
* Email – имейл
* EmailConfirmed – дали е потвърден email-а
* PasswordHash – хеш на паролата
* PhoneNumber – телефонен номер
* PhoneNumberConfirmed – дали е потвърден тел. номер
* TwoFactorEnabled – дали е настроено двуфакторно удостоверяване
* IsAdmin – роля на потребителя

Таблица **Places** – съдържа информация за обектите:

* Id (PK) – идентификационен номер
* UserId (FK, AspNetUsers) – id на собственик
* CityId (FK, Cities) – id на град
* CategoryId (FK, Categories) – id на категория
* Аddress – адрес
* Description – описание
* PriceByNight – цена за нощувка
* MaxGuest – максимален брой гости
* BathroomsNum – брой бани
* Pets – дали са позволени домашните любимци
* Smoking – дали е позволено е пушенето
* BedroomsNum – брой спални стаи
* BedsNum – брой легла
* Rating - оценка

Таблица **Images** - съдържа информация за снимките:

* Id (PK) – идентификационен номер
* PlaceId (FK, Places) – id на място, към което принадлежи
* Name – име на снимка
* Path – път към снимка
* Ext – разширение на файла

Таблица **Еxtras** – съдържа информация за екстрите:

* Id (PK) – идентификационен номер
* Name – име на екстрата

Таблица **PlaceExtras** – съдържа информация за екстрите на един обект:

* Id (PK) – идентификационен номер
* PlaceId (FK, Places) – id на обектът, към който принадлежи екстрата
* ExtraId (FK, Extras) – id на екстра

Таблица **Reservations** – съдържа информация за резервациите:

* Id(PK) – идентификационен номер
* PlaceId (FK, Places) – id на мястото, за който е резервацията
* UserId (FK, AspNetUsers) – id на наемателя
* StartDate – начална дата
* EndDate – крайна дата
* PricePerNight – цена за нощувка
* TotalPrice – обща сума
* NumNights - продължителност в дни
* Reviewed – получена ли е обратна връзка
* Active – дали е активна

Таблица **Reviews** – съдържа информация за обратната връзка:

* Id(PK) – идентификационен номер
* PlaceId (FK, Places) – id на мястото, за което се отнася
* UserId (FK, AspNetUsers) – id на потребителя, който е изпратил обратната връзка
* Rating – оценка
* Comment – коментар

Таблица **Categories** - съдържа информация за категории на места за отдаване под наем:

* Id(PK) – идентификационен номер
* Name – име на категория
* ImageUrl – път към снимка за категория
* ImageName – име на снимка за категория

Таблица **Cities** - съдържа информация за градове:

* Id(PK) – идентификационен номер
* Name – име на град
* RegionId (FK, Regions) - id на областта, в която се намира градът

Таблица **Regions** - съдържа информация за области:

* Id(PK) – идентификационен номер
* Name – име на област

Всички таблици съдържат полета:

* CreatedOn – дата на създаване
* ModifiedOn – дата на промяна
* IsDeleted – информация за изтриване
* DeletedOn - дата на създаване

# ГЛАВА 3. Избор на програмни езици, програмна среда, сървър за бази данни и архитектурен модел

## 3.1 Програмни езици

**C#** е обектно-ориентиран език за програмиране, разработен от Microsoft, като част от софтуерната платформа .NET.  Програмите на C# представляват един или няколко файла с разширение .cs., в които се съдържат дефиниции на класове и други типове. Тези файлове се компилират от компилатора на C# (csc) до изпълним код и в резултат се получават асемблита – файлове със същото име, но с различно разширение (.exe или .dll).

Едно от най-големите предимства на .NET Framework е вграденото автоматично управление на паметта. То освобождава програмистите от сложната задача сами да заделят памет за обектите и да търсят подходящия момент за нейното освобождаване. Това сериозно повишава производителността на програмистите и увеличава качеството на програмите, писани на C#.За управлението на паметта в .NET Framework се грижи специален компонент от CLR, наречен „система за почистване на паметта“ (garbage collector). Основната задача на системата Garbage collector-а е да следи кога заделената памет за променливи и обекти вече не се използва, да я освобождава и да я прави достъпна за последващи заделяния на нови обекти.

Предимства на C#:

* C# е създаден като прост, модерен с общо предназначение и обектно-ориентиран език за програмиране.
* Езикът е предназначен за използване в развиващите се софтуерни компоненти, той е подходящ и за разполагане в разпределена среда.
* На езика C# и върху .NET платформата може да бъде разработван разнообразен софтуер, като офис приложения, уеб приложения, уеб сайтове, настолни приложения, мултимедийни Интернет приложения, приложения за мобилни телефони, различни видове игри и много други.

**JavaScript** е програмен език, който позволява динамична промяна на поведението на браузъра в рамките на дадена HTML страницата. JavaScript се зарежда, интерпретира и изпълнява от Web браузъра, който му осигурява достъп до Обектния модел на браузъра. JavaScript функции могат да се свържат със събития на страницата (например: движение/натискане на мишката, клавиатурата или елемент от страницата, и други потребителски действия). Прието е JavaScript програмите да се наричат скриптове.

JavaScript е един от най-използваните скриптови езици за програмиране в Интернет. Toзи език осъществява моментното и динамично взаимодействие между потребител и браузър. Благодарение на JavaScript Web страниците, които използваме в момента, изпълняват много повече функции от това просто да зареждат данни. JavaScript позволява създаването на интерактивни и адаптивни интерфейси с много добър дизайн и динамични функционалности, изцяло с насоченост към крайния потребител.

**HTML** е основният маркиращ език за описание и дизайн на уеб страници. Той осигурява множество от специални елементи, които описват как трябва да изглежда една уеб страница.

**Cascading Style Sheets(CSS)** - определя изгледа и оформлението на текста и други материали. HTML елементите са основната градивна единица на уеб страниците. Чрез тях се оформят отделните части от текста на една уеб страница като заглавия, цитати, раздели, хипертекстови препратки и т.н.

**Bootstrap** е client-side среда с отворен код, която съдържа набор от инструменти за създаване на Web приложения и Webсайтове. В Bootstrap са включени HTML и CSS дизайн шаблони за типография, форми бутони, навигация и други компоненти за интерфейса на Web-приложението и/ли Web-страниците. Има няколко вградени файлове, които не е препоръчително да се променят. Тези файлове са с разширения .css и .js. Съдържа няколко JavaScript компонента под формата на jQuery плъгини. Те предоставят допълнителни потребителски интерфейс елементи като диалогови прозорци и пояснения. Също така разширяват функционалността на някои съществуващи интерфейс елементи. Характеристиките, които правят Bootstrap предпочитан за използване са: лесна приспособимост, responsive дизайн, перфектна grid система, обширен списък на компонените, пакетни Javascript плъгини, предоставени стилове за всички основни HTML елементи и добра документация.

## 3.2 Програмна среда.

**Microsoft Visual Studio** е среда за разработка на софтуерни приложения за Windows и за платформата .NET Core, наследник на .NET Framework. Използва се за разработка на конзолни и графични потребителски интерфейс приложения, както и Windows Forms или WPF приложения, Web сайтове, Web приложения и Web услуги на всички поддържани платформи от Microsoft Windows, Windows Mobile, Windows CE, .NET Framework, .NET Core, .NET Compact Framework и Microsoft Silverlight.

Visual Studio предоставя мощна интегрирана среда за писане на код, компилиране, изпълнение, дебъгване (както за високо така и за машинно ниво), тестване на приложения, дизайн на потребителски интерфейс (форми, диалози, Web страници, визуални контроли и други), моделиране на данни, моделиране на класове, изпълнение на тестове, пакетиране на приложения и стотици други функции. Могат да се добавят и плъгини, които повишават функционалността на почти всяко ниво – включително добавянето на поддръжка за source-control системи (като Subversion и Visual SourceSafe или GitHub).

Visual Studio поддържа различни езици за програмиране (C#, VB.NET, C/C++, F#, XML/XSLT, HTML/XHTML, JavaScript, CSS и други) и различни технологии за разработка на софтуер (Win32, COM, ASP.NET Core, ADO.NET Entity Framework, Windows Forms, WPF, Silverlight и още десетки други Windows и .NET технологии).

Версията Visual Studio 2019 предлага много подобрения на производителността в подкрепа на работата с приложения за Windows, разработката на мултиплатформени мобилни приложения, Web и cloud development и др.

**ASP.NET Core**

ASP.NET Core е многоплатформенa рамка (cross-platform framework) с отворен код. Предоставя възможност за създаване Web приложения и услуги, IoT приложения, програмни модули за мобилни приложения и всякакъв вид Web-базирани solutions. Програмният продукт може да бъде „качен“ в облачно пространство (например Azure) или да се ползва локално. ASP.NET Core притежава отлична документация. ASP.NET Core осигурява интеграция на модерни рамки от страна на клиента (Angular, Blazor и др.) и потоци за разработка (MVC, WebAPI, Razor Pages, SignalR). Приложенията на ASP.NET Core се поддържат и могат да бъдат стартирани и от .NET Framework.

**Entity Framework Core**

Entity Framework (EF) Core е многоплатформена версия на популярната технология за достъп до данни Entity Framework. EF Core служи като обектно-релационен mapper (O/RM), като позволява на .NET разработчиците да работят с база данни, посредством .NET обекти.

В EF Core достъпът до данни се осъществява с помощта на модел. Един модел се съставя от entity класове и контекстен обект (DbContext), който представлява сесия с базата данни и позволява да се подават заявки и да се запазват данни. Моделът може да се генерира от вече съществуваща база данни (Database First); да се напише програмен код за модел, съответстващ на такъв в базата (Model First); или да се използват EF миграции за създаване на база данни от модел (Code First).

## **3.3 Система за управление на база данни**

**Microsoft SQL Server** е система за управление на релационни бази данни, разработена от Microsoft. В тази си роля (на система за управление) SQL Server има основната функция да съхранява и извлича данни, по заявки на други софтуерни приложения, които могат да вървят на същия или друг компютър в дадена мрежа (в т.ч. и Интернет). MS SQL Server има над дузина различни издания, които са подходящи за различни аудитории и различни натоварвания: от приложения за отделни персонални компютри до масивни приложения, работещи постоянно в онлайн режим и обхващащи огромно количество компютри. Microsoft SQL Server има много висока степен на защита.

SQL Server Management Studio(SSMS) **–** е софтуерно приложение, което се използва за конфигуриране, управление и администриране на всички компоненти в SQL Server. То съчетава широка група от графични инструменти с голям брой текстови редактори, осигуряващи на разработчиците и администраторите всички нива на достъп до сървъра.

## 3.4. Архитектурен модел.

Избраният архитектурен шаблон за дизайн на приложението е **Модел-Изглед-Контролер** (Model-View-Controller или MVC).  Архитектурният модел на Model-View-Controller (MVC) разделя приложенито на три основни групи компоненти: модели, изгледи и контролери. Този шаблон на работа помага да се постигне отделяне на бизнес логиката от графичния интерфейс и данните в едно приложение. Потребителските заявки се препращат към контролер, който е отговорен за работата с модела за извършване на потребителски действия и/или извличане на резултати от заявки. Контролерът избира изгледа, който да се покаже на потребителя, и му предоставя всички данни от модела, които той изисква.

**Модел** – ядрото на приложението, предопределено от областта, за която се разработва; обикновено това са данните от реалния свят, които се моделират и над които се работи – въвеждане, промяна, показване и т.н. Трябва да се прави разлика между реалния обкръжаващ свят и въображаемият абстрактен моделен свят, който е продукт на разума, който се възприема като твърдения, формули, математическа символика, схеми и други помощни средства. Например в приложението това са класовете, описващи потребителите, техните обекти, резервации, които са осъществили и т.н.

**Изглед** (англ. View) – тази част от изходния код на приложението, отговорна за показването на данните от модела. Изглед модела (view model) позволява да се оформят няколко изгледа (views) от един или повече модели (models). Този модел е оптимизиран за потребление и изпълнение. В случая изгледът се състои ASP страници.

**Контролер** – тази част от сорс кода (клас или библиотека), която взима данните от модела или извиква допълнителни методи върху модела, предварително обработва данните, и чак след това ги дава на изгледа. Контролерите са класове, които се създават в MVC приложението. Намират се в папка Controllers. Всеки един клас, който е от този тип, трябва да има име завършващо с наставка „Controller“. Контролерите обработват постъпващите заявки, въведени от потребителя и изпълняват подходящата логика за изпълнение на приложението. Класът контролер е отговорен за следните етапи на обработка: намиране и извикване на най-подходящия метод за действие (action method) и валидиране, че може да бъде извикан, взимането на стойности, които да се използват като аргументи в метода за действие, отстраняване на всички грешки, които могат да възникнат по време на изпълнението метода за действие.

Това разграничаване на отговорности помага при писане и актуализиране на програмен код, отстраняване на грешки и тестване даден на компонент (модел, изглед или контролер). Тези действия ще са по-сложни ако имат зависимости в две или повече от тези три области. Например логиката за потребителския интерфейс има тенденция да се променя по-често от бизнес логиката. Ако презентационният код и бизнес логиката се комбинират в един общ компонент, то този, който съдържа бизнес логиката, ще трябва да се променя при всяка актуализация на потребителския интерфейс. Това често довежда до грешки и изисква повторно тестване на бизнес логиката след всяка минимална промяна на потребителския интерфейс.

**Предимства на MVC:**

* Моделът е независим от контролера и изгледа.
* Моделът може да бъде планиран и осъществен независимо от другите части на системата.
* За един и същи модел могат да бъдат осъществени различни изгледи (интерфейси) – например Web интерфейс и нативен интерфейс към Facebook.
* Контролерът и изгледът могат да бъдат променени, без да се налага промяна в модела.

**Недостатъци на MVC:**

* Софтуерната система достига по-високо ниво на сложност откъм четимост и боравене с нея.

**ASP.NET Core MVC – Основни понятия**

**Маршрутизация (Routing)**

ASP.NET Core MVC е изграден на основата за маршрутизация в ASP.NET Core - мощен компонент за URL mapping, който позволява създаването на приложения с разбираеми и леснодостъпни URL адреси. Това дава възможност на разработчиците да именуват URL адреси, които работят добре с механизмите за оптимизация на търсачките (SEO) в Интернет пространството.

**Обвързване на модел (Model Binding)**

Обвързването на моделите на ASP.NET Core MVC преобразува данните от клиентските заявки (като например попълване на форми) в обекти, с които контролерът може да се борави. В резултат на това логиката на контролера не трябва да върши работата по определяне на вида на данните от входящите заявки; той просто взима данните като параметри в своите методи.

Пример:

public async Task<IActionResult> Login(LoginViewModel model, string returnUrl = null) { ... }

**Валидиране на модели ( Model validation)**

ASP.NET Core MVC поддържа валидиране чрез добавяне на атрибути за анотация на данните към модела. Те служат за проверка на валидността на подаваните стойности още от страна на клиента, преди да бъдат прехвърлени към сървъра, а в друг случай - откъм сървърна страна - преди да се извика действието на даден контролер.

Пример:

using System.ComponentModel.DataAnnotations;

public class LoginViewModel

{

[Required]

[EmailAddress]

public string Email { get; set; }

[Required]

[DataType(DataType.Password)]

public string Password { get; set; }

[Display(Name = "Remember me?")]

public bool RememberMe { get; set; }

}

Метод на контролер:

public async Task<IActionResult> Login(LoginViewModel model, string returnUrl = null)

{

if (ModelState.IsValid)

{

// work with the model

}

// At this point, something failed, redisplay form

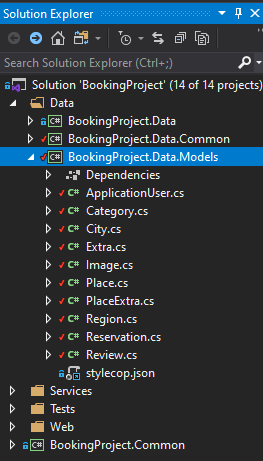
return View(model);

}

# ГЛАВА 4. Описание на компонентите на Web-системата и тяхното взаимодействие

## 4.1. База данни.

Моделът на базата е изграден като библиотека от класове и всяка отделна таблица е описана като един клас. По този начин с таблиците се работи като с обекти в обектно-ориентираното програмиране (фиг.2).



фиг.2 Класове в приложението

### 4.1.1 Класове - модели в приложението (Models)

**ApplicationUser.cs -** клас, описващ таблицата AspNetUsers (потребители)

**Category.cs -** клас, описващ таблицата Categories (категории за места под наем)  
**City.cs -** клас, описващ таблицата Cities (градове)   
**Extra.cs -**клас, описващ таблицата Extras (екстри)

**Image.cs -** kлас, описващ таблицата Images (снимки)

**Place.cs -** клас, описващ таблицата Places (места за отдаване под наем)

**PlaceExtra.cs** - клас, описващ таблицата PlaceExtras (екстри към място)  
**Region.cs -** клас, описващ таблицата Regions (области)

**Reservation.cs** **-** клас, описващ таблицата Reservations (резервации)

**Review.cs** - **-** клас, описващ таблицата Reviews (ревюта)

### 4.1.2. Достъп до данните от базата

Някои от командите на EF Core Tools (например командите за миграции) изискват да се създаде инстанция на DbContext по време на проектиране (at design time), с цел събиране на подробности за типовете обекти на приложението и как те ще се проектират в схемата на базата данни. В повечето случаи е желателно създаденият DbContext да бъде конфигуриран по подобен начин, както би бил конфигуриран по време на изпълнение(at run time). Има различни начини за създадаване на DbContext. За разработеното приложение е избрано DbContext да бъде осигурен с помощта на Services от доставчика на услуги (Application Service Provider). ASP.NET Core използва depencency injection като основна функция за управление на зависимости. За да знае средата как да борави с тези зависимости или „services“, те първо трябва да бъдат конфигурирани.

public class Program

{

public static void Main(string[] args)

=> CreateHostBuilder(args).Build().Run();

// EF Core uses this method at design time to access the DbContext

public static IHostBuilder CreateHostBuilder(string[] args)

=> Host.CreateDefaultBuilder(args)

.ConfigureWebHostDefaults(

webBuilder => webBuilder.UseStartup<Startup>());

}

public class Startup

{ // This method gets called by the runtime. Use this method to add services to the container.

public void ConfigureServices(IServiceCollection services)

{

services.AddDbContext<ApplicationDbContext>(

options => options.UseSqlServer(this.configuration.GetConnectionString("DefaultConnection")));

}

}

public class ApplicationDbContext : IdentityDbContext<ApplicationUser, ApplicationRole, string>

{

public ApplicationDbContext(DbContextOptions<ApplicationDbContext> options)

: base(options)

{

}

public DbSet<Place> Places { get; set; }

public DbSet<Review> Reviews { get; set; }

public DbSet<Image> Images { get; set; }

public DbSet<Category> Categories { get; set; }

public DbSet<Reservation> Reservations { get; set; }

public DbSet<City> Cities { get; set; }

public DbSet<Region> Regions { get; set; }

public DbSet<Extra> Extras { get; set; }

public DbSet<PlaceExtra> PlaceExtras { get; set; }

}

Пример – CategoriesService.cs

namespace BookingProject.Services.Data

{

using System.Collections.Generic;

using System.Linq;

using BookingProject.Data.Common.Repositories;

using BookingProject.Data.Models;

using BookingProject.Services.Mapping;

public class CategoriesService : ICategoriesService

{

private readonly IDeletableEntityRepository<Category> categoriesRepository;

public CategoriesService(

IDeletableEntityRepository<Category> categoriesRepository)

{

this.categoriesRepository = categoriesRepository;

}

public IEnumerable<T> GetAll<T>(int? count = null)

{

IQueryable<Category> query =

this.categoriesRepository.All().OrderBy(x => x.Name);

if (count.HasValue)

{

query = query.Take(count.Value);

}

return query.To<T>().ToList();

}

public T GetByName<T>(string name)

{

var category = this.categoriesRepository.All()

.Where(x => x.Name.Replace(" ", "-") == name.Replace(" ", "-"))

.To<T>().FirstOrDefault();

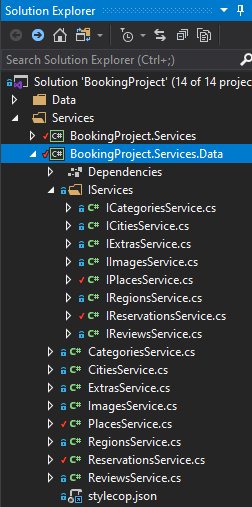
return category;

}

}

}

Папка Booking.Project.Data(фиг.3) съдържа всички Services, в които е описана логиката за извличане на данни от базата. Програмният код за тях е приложен в.

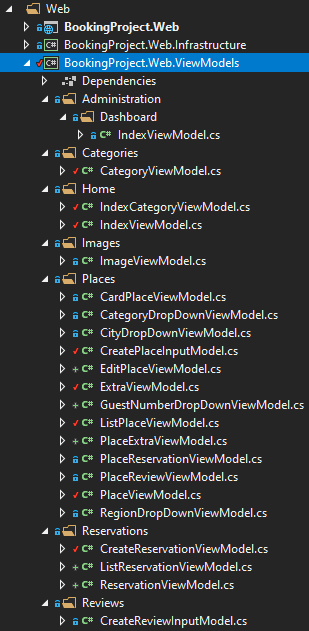
**

фиг.3 Структура на Service класовете

## 4.2. Модели за изглед (View Models)

Моделът на изглед е абстракция на изгледа(view), съставен от публични полета(public properties) и команди. Моделът на изглед е описан като състояние на данните в модела. За да функционира ефективно, се изисква свързваща технология или код. Тези модели определят правила, с които потребителя от клиентска страна трябва да се съобразява, за да се запази консистентност на данните в сървъра.

Създадените модели за изглед в Web-приложението са следните: (фиг.4)



фиг.4 Модели за изглед (View Models)

### 4.2.1. Библиотека AutoMapper

За създаване на зависимост (dependency injection) в приложението между класовете-модели (models) и моделите за изглед (view models) е използвана библиотеката AutoMapper. Тя служи за трансформиране на един тип обект в друг.

Пример:

**Основен модел**

public class User

{

public int Id { get; set; }

public string FirstName { get; set; }

public string LastName { get; set; }

public string Email { get; set; }

public string Address { get; set; }

}

**Модел за изглед**

public class UserViewModel

{

public string FirstName { get; set; }

public string LastName { get; set; }

public string Email { get; set; }

}

**Трансформиране (mapping):**

public UserProfile()

{

CreateMap<User, UserViewModel>();

}

public UserProfile()

{

CreateMap<User, UserViewModel>()

.ForMember(dest =>

dest.FName,

opt => opt.MapFrom(src => src.FirstName))

.ForMember(dest =>

dest.LName,

opt => opt.MapFrom(src => src.LastName))

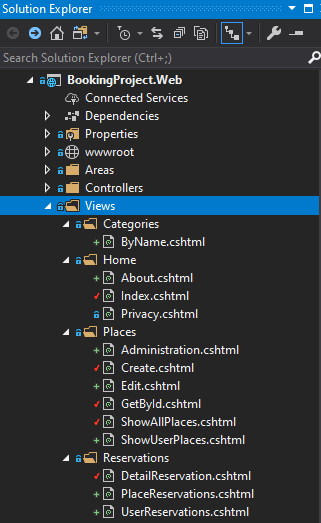
}

### 4.2.2. Валидиране на данни чрез анотации (Data Annotation Validations)

При създаване на mapping логика понякога може да има разминаване в именуването на полетата на основния модел и моделите за изглед. Затова се приема за добра практика валидационните атрибути за данни(data annotation validations ) да се описват и в моделите за изглед, като съответно извеждат на клиентска страна и подходящо съобщение за грешка.

## 4.3. Изгледи (Views)

Изгледите (Views) (фиг.5) са отговорни за представянето на съдържание в потребителския интерфейс. Те използват механизма за изглед на Razor (Razor view engine), за да вградят .NET код в HTML маркиране във файлове с разширение ''.cshtml''. Всяка логика в тях е минимална и трябва да се отнася само до представяне на съдържанието.



фиг.5 Изгледи (Views)

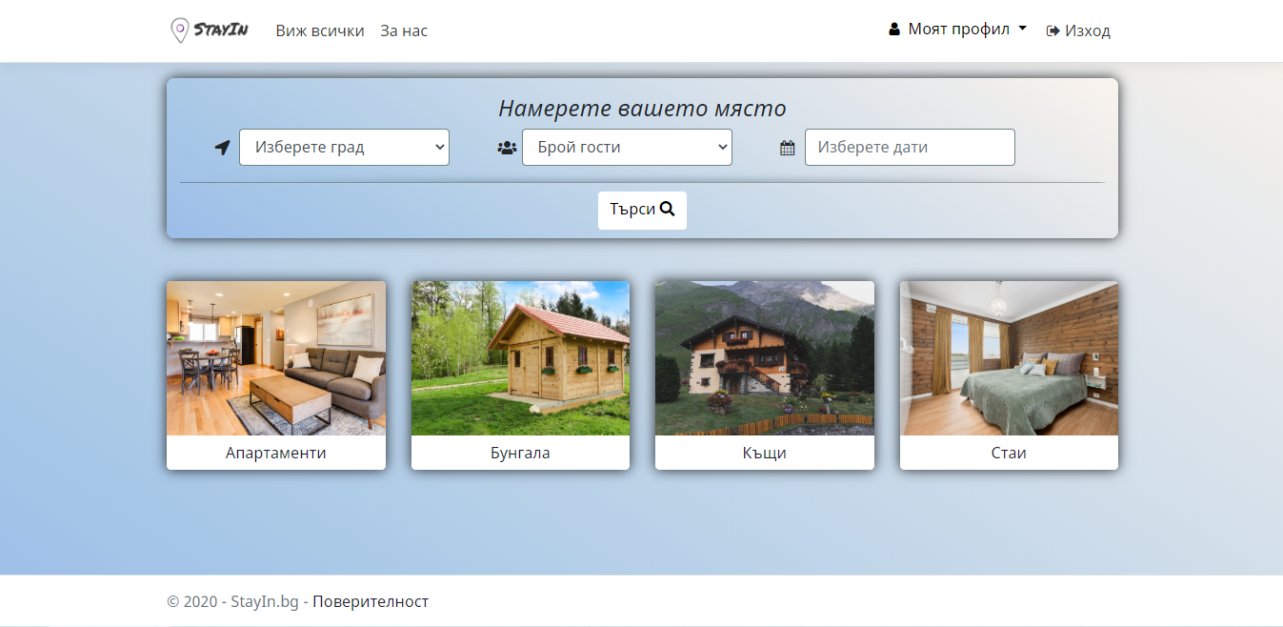
Програмният код за изгледите е добавен в приложение 2.

**Резултатите след компилиране на ''.cshtml'' – файловете, показани в браузър, са следните:**

### Начален изглед (Index View) (фиг.6)

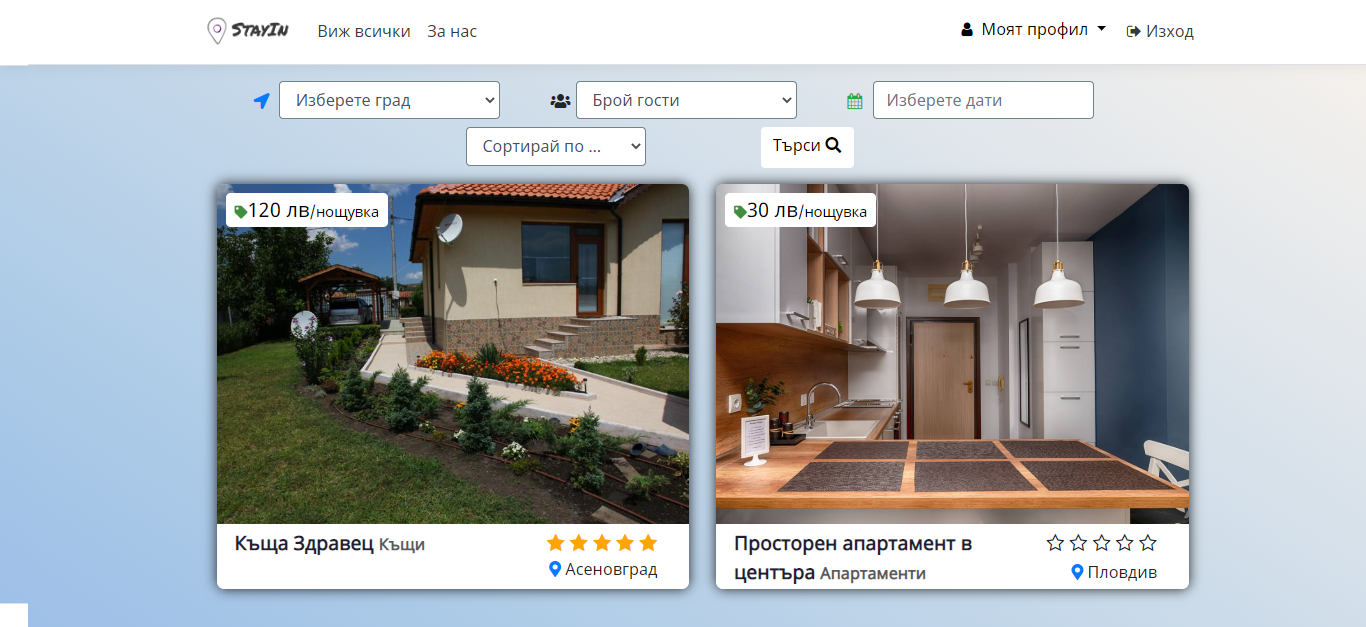
Съдържа компонентите, които потребителят вижда при начално зареждане на Web-приложението. Един от тях е предоставя на потребителя филтри за търсене на подходящ за наем обект. Филтрите съдържат 3 полета: град, брой наематели и дати на наемане. Първите две представляват полета за въвеждане. При избор на последното поле се отваря календар, в който потребителя може да избере начална и крайна дата на резервацията. За подредба на резултатите е приложен и филтър за сортиране спрямо цената на обекта (по нарастващ или намаляващ ред).

Друг вариант за търсене е да се търсят всички места от дадена категория без да се прилагат филтри.



фиг.6 Index View

* **Изглед за всички обекти в системата (ShowAllPlaces View)** (фиг.7)

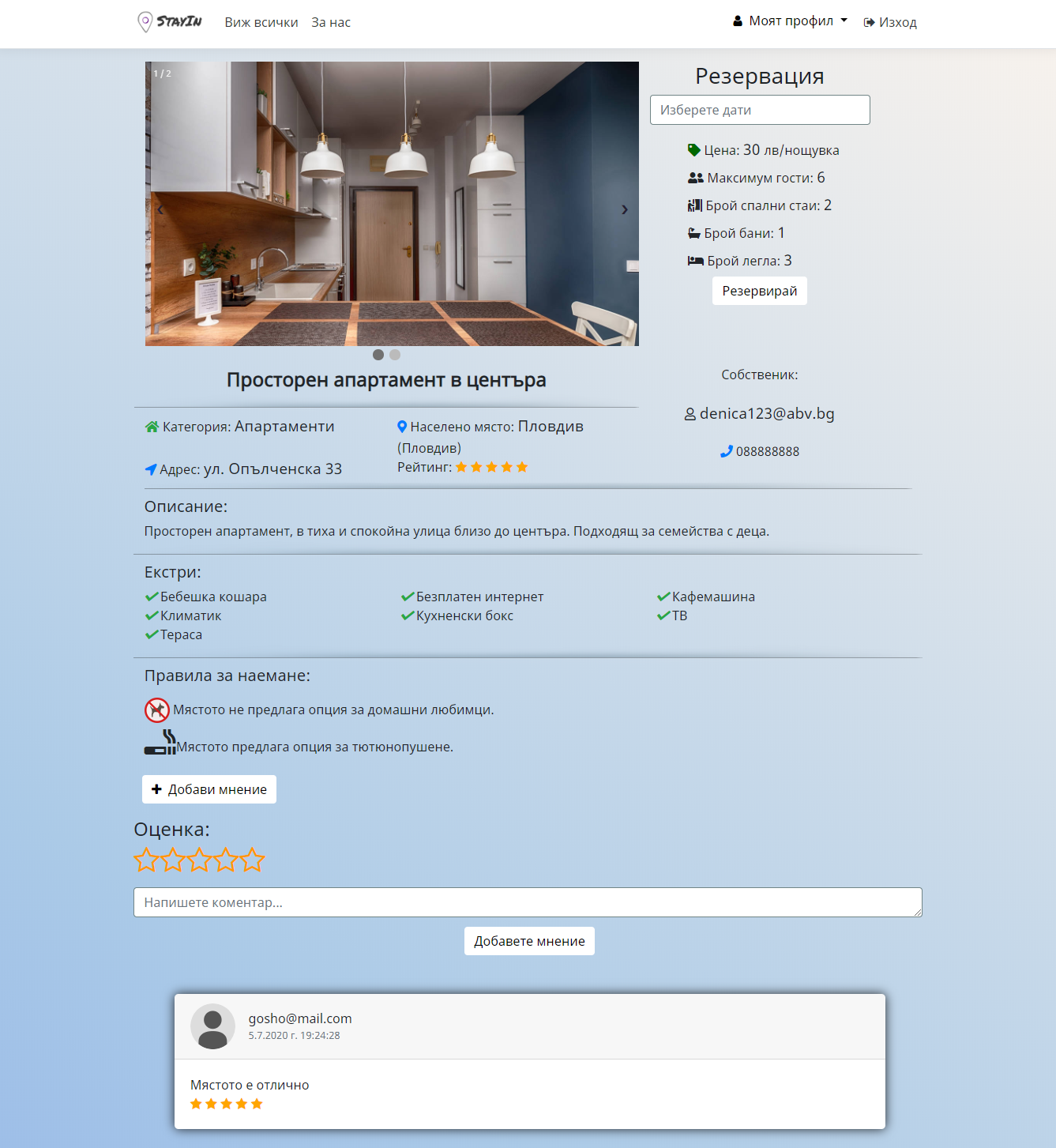
Този изглед визуализира всички възможни обекти за наемане. Формата за търсене от началната страница също отвежда до него - ако има избрани филтри, резултатите се свеждат до тези, които отговарят на изискванията. За всеки обект е изведена основна информация за име, категория, град, оценка получена от ревюта. При посочване с мишката върху обект се появява и допълнителна информация за него. 

фиг.7 ShowAllPlaces View

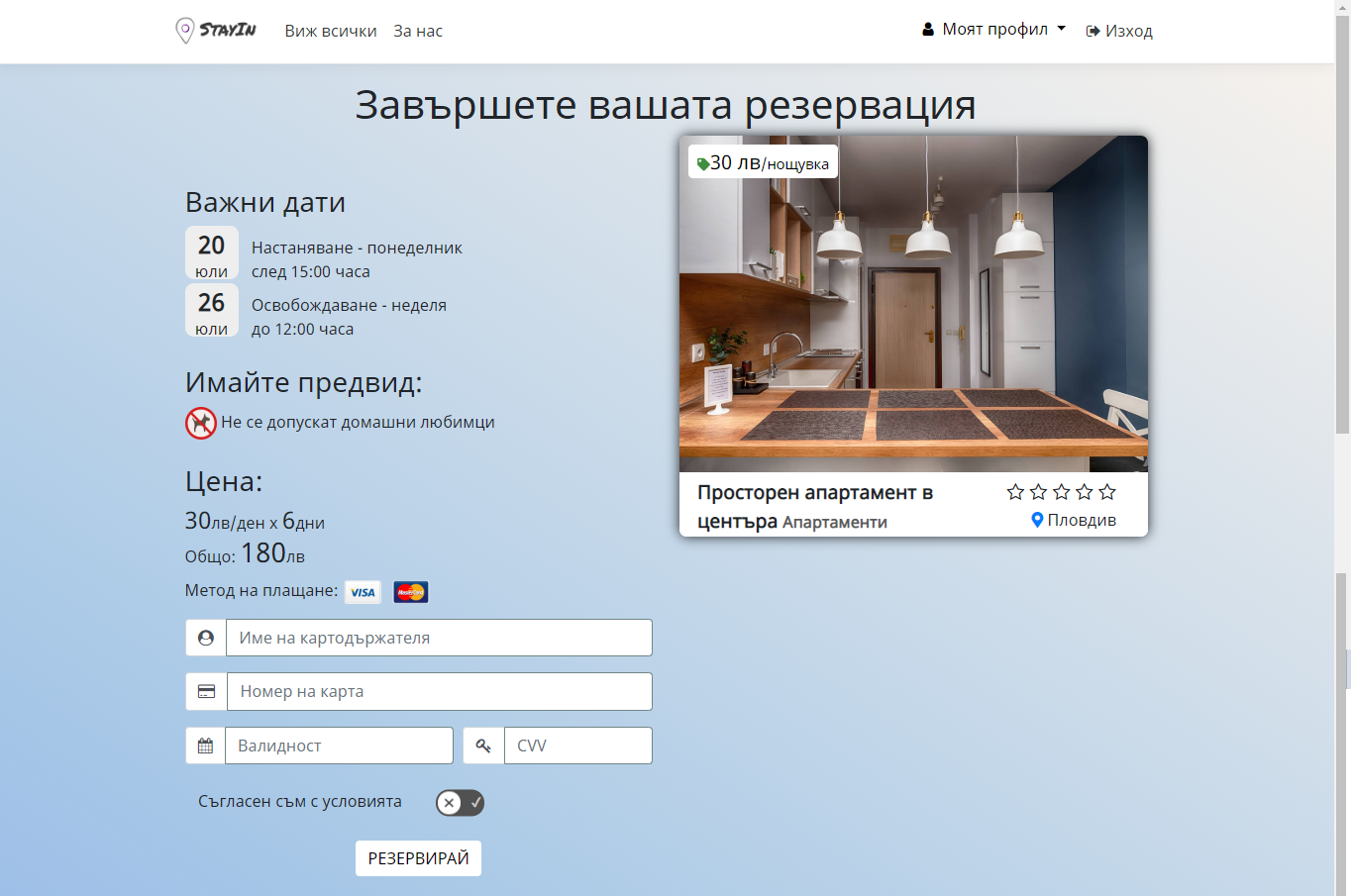
* **Изглед за обект по идентификационен номер (GetById View)** (фиг.8)

Изгледът визуализира детайлите за определен обект. Интерфейсът разполага с удобна галерия. След това в страницата има подробна информация за обекта и неговите екстри, а най-отдолу стои секцията за оценки и коментари.

На тази страница се намира и секцията за резервация. При избиране на дати за резервация, системата ще изиска от потребителя да се идентифицира чрез вход в системата или да бъде направена нова регистрация. Ако потребителят е вече в системата, то той ще се отведе към изгледа за потвърждение на резервацията. (фиг.9) Там са показват най-важните неща за нея, а именно – датите, правилата определени от собственика и обобщена информация за обекта, който предстои да се наеме. След въвеждане на данни за кредитна карта и съгласяване с условията, бутонът за резервация става активен. След успешно запазване, потребителят получава известие в попъп прозорец и бива пренасочен към страница, съдържаща всички негови предстоящи и предходни резервации.



фиг.8 GetById View



фиг.9 DetailReservation View

**Изгледите за потребителите на системата включват: изглед за преглед на резервации, за добавяне на собствено място под наем, за списък с всички негови места и за настройки на профила.**

* **Изглед за резервации, направени от даден потребител (UserReservations View)** (фиг.10)

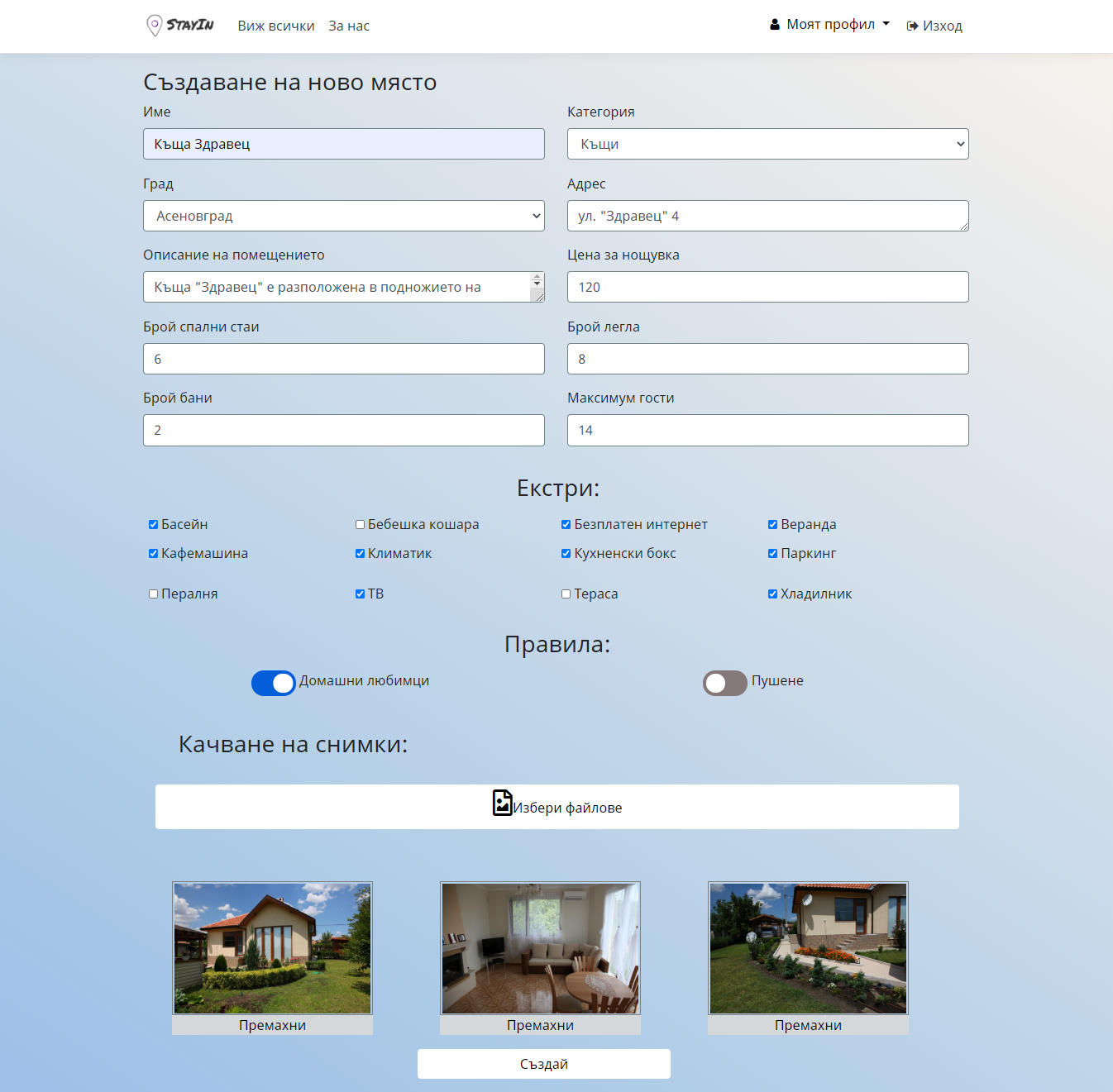
Резервациите се извеждат в списък от предстоящи и отминали. Във всеки запис за резервация се съдържа основна информация относно обекта, собственика, датите, съответно за колко дни е нает, както и сумата за престоя.



фиг.10 UserReservations View

* **Изглед за добавяне на ново място в системата (Create View)** (фиг.11)

Страницата за добавяне на новo място представлява форма, в която се въвежда пълната информация за новата обява. В началото се попълва основната информация като име, адрес, цена на нощувка и описание. Категория и град се зареждат се избират от падащо меню. Въвеждането на екстри и условията на обекта се осъществява с бутони от тип Checkbox. Собственикът трябва да прикачи поне една снимка, за да довърши операцията. Той има възможността да избере файлове чрез бутон, който отваря файловия мениджър. След тяхното прикачване снимките излизат подредени една до друга за финален преглед от потребителя преди да бъдат изпратени към сървъра. В този момент могат да бъдат премахнати или да се добавят още снимки.

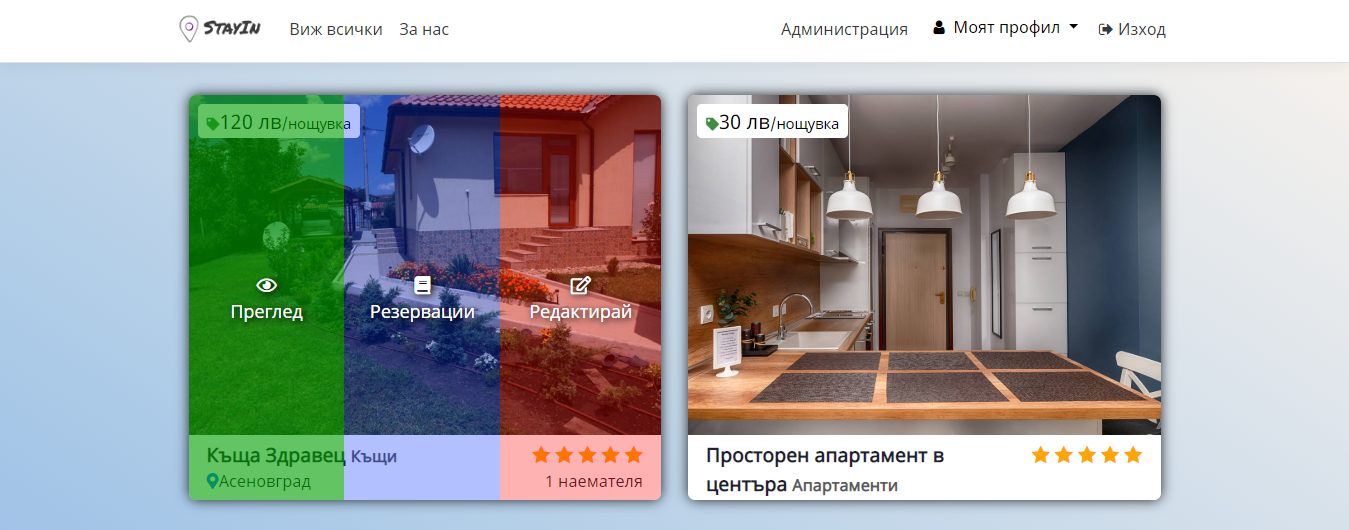


фиг.11 Create View

* **Изглед за управление на обектите (ShowUserPlaces View)** (фиг.12)

След създаване на обявата, потребителят се пренасочва към панела за управление. Извежда се списък от всички притежавани обекти на съответния юзър като при задържане на мишката върху някой обект се показват 3 възможности – Преглед, Резервации, Редактирай.

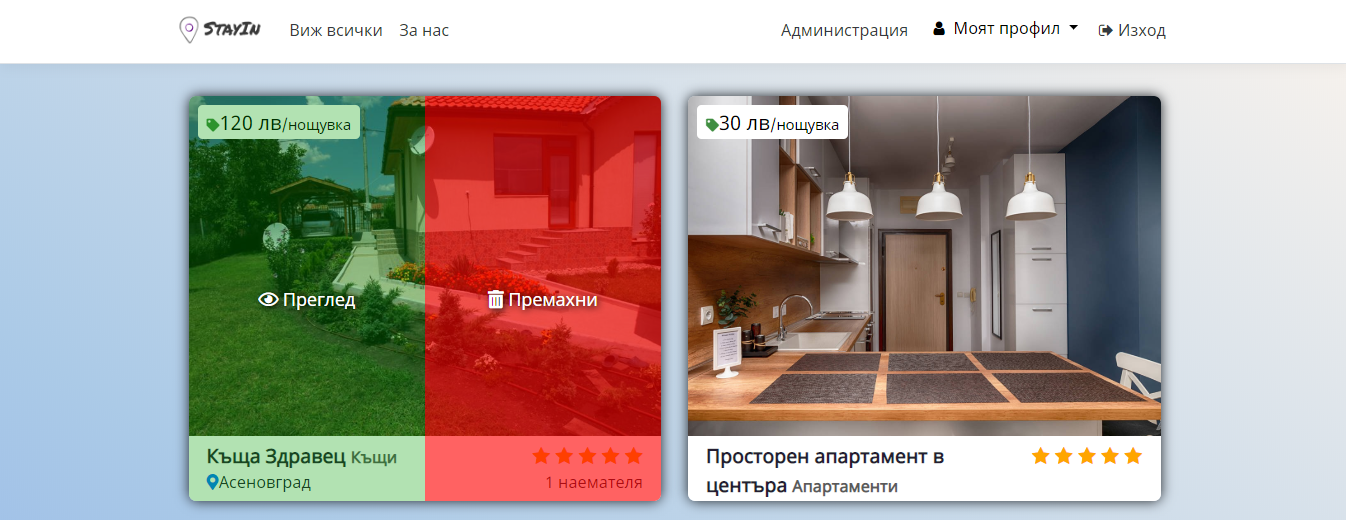
При избор на преглед потребителят се пренасочва към детайлната страница на обекта. В таб резервации се отваря страница, съдържаща информация за предстоящи или отминали резервации за конкретното място. Всеки запис съдържа данните за наемателя, датите, за които е запазена и общата сума за престоя.



фиг.12 ShowUserPlaces View

* **Изглед за контролен панел на администратор (Administration View)** (фиг.13)

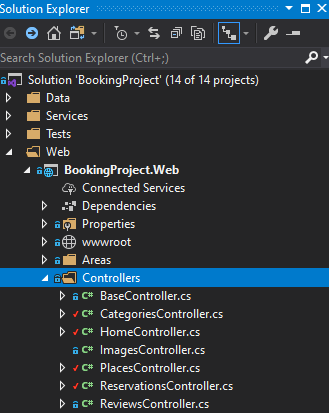
Администраторът на сайта разполага с допълнително поле „Администрация“ в главното меню. То отваря списък от всички съществуващи обекти в системата. При задържане на мишката върху някой от тях администраторът има опции за преглед и изтриване на мястото. Така той го премахва от Web- приложението, то спира да е видимо за юзърите, но данните за него остават в базата.



фиг.13 Administration View

## 4.4. Контролери (Controllers)

Контролерите(Controllers) в приложението (фиг.14) са компонентите, които управляват взаимодействието с потребителя, работят с модела и в крайна сметка избират изглед, който да изобразяват. В MVC модела, контролерът е входната точка за въвеждане и е отговорен за избора на кои модели да работи и кой изглед да изобрази (оттук и името му - контролира как приложението отговаря на дадена заявка).



фиг.14 Контролери (Controllers)

# ГЛАВА 5. Приложимост. Изводи.

Разработена е Web-система за отдаване на помещения под наем, обхващаща всички основни нужди на един потребител. Основната цел на тази дипломна работа е създаде продукт, осигуряващ на ползвателя безпроблемно резервиране на място или добавяне на собствен обект в системата. Изграден е приятен и удобен за употреба интерфейс с адаптивен дизайн, който улеснява работата с приложението както на персонален компютър, така и на мобилно устройство.

От тази гледна точка настоящата дипломна работа е актуална, като описва концепцията и инструментите за проектиране, създаване и администриране на Web-приложение, като се използват съвременни технологии и е избрана тематика с добра приложимост.

Стига се до заключението как развитието на познанията в компютърните науки позволява да се разработи лесноизползваем и достъпен софтуер, който улеснява значително операциите с данни и който би могъл да се внедри в най-различни сфери.