****

**ДИПЛОМНА РАБОТА**

Тема:

Уеб сайт на фирма за намиране и наемане на жилища и офис сгради

Ръководител:

**Инж. Десислава Христова Георгиева**

Ученик: **Хюсеин Джингиз Рашид**

Професия: **Системен програмист**

Специалност: **Системно програмиране**

**2025г. гр. Бургас**

**Съдържание:**

1. **Увод:**  
   1.1. Основни цели и задачи на дипломния проект  
   1.2. Кратко описание на функционалностите на уеб сайта  
   1.3. Обосновка на избора на тема и технологии
2. **Основна част:**  
   2.1. Описание на изискванията към уеб сайта  
   2.2. Проектиране на архитектурата  
   2.2.1. Трислойна архитектура (Слой за данни, слой за услуги, слой за потребителски интерфейс)  
   2.3. Проектиране на база данни  
   2.4. Разработка на основни функции на системата  
   2.4.1. Каталог на имоти  
   2.4.2. Търсене и филтриране на имоти  
   2.4.3. Профил на потребителя  
   2.4.4. Заявки за огледи  
   2.4.5. Административен панел  
   2.5. Технологии и инструменти за реализация  
   2.5.1. Избор на технологии (C#, ASP.NET или JavaScript)  
   2.5.2. Описание на използваните библиотеки и фреймуърци (Framework)  
   2.6. Компонентни тестове на ключовите функционалности.
3. **Заключение:**  
   3.1. Резултати от проекта  
   3.2. Проблеми и предизвикателства при разработката  
   3.3. Препоръки и възможности за доразвиване на проекта  
   3.4. Потенциал за бъдещо приложение и разширения на системата
4. **Списък на използваната литература:**  
   (Изброяване на всички използвани източници, подредени по азбучен ред.)
5. **Приложения:**  
   5.1. Скриптове и код на системата  
   5.2. Диаграми и UML модели  
   5.3. Примерни резултати от тестове  
   5.4. Допълнителни екранни снимки на интерфейса и функционалностите
   1. **. Основни цели и задачи на дипломния проект:**

Основната цел на дипломния проект е да се създаде функционален и добре структуриран уеб сайт за наемане на жилища и офис сгради. Проектът трябва да бъде разработен с трислойна архитектура, като включва слой за данни, слой за услуги и слой за потребителски интерфейс. Целта е да се реализират основни функционалности за търсене, филтриране, подаване на заявки за огледи и управление на имоти, като същевременно се осигури лесна администраторска част за управление на обявите и запитванията.

**Основни цели:**

1. **Разработване на пълноценен уеб сайт**:
   * Уеб сайтът ще бъде проектиран като онлайн платформа, която позволява на потребителите да търсят, разглеждат и наемат жилища или офисни пространства.
   * Интерфейсът ще бъде удобен и интуитивен, за да гарантира положително потребителско изживяване.
   * Административният панел ще осигури възможност за управление на обявите, като създаване на обяви, изтриване и т.н., потвърджадаване на заявките за огледи и генериране на доклади.
   * Вградена система за заявяване на огледи, която включва календарна интеграция и изпращане на известия при потвърждение или отказ.
   * Поддръжка на множество езици, включително български, за по-добро потребителско изживяване.
   * Мобилна оптимизация, която гарантира правилното функциониране на платформата на различни устройства.
   * Възможност за добавяне на обяви от потребители с роля "администратор".
   * Автоматизирано изпращане на имейли при промяна в статуса на обява или при актуализация на важни детайли.
   * Интегрирана система за известия при промени в състоянието на имота, като нови цени, статуси и наличност.

**1.2. Кратко описание на функционалностите на уеб сайта:**

**Rental Portal** е уеб сайт за отдаване под наем на имоти, който включва следните основни функционалности:

1. **Управление на потребителски профили**: Потребителите могат да създават и редактират профили, да виждат история на наемите и резервациите си.
2. **Търсене и филтриране на имоти**: Възможност за търсене на имоти по различни критерии като тип, цена, площ, местоположение и други.
3. **Създаване на обяви за имоти**: Потребителите могат да добавят нови имоти с подробности като описание, снимки и цени.
4. **Заявки за огледи**: Потребителите могат да искат огледи на имоти, като заявките се обработват от администраторите.
5. **Административен панел**: Администраторите могат да управляват потребители, имоти и да следят статистики за сайта.
6. **Ревюта и оценки на имоти**: Потребителите могат да оставят ревюта и оценки на наетите имоти.
7. **Интеграция на плащания**: Платформи за обработка на плащания за наем и генериране на фактури.
8. **Поддръжка на мобилни устройства**: Сайтът е оптимизиран за мобилни устройства.

Целта на сайта е да осигури лесен и безопасен начин за наемане на имоти с удобни инструменти за потребителите и администраторите.

1.3. Обосновка на избора на тема и технологии:

### Обосновка на избора на тема и технологии за ****Rental Portal****

#### 1. **Тема на проекта: Rental Portal**

**Rental Portal** е уеб платформа, която предлага услуги за отдаване под наем на имоти, като осигурява лесен достъп за потребителите до обяви, информация за имоти, възможности за заявяване на огледи и пълно управление на процеса на наемане. Темата е актуална и значима, защото наемането на имоти е важна част от ежедневието на много хора и бизнеси. Онлайн платформите за недвижими имоти стават все по-важни, а сега има нужда от ефективни и сигурни решения за управлението на процесите в този сектор. И аз имах интерес върху интернет програмирането.

#### 2. **Избор на технологии:**

##### **ASP.NET Core:**

* **Реализиране на съвременни уеб приложения**: ASP.NET Core е мощен, гъвкав и популярен фреймуорк за изграждане на уеб приложения с висока производителност. Той е подходящ за **Rental Portal**, защото осигурява бързо и надеждно изпълнение на сървърната логика, добра сигурност, както и възможност за лесна интеграция с различни външни системи като плащания, известия и т.н.
* **Поддръжка на MVC архитектура**: С ASP.NET Core е лесно да се имплементира **многостепенна архитектура (MVC)** за разделяне на логиката на потребителски интерфейс (View), бизнес логика (Controller) и данни (Model), което прави приложението по-гъвкаво, поддържано и разширяемо.

##### **SQL Server:**

* **Издръжливост и сигурност**: SQL Server е надеждна и ефективна система за управление на бази данни, подходяща за съхранение на критична информация като данни за потребители, имоти, заявки и плащания. За **Rental Portal** е важно да се гарантира целостта и сигурността на данните, а SQL Server предлага множество функции за защита и възстановяване на данни.
* **Мащабируемост**: SQL Server предлага възможност за мащабиране при увеличаване на броя на потребителите и обявите за наем. Това е особено важно за потенциалния растеж на Rental Portal.

##### **SMSS (SQL Server Management Studio):**

* **Управление на бази данни**: SMSS предоставя мощен графичен интерфейс за управление на базата данни SQL Server. С него администраторите могат да създават, управляват и оптимизират бази данни, да изпълняват запитвания, да анализират производителността и да осигуряват безопасността на данните. За разработчиците на **Rental Portal** SMSS е лесен и удобен инструмент за работа с данни, което ускорява процеса на разработка и поддръжка на приложението.
* **Поддръжка на големи данни**: SMSS помага при анализа и оптимизацията на големи обеми от данни, като например търсене и филтриране на обяви за имоти, което е важно за функционалността на **Rental Portal**.

**Front-End: HTML, CSS, Bootstrap:**

* HTML: Използва се за изграждане на структурата на уеб страниците на Rental Portal. Това включва създаване на елементи като форми, списъци с имоти, таблици за потребители и други компоненти, които осигуряват функционалност и навигация в платформата.
* CSS: Отговорен е за визуалния стил на уеб страниците. С CSS се дефинират цветовете, шрифтовете, разстоянията и други елементи на дизайна, които правят платформата привлекателна и лесна за използване.
* Bootstrap: Използва се за ускоряване на процеса на дизайн, като осигурява готови компоненти и стилове. Това включва навигационни барове, бутони, формуляри и мрежови структури, които гарантират отзивчив (responsive) и съвременен дизайн, съвместим с различни устройства.

Тези технологии се използват за създаване на интуитивен и удобен за потребителя интерфейс, който е лесен за навигация и адаптивен към мобилни устройства.

#### 3. **Заключение:**

Изборът на темата **Rental Portal** и технологиите **ASP.NET Core**, **SQL Server** и **SMSS** е оправдан, защото тези технологии осигуряват висока производителност, сигурност, мащабируемост и лесна поддръжка на проекта. Те са идеални за създаване на сигурна и ефективна уеб платформа, която може да расте и да отговори на нуждите на потребителите в бързо развиващия се пазар на недвижими имоти.

1. **Изграждане на трислойна архитектура**:

Трислойната архитектура е основен принцип при изграждането на уеб приложения и осигурява ясно разделение на отговорностите между различните компоненти на системата. Това подобрява мащабируемостта, сигурността и поддръжката на платформата. В настоящия проект се прилага следният модел:

**2.1. Слой за данни (Data Layer):**Слоят за данни е най-долният слой в архитектурата и играе ключова роля в съхраняването и управлението на информацията. Той е отговорен за взаимодействието с базата данни, осигуряването на целостта на данните и ефективното им извличане при необходимост.

* Данните за потребителите, имотите и заявките за огледи ще бъдат съхранявани в добре структурирана релационна база данни.
* Използвани технологии:
  + **Microsoft SQL Server** – релационна система за управление на бази данни (RDBMS), използвана за съхранение и управление на информацията в проекта. Осигурява надеждност, сигурност и добра производителност, като поддържа мащабируемост и възможност за работа с големи обеми данни.
  + **SQL Server Management Studio** (SMSS) – интегрирана среда за управление на Microsoft SQL Server, която позволява администриране, писане и изпълнение на SQL заявки, наблюдение на производителността и конфигуриране на базата данни. Използва се за управление на релационната схема, индексиране, създаване на резервни копия и други административни задачи.
* Ще бъдат създадени таблици за потребители, имоти, заявки за огледи, нотификации, както и допълнителни таблици за съхранение на отзиви, предпочитания.
* Използване на **Entity Framework Core** за ефективно управление на ORM (Object-Relational Mapping), което ще позволи по-бърза работа с базата данни.
* Връзките между различните таблици ще бъдат дефинирани чрез **ключови връзки (foreign keys)**, за да се гарантира консистентността на данните.
* Ще се прилагат **оптимизационни техники** като индексиране на таблици и кеширане на често използвани данни (като пароли) с цел подобряване на производителността.
* Автоматизирано създаване на резервни копия (backups), за да се предотврати загуба на информация при технически неизправности.

**2.2. Слой за услуги (Business Logic Layer):**Този слой се намира между слоя за данни и потребителския интерфейс и е отговорен за обработката на заявките, прилагането на бизнес правилата и осигуряването на логиката зад функционирането на системата.

* Ще включва услуги за управление на потребители, обработване на регистрация, вход и настройка на потребителския профил.
* Разработка на услуга за търсене и филтриране на имоти, която ще позволи на потребителите да задават различни критерии като местоположение, цена, брой стаи и налични удобства.
* Обработка на заявки за огледи, като системата ще осигури възможност за изпращане, потвърждение или отказване на заявки, както и автоматично известяване на потребителите чрез имейл или нотификации.
* Генериране на статистически отчети за администраторите относно популярността на имотите, броя на извършените огледи и най-търсените критерии.
* Интеграция на система за плащания, която ще позволи на потребителите да извършват транзакции за резервация или наемане на имот.
* Ще бъдат внедрени алгоритми за автоматично препоръчване на имоти, базирани на предишни търсения, запазени обяви и потребителски предпочитания.
* Логика за управление на оценки на имоти, като потребителите ще могат да оставят коментари и звездни оценки след извършен оглед.
* Осигуряване на защита срещу неоторизиран достъп чрез валидиране на входните данни и прилагане на ролево-базирана система за достъп.

**2.3. Слой за потребителски интерфейс (Presentation Layer):**Този слой представлява видимата част на системата и осигурява взаимодействие с потребителите чрез уеб браузър или мобилни устройства. Той трябва да бъде удобен, интуитивен и визуално привлекателен.

* Уеб интерфейсът ще бъде изграждан **с HTML, CSS, JavaScript** (с помощта на **Bootstrap**), което ще осигури респонсив дизайн и добра визуална структура.
* Ще бъде внедрен **AJAX** за динамично зареждане на съдържание, което ще подобри потребителското изживяване, като намали презарежданията на страниците.
* Реализиране на търсачка с интуитивен дизайн, която ще позволява бързо въвеждане на критерии и визуализиране на резултати в реално време.
* Внедряване на **Google** **Maps** **API**, което ще позволи интерактивно разглеждане на локациите на имотите директно в сайта.
* Секция за управление на профилите, където потребителите ще могат да редактират личните си данни, да запазват любими обяви и да управляват получените нотификации.
* Ще бъдат създадени различни изгледи за стандартни потребители и администратори, като административният панел ще предоставя инструменти за модериране на обявите, управление на потребителите и анализ на статистики.
* Осигуряване на поддръжка на тъмна и светла тема, която потребителите ще могат да сменят според личните си предпочитания.
* Оптимизация за мобилни устройства, така че сайтът да работи гладко както на настолни компютри, така и на смартфони и таблети.
* Ще бъде разработена система за уведомления, която ще информира потребителите за промени в техните запазени имоти, нови съобщения и предстоящи огледи.

1. **Интерактивност и персонализация**:
   * **Потребителски профили**: Системата трябва да предоставя възможност на потребителите да създават персонализирани профили, в които да запазват своите предпочитания и интереси. Това включва опцията за съхранение на любими имоти в страницата с обяви (Properties). Персонализацията може да обхваща и настройване на известия за промени в цените или актуализации на обявите, като например промяна на заглавието. Освен това, потребителите трябва да имат възможността да променят паролата си, а ако я забравят, да могат да посетят страницата „Забравили ли сте паролата?“ и да получат имейл с линк за възстановяване на паролата.
   * **Търсене с филтри**: Системата трябва да предлага усъвършенстван инструмент за търсене на имоти, който да включва разнообразие от филтри. Те могат да бъдат по различни категории като цена, местоположение, размер на имота, тип на имота (къща, апартамент, офис.), филтрацията става автоматично като използваме техниката **AJAX**
2. **Управление на обяви**:
   * **Административен панел**: Системата трябва да предоставя мощен административен панел, който позволява на администраторите и агентите да управляват всички обяви в сайта. Това включва възможност за лесно добавяне на нови имоти с всички необходими детайли (снимки, описание, характеристики, цена и т.н.), както и редактиране на съществуващи обяви. Системата трябва да предлага и функция за премахване на обяви, които вече не са актуални или са били продадени/наети.
   * **Анализ на популярността на обявите**: Възможността за генериране на доклади и аналитични данни е от съществено значение за администраторите и агентите. Системата трябва да предлага подробни отчети, които показват какви обяви са най-гледани, кои обяви са най-често добавяни в любими или запазвани за бъдещо разглеждане, както и кои обяви са имали най-много заявки или контакти от потенциални купувачи. Това може да бъде подкрепено с графики и диаграми, които да предоставят преглед на тенденциите на пазара. Това става в административният панел.
3. **Сигурност и стабилност на системата**:
   * **Сигурност на потребителските данни:** За да се осигури висока степен на защита на личните данни на потребителите, сайтът трябва да използва съвременни технологии за криптиране и защита на информацията като хеширане на паролите. Това включва криптиране на пароли и лични данни. Системата осигурява автентикация и авторизация за достъп до потребителските профили и административните панели. Ако потребителят не е удостоверен, няма да може да поиска срещи за разглеждане на имот и; при опит – ще бъде пренасочен към страницата за вход или регистрация.
   * **Контрол на достъпа:** Важно е да се внедрят различни нива на достъп за различни типове потребители. Например, само администраторите и сертифицираните агенти трябва да имат възможност за редактиране или премахване на обяви, докато обикновените потребители могат само да разглеждат обявите и да запазват любими имоти.

**Основни задачи:**

1. **Изграждане на база данни**:
   1. Създаване на база данни:
      * За да бъде осигурена надеждна и ефективна работа на платформата, първата стъпка в процеса на разработка е изграждането на база данни. Тя ще съдържа всички необходими данни за потребителите, имотите, заявките за огледи и настройките на известията.
      * Основните таблиците, които са създадени са:
        + **Users** - съдържа информация за регистрираните потребители като име, имейл, парола, роля (администратор, агент, клиент), дата на регистрация и настройки на профила.
        + **Properties**- описва всеки обявен имот с полета като заглавие, описание, адрес, цена, площ, етаж, брой стаи, наличност, снимsки и дата на последна актуализация.
        + **UserFavorites** – таблица без първичен ключ, запазва любимите имоти на потребителя и ви показва любимите ви имоти в раздела с настройки, където можете да промените паролата си и да изберете дали да получавате известия от определени типове или не
        + **Appointments** - включва информация за огледите, като кой потребител е направил заявката, за кой имот се отнася, дата на заявката и дали е потвърдена.
        + **NotificationsAndPreferences** - позволява на потребителите да персонализират своите известия за нови имоти, промени в цените или други свойства на имотите и предстоящи огледи.

1.2. Оптимизация на база данни:

* + - За да бъде осигурена ефективност при търсене и филтриране на имоти, ще бъдат изпозлвани:
      * Индекси върху най-често използваните колони като цена (price), площ (area) и адрес (address), за да се подобри производителността при заявки.
      * Нормализация на базата данни, за да се избегне излишно дублиране на данни.
      * Релации между таблиците, които осигуряват правилната организация на данните и улесняват интеграцията на функциите в приложението.

1. **Разработка на функционалности за потребителския интерфейс:**

**Дизайн и навигация:** За да се осигури удобно потребителско изживяване, интерфейсът трябва да бъде интуитивен и достъпен както от настолни компютри, така и от мобилни устройства. Основните секции на сайта ще включват:

* + - Начална страница с навигация на другите страници.
    - Страница за разглеждане на обяви с възможност за филтриране по цена, местоположение, тип на имота, брой стаи и други важни свойства.
    - Детайлна страница за даден имот, съдържащи галерия със снимки, описание, карта с локация (интегриран от Google Maps API) и бутон за заявка на оглед.
    - Форма за контакт, който служи като имейл, изпраща се имейл на [rentalproject123@abv.bg](mailto:rentalproject123@abv.bg) и администратори които имат достъп до този имейл могат да видят съобщенията.

**Регистрация, вход и управление на профил:**

* + - Регистрацията ще изисква въвеждане на име, имейл и парола. Като паролата се хешира автоматично. Хеширането е еднопосочна математическа функция, която превръща данните в низ от неописуем текст, който не може да бъде обърнат или декодиран. В контекста на киберсигурността, хеширането е начин за запазване на чувствителна информация и данни — включително пароли, съобщения и документи — защитени.
    - Потребителите ще могат да актуализират информацията си, като парола и да настройват известията си и да виждат любимите обяви.

**Имейл услуги:**

* + - **Напомняне за предстоящи огледи на имоти:**
      * Потребителите ще получават автоматични напомняния за предстоящите им огледи на избраните имоти. Това ще включва детайли като дата, час, адрес на имота и контактна информация на агента или собственика.
    - **Известия при промени в цената или други характеристики на имот:**
      * Ако потребителят е добавил даден имот в списъка си с любими, той ще получава известие при всяка промяна в цената, наличността или други ключови характеристики на имота. Това ще помогне на потребителите да следят актуалните условия на пазара в реално време.
    - **Потвърждение на заявка за оглед от администратора:**
      * След като потребител подаде заявка за оглед на даден имот, администраторът на сайта или собственикът на имота ще трябва да потвърдят заявката. Потребителят ще получи известие за одобрението или отказа, заедно с допълнителни инструкции относно уговорения оглед.
    - **Забравена парола:**
      * Ако потребителят е забравил паролата си, той може да поиска, за да може паролата му да бъде нулирана. Изпраща се имейл до потребителя, който съдържа връзка към страница в нашия сайт, която не може да бъде достъпна по друг начин, имате нужда от конкретен токен за достъп до тази конкретна страница.

1. **Разработка на функционалности за административния панел:**

**Управление на обяви:** Администраторите ще имат достъп до система за управление на обявите, която ще им позволи:

* + - Добавяне на нови обяви с детайлно описание, цена, местоположение и други важни свойства, ключови параметри.
    - Редактиране на съществуващи обяви, като име, местоположение, цена и други. За да се актуализира информацията при нужда.
    - Качване на изображения като URL за по-добра визуализация на имотите. Може да се качва множество снимки ако URL са разделени със запетаи.
    - Деактивиране или премахване на оферти, които вече не са налични или не отговарят на актуалните изисквания.

**Управление на заявки за огледи:** Администраторите ще имат контрол върху процеса на заявяване на огледи чрез следните функции:

* + - Преглед на списък със всички подадени заявки за огледи на имоти.
    - Възможност за одобряване или отказване на заявките, като се изпраща автоматично уведомление до потребителя.
    - Анализ на най-търсените имоти чрез автоматично генерирани доклади, които ще показват броя на заявките за всеки имот.

1. **Интеграция с логика за услуги:**

**Обработка на заявки:** Заявките за огледи и интерес към обяви ще се обработват чрез бек-енд логика, която ще осигури:

* + - Автоматично изпращане на потвърждения - след успешно подадено заявка, системата ще изпраща потвърждение по имейл на потребителите с информация за датата, часа и титлата на имота.
    - Уведомления за администраторите - при нова заявка, съответният администратор ще може да потвърди или откаже заявката директно в административния панел.
    - Отказ и повторно заявяване - ако потребителят не може да присъства на уговорения оглед, ще има възможност да откаже или заяви нов час чрез своя профил.

**Филтриране на резултатите:** За да се улесни търсенето на имоти и да се предложат най-подходящите оферти, ще бъдат внедрени различни алгоритми за интелигентно филтриране и препоръчване:

* + - Филтриране по зададени критерии - потребителите ще могат да търсят имоти въз основа на следните параметри:
      * Цена (минимална и максимална);
      * Тип на имота (къща, апартамент, офис);
      * Квадратура;
      * Брой стаи;
      * Локация (Изгрев, Славейков, Зорница, Лазур, Меден Рудник)

Филтрацията става автоматично като използваме AJAX:

1. Какво е AJAX? AJAX (Asynchronous JavaScript and XML) е технология, която позволява на уеб приложенията да комуникират със сървъра асинхронно, без да се налага презареждане на цялата уеб страница. Това прави уебсайтовете по-бързи, динамични и интерактивни.
2. Как работи AJAX (Стъпка по стъпка): AJAX използва обекта XMLHttpRequest (или модерния fetch() API), за да изпраща и получава данни между клиента и сървъра:
   1. **Стъпка 1: Действие от потребителя:**
      1. Потребителят извършва действие на уеб страницата (например филтриране на данни в този случай.)
      2. JavaScript засича действието и инициира AJAX заявка.
   2. **Стъпка 2: Създаване на AJAX заявка:**
      1. JavaScript създава XMLHttpRequest или fetch() заявка, която ще бъде изпратена към сървъра.
      2. Определя се методът на заявката (GET, POST, PUT, DELETE).
   3. **Стъпка 3: Изпращане на заявката към сървъра:**
      1. Запитването се изпраща към сървъра без презареждане на страницата.
      2. Сървърът обработва заявката и подготвя отговор (може да бъде в JSON, XML или HTML формат).
   4. **Стъпка 4: Получаване и обработка на отговора:**
      1. След като сървърът отговори, JavaScript обработва данните и динамично актуализира уеб страницата.
3. **Предимства на AJAX:**
   1. По-бързо зареждане - само част от страницата се обновява, без да се презарежда цялата страница.
   2. По-добро потребителско изживяване - позволява интерактивни функции като авто-попълване на формуляри, търсене в реално време и динамични списъци.
   3. Намалява натоварването на сървъра - само необходимите данни се изпращат и получават.
   4. Гъвкавост - AJAX - може да се комбинира с всяка бекенд технологии (PHP, .NET, Node.js и др.)
4. **Недостатъци на AJAX:**
   1. Не работи без JavaScript - ако JavaScript е изключен в браузъра, AJAX няма да функционира.
   2. Сложност - изисква повече кодиране и обработка на грешки в сравнение на традицинонния подход.
   3. SEO (Search Engine Optimization) проблеми - търсачките не индексират AJAX съдържанието така добре, освен ако не се използват допълнителни техники.
   4. Крос-домейн ограничения - AJAX заявките към различни домейнси са ограничен поради CORS (Cross-Origin Resource Sharing).

AJAX е мощна технология, която позволява на уеб страниците да бъдат динамични и интерактивни. С използването на fetch(), XMLHttpRequest, или jQuery, можем да изпращаме и получаваме данни асинхронно, което подобрява потребителското изживяване.

1. **Как използвах AJAX в Rental Portal:**
   1. **Къде използвах AJAX?**
      1. Филтриране на имоти без презареждане на сайта:
         * Потребителят избира критерии за филтриране (цена, местоположение, брой стаи и т.н.).
         * Страницата се обновява само със съответните резултати, без да се презарежда цялото съдържание.
   2. **Как го имплементирах?:**
      1. **Филтриране на имоти с AJAX:**
         * За да направя динамично филтриране, използвах JavaScript Fetch API и jQuery AJAX, които изпращат филтрите към контролера в ASP.NET и връщат частичен изглед със съответните резултати.
           1. **FrontEnd(JavaScript Fetch API):**

1. document.addEventListener("DOMContentLoaded", function () {

2. const filterForm = document.getElementById("filterForm");

3.

4. if (filterForm) {

5. filterForm.addEventListener("submit", function (event) {

6. event.preventDefault();

7. const formData = new FormData(filterForm);

8. const searchParams = new URLSearchParams(formData);

9.

10. fetch("/Properties/Filter", {

11. method: "POST",

12. headers: {

13. "X-Requested-With": "XMLHttpRequest",

14. "Content-Type": "application/x-www-form-urlencoded",

15. },

16. body: searchParams.toString(),

17. })

18. .then(response => response.text())

19. .then(html => {

20. document.getElementById("propertyList").innerHTML = html;

21. })

22. .catch(error => console.error("Error:", error));

23. });

24.

25. // Автоматично изпращане на формата при промяна на филтър

26. const filterInputs = filterForm.getElementsByClassName("filter-input");

27. Array.from(filterInputs).forEach(input => {

28. input.addEventListener("input", () => {

29. filterForm.dispatchEvent(new Event("submit"));

30. });

31. });

32. }

33. });  
  
$(document).ready(function () {

34. // Trigger AJAX on filter change

35. $(".filter-input").on("change", function () {

36. var formData = $("#filterForm").serialize();

37.

38. $.ajax({

39. url: "/Properties/Filter",

40. type: "POST",

41. data: formData,

42. success: function (result) {

43. // Update the property list

44. $("#propertyList").html(result);

45. },

46. error: function () {

47. alert("An error occurred while filtering properties.");

48. }

49. });

50. });

51. });

52.

* + - * 1. **Backend(ASP.NET Properties Controller, метод за обработка на заявката):**

1. public async Task<IActionResult> Filter(

2. string searchTitle,

3. List<string> addresses,

4. List<string> propertyTypes,

5. double? minPrice,

6. double? maxPrice,

7. double? minArea,

8. double? maxArea,

9. int? minRooms,

10. bool? isAvailable)

11. {

12. try

13. {

14. // Base query

15. string currentLanguage = HttpContext.Request.Cookies["lang"] ?? "bg";

16. var propertiesQuery = \_context.Properties.AsQueryable();

17.

18. // Filter by Title (depending on language)

19. if (!string.IsNullOrEmpty(searchTitle))

20. {

21. if (currentLanguage == "bg")

22. {

23. propertiesQuery = propertiesQuery.Where(p => p.Title\_bg.Contains(searchTitle));

24. Console.WriteLine($"Filtering by Title\_bg: {searchTitle}");

25. }

26. else

27. {

28. propertiesQuery = propertiesQuery.Where(p => p.Title.Contains(searchTitle));

29. Console.WriteLine($"Filtering by Title: {searchTitle}");

30. }

31. }

32.

33. // Filter by Addresses (depending on language)

34. if (addresses != null && addresses.Any())

35. {

36. if (currentLanguage == "bg")

37. {

38. propertiesQuery = propertiesQuery.Where(p => addresses.Contains(p.Address\_bg));

39. Console.WriteLine($"Filtering by Address\_bg: {string.Join(", ", addresses)}");

40. }

41. else

42. {

43. propertiesQuery = propertiesQuery.Where(p => addresses.Contains(p.Address));

44. Console.WriteLine($"Filtering by Address: {string.Join(", ", addresses)}");

45. }

46. }

47.

48. // Filter by Property Types (depending on language)

49. if (propertyTypes != null && propertyTypes.Any())

50. {

51. if (currentLanguage == "bg")

52. {

53. propertiesQuery = propertiesQuery.Where(p => propertyTypes.Contains(p.PropertyType\_bg));

54. Console.WriteLine($"Filtering by PropertyType\_bg: {string.Join(", ", propertyTypes)}");

55. }

56. else

57. {

58. propertiesQuery = propertiesQuery.Where(p => propertyTypes.Contains(p.PropertyType));

59. Console.WriteLine($"Filtering by PropertyType: {string.Join(", ", propertyTypes)}");

60. }

61. }

62.

63. // Other filters (price, area, rooms, availability)

64. if (minPrice.HasValue)

65. {

66. propertiesQuery = propertiesQuery.Where(p => p.Price >= minPrice.Value);

67. Console.WriteLine($"Filtering by minPrice: {minPrice.Value}");

68. }

69. if (maxPrice.HasValue)

70. {

71. propertiesQuery = propertiesQuery.Where(p => p.Price <= maxPrice.Value);

72. Console.WriteLine($"Filtering by maxPrice: {maxPrice.Value}");

73. }

74. if (minArea.HasValue)

75. {

76. propertiesQuery = propertiesQuery.Where(p => p.Area >= minArea.Value);

77. Console.WriteLine($"Filtering by minArea: {minArea.Value}");

78. }

79. if (maxArea.HasValue)

80. {

81. propertiesQuery = propertiesQuery.Where(p => p.Area <= maxArea.Value);

82. Console.WriteLine($"Filtering by maxArea: {maxArea.Value}");

83. }

84. if (minRooms.HasValue)

85. {

86. propertiesQuery = propertiesQuery.Where(p => p.NumberOfRooms >= minRooms.Value);

87. Console.WriteLine($"Filtering by minRooms: {minRooms.Value}");

88. }

89. if (isAvailable.HasValue)

90. {

91. propertiesQuery = propertiesQuery.Where(p => p.IsAvailable == isAvailable.Value);

92. Console.WriteLine($"Filtering by isAvailable: {isAvailable.Value}");

93. }

94.

95. // Debug: Print out the properties before returning

96. var propertiesList = await propertiesQuery.ToListAsync();

97. Console.WriteLine("Filtered properties:");

98. foreach (var property in propertiesList)

99. {

100. Console.WriteLine($"Property Id: {property.Id}, Address: {property.Address}");

101. }

102.

103. var filteredProperties = await propertiesQuery.ToListAsync();

104. Console.WriteLine($"Filtered properties count: {filteredProperties.Count}");

105.

106. if (Request.Headers["X-Requested-With"] == "XMLHttpRequest")

107. {

108. return PartialView("\_PropertyListPartial", filteredProperties);

109. }

110.

111. return View("Index", filteredProperties);

112. }

113. catch (Exception ex)

114. {

115. Console.WriteLine($"Error during filtering properties: {ex.Message}");

116. return View("Error");

117. }

118. }

119.

* + - * 1. **Частичен изглед(\_PropertyListPartial.cshtml):**

1. @model IEnumerable<RentalPortal.Models.Property>

2. @inject Microsoft.AspNetCore.Http.IhttpContextAccessor HttpContextAccessor

3.

4. @{

5. ViewData["Title"] = "Create Property";

6. string lang = HttpContextAccessor.HttpContext.Request.Cookies["lang"] ?? "bg";

7. var localizationPath = "wwwroot/translations/property/propertyPartialViewPageTranslation.json";

8. var localization = System.IO.File.ReadAllText(localizationPath);

9. var langData = Newtonsoft.Json.JsonConvert.DeserializeObject<dynamic>(localization)[lang];

10. }

11.

12. @foreach (var item in Model)

13. {

14. <div class="property-card">

15. <img src="@item.Photos" alt="Property Image">

16. <h2>@(lang == "bg" ? item.Title\_bg : item.Title)</h2>

17. <p>@(lang == "bg" ? item.Description\_bg : item.Description)</p>

18. <div class="meta-info">

19. <span>@langData["price\_label"]: @item.Price</span>

20. <span>@langData["area\_label"]: @item.Area m²</span>

21. <span>@langData["rooms\_label"]: @item.NumberOfRooms</span>

22. <span>@langData["address\_label"]: @(lang == "bg" ? item.Address\_bg : item.Address)</span>

23. <span>@langData["available\_label"]: @(item.IsAvailable ? langData["yes"] : langData["no"])</span>

24. </div>

25. <div class="actions">

26. @if (User.IsInRole("Admin"))

27. {

28. <a asp-action="Edit" asp-route-id="@item.Id" class="btn btn-primary btn-sm">Edit</a>

29. <a asp-action="Delete" asp-route-id="@item.Id" class="btn btn-danger btn-sm">Delete</a>

30. }

31. <a asp-action="Details" asp-route-id="@item.Id" class="btn btn-info btn-sm">Details</a>

32. <a asp-action="CreateWithProperty" asp-controller="Appointments" asp-route-propertyId="@item.Id" class="btn btn-success btn-sm">Book Appointment</a>

33.

34. <!-- Favorite Button -->

35. @if (User.Identity.IsAuthenticated)

36. {

37. var favoriteProperties = ViewBag.FavoriteProperties as List<int> ?? new List<int>();

38. bool isFavorited = favoriteProperties.Contains(item.Id);

39. if (isFavorited)

40. {

41. <form method="post" action="@Url.Action("RemoveFromFavorites", "Properties", new { propertyId = item.Id })" class="d-inline">

42. <button type="submit" class="btn btn-warning btn-sm">Remove</button>

43. </form>

44. }

45. else

46. {

47. <form method="post" action="@Url.Action("AddToFavorites", "Properties", new { propertyId = item.Id })" class="d-inline">

48. <button type="submit" class="btn btn-outline-warning btn-sm">Favorites</button>

49. </form>

50. }

51. }

52. </div>

53. </div>

54. }

55.

* 1. **Какви проблеми срещнах и как ги реших?**
     1. **Проблем 1:** AJAX заявките първоначално не работеха, защото забравих да добавя data-ajax="true" в някои форми.
        + Решение: Дебъгнах кода с Chrome DevTools и поправих грешката.
        + Пример:  
          <form id="filterForm" class="filter-form" method="get" **data-ajax="true">**   
          В този фрагмент от код нямаше „data-ajax=”true”“.
     2. **Проблем 2:** Филтрирането не работеше, защото не връщах PartialView в контролера.
        + Решение: Уверих се, че return PartialView("\_PropertyListPartial", filteredProperties); връща само нужния HTML.
  2. **Заключение:** Използването на AJAX в Rental Portal подобри значително скоростта и потребителското изживяване. Вместо тежки презареждания, AJAX позволява динамично взаимодействие с уебсайта, което го прави по-модерен, бърз и ефективен.

1. **Подобряване на потребителското изживяване:**
2. **Документация и представяне:**

2.1. Дизайн и навигация:

* + - За да се осигури удобно потребителско изживяване, интерфейсът трябва да бъде интуитивен, адаптивен и достъпен както от настолни компютри, така и от мобилни устройства. Основните секции на сайта ще включват:
      * Начална страница с навигация на другите страници.
      * Страница за разглеждане на обяви с възможност за филтриране по цена, местоположение, тип на имота, брой стаи и др.
      * Детайлна страница на даден имот, съдържаща галерия със снимки, описание, карта с локацията (реализиран с Google Maps API на JavaScript.) и бутон за заявка на оглед.

2.2. Регистрация, вход и управление на данни:

* + - Регистрацията ще изисква въвеждането на име, имейл и парола. Като паролата ще се хешире автоматично. Хеширането е еднопосочна математическа функция, която превръща данните в низ от неописуем текст, който не може да бъде обърнат или декодиран. Тези пароли и другите свойства се запазват в таблицата **User** на базата данни.
    - Потребителите ще могат да актуализират информацията си, да настройват известията и да се запазват любими имоти и да ги виждат там.

2.3. Имейл услуги:

* + - Системата ще изпраша автоматични имейли за:
      * Напомняне на предстоящи изгледи на обява - Потребителите ще получават автоматични напомняния за предстоящите им огледи на избраните имоти. Това ще включва детайли като дата, час, адрес на имота и контактна информация на агента или собственика.
      * Промени в цени или други свойства на имот - Ако потребителят е добавил даден имот в списъка си с любими, той ще получава известие при всяка промяна в цената, наличността или други ключови характеристики на имота. Това ще помогне на потребителите да следят актуалните условия на пазара в реално време.
      * Потвърждение на изглед на изглед която сме направили, това става от администратора - След като потребител подаде заявка за оглед на даден имот, администраторът на сайта или собственикът на имота ще трябва да потвърдят заявката. Потребителят ще получи известие за одобрението или отказа, заедно с допълнителни инструкции относно уговорения оглед.

**Възможни технологии и инструменти:**

* **ASP.NET** за изграждане на сървърната част.
* **Entity Framework Core и Microsoft SQL Server Management 18** за работа с база данни.
* **JavaScript, HTML, CSS** за изграждане на фронтенд (frontend) част.
* **Bootstrap** за адаптивен дизайн.
* **XUnit** или **NUnit** за компонентни тестове.

Този проект ще предостави функционален уеб сайт, който отговаря на нуждите както на потребителите, търсещи жилища или офис пространства, така и на администраторите и агентите, които трябва да управляват обявите и заявки за огледи.

**1.2. Кратко описание на функционалностите на уеб сайта**

Уеб сайтът предоставя платформа за намиране и наемане на жилища и офис сгради, с разнообразни функционалности за потребители и администратори.

### Функционалности за потребители:

1. **Каталог на имоти**: Разглеждане на налични имоти с подробна информация (снимки, локация, цена, площ, описание).
2. **Търсене и филтриране**: Филтриране на имоти по тип, местоположение, цена, площ и други параметри. Сортиране по цена, площ или дата.
3. **Профил на потребителя**: Създаване на личен профил за запазване на любими имоти, проследяване на заявки за огледи и получаване на известия за нови оферти.
4. **Заявки за оглед**: Подаване на заявки за оглед на избрани имоти с посочена предпочитана дата и час.

### Функционалности за администратори и агенти:

1. **Административен панел**: Управление на обявите (добавяне, редактиране, изтриване), преглед на заявки за огледи и генериране на доклади за популярността на имотите.
2. **Управление на огледи**: Потвърждаване и организация на огледи, изпращане на уведомления до потребителите.

### Допълнителни функции:

* **Персонализирани известия**: Получаване на известия за нови имоти, отговарящи на критериите на потребителя.
* **Доклади и статистики**: Генериране на отчети за най-гледаните и най-често запазваните обяви.

Целта на сайта е да осигури лесно търсене и наемане на имоти, както и ефективно управление на обяви и заявки за огледи от страна на администраторите и агентите.

**1.3. Обосновка на избора на тема и технологии:**

### Обосновка на избора на тема:

Изборът на тема за създаване на уеб сайт за намиране и наемане на жилища и офис сгради е мотивиран от актуалността и необходимостта на подобни платформи в съвременното общество. В условията на динамично развиващия се пазар на недвижими имоти и все по-нарастващото значение на онлайн услугите, съществуването на ефективни и удобни платформи за търсене и наемане на имоти е ключово както за потребителите, така и за агентите.

Уеб сайтът ще предложи на потребителите възможност за лесно и удобно търсене на имоти, филтриране по различни критерии и подаване на заявки за огледи, което ще им спести време и усилия. От друга страна, администраторите и агентите ще имат достъп до удобни инструменти за управление на обяви, заявки и отчети, което ще улесни тяхната работа и ще подобри ефективността на бизнеса.

Тази тема е интересна и полезна както от практическа, така и от теоретична гледна точка, тъй като предлага възможност за прилагане на съвременни технологии за изграждане на сложни и мащабируеми уеб приложения, работа с база данни, както и за решаване на реални проблеми, свързани с управлението на недвижими имоти.

### Обосновка на избора на технологии:

1. **ASP.NET**:
   * **Мощност и мащабируемост**: ASP.NET е съвременен, високопроизводителен и мащабируем фреймуърк, който е идеален за разработката на уеб приложения. Той предлага висока производителност и поддръжка на множество платформи (Windows, Linux, macOs), което прави приложението лесно за внедряване и поддръжка.
   * **Сигурност и стабилност**: ASP.NET предоставя вградени механизми за сигурност, като удостоверяване (authentication), авторизация и защита от нападения като CSRF и XSS. Това е важно за защита на чувствителни данни, като потребителски профили и информация за имоти.
   * **Поддръжка за трислойна архитектура**: ASP.NET е отличен избор за създаване на трислойни приложения, което е в съответствие с изискванията на дипломния проект. Технологията предлага лесно разделяне на слоевете за данни, услуги и потребителски интерфейс, което води до по-добра поддръжка и модификация на системата.
2. **Entity Framework**:
   * **Работа с база данни**: Entity Framework е ORM (Object-Relational Mapping) инструмент, който улеснява взаимодействието с база данни, като позволява използването на C# за извършване на SQL операции. Това ще осигури лесно и ефективно съхранение и извличане на данни за имотите, потребителите и заявките за огледи.
   * **Поддръжка на сложни заявки**: Entity Framework предлага мощни възможности за генериране на сложни заявки към базата данни, включително филтриране, сортиране и групиране на данни, което е необходимо за функционалността на търсене и филтриране на имоти.
3. **JavaScript, HTML, CSS (и Bootstrap)**:
   * **Интерактивност и UX**:JavaScript е основната технология за създаване на динамично поведение на сайта, което е важно за функционалности като търсене в реално време, сортиране на резултати, заявки за огледи и т.н.
   * **Адаптивност**: С помощта на HTML и CSS, заедно с CSS фреймуъркове (framework)като **Bootstrap**, може да се създаде адаптивен дизайн, който да осигури добро потребителско изживяване както на настолни компютри, така и на мобилни устройства.
   * **Модерни практики за фронтенд**: Използването на CSS фреймуъркове (framework) **Bootstrap** или ще позволи бързо и ефективно изграждане на отзивчиви и модерни интерфейси, спестявайки време за стилизиране и осигурявайки единен вид на сайта.
4. **Тестови технологии (например XUnit или NUnit)**:
   * **Осигуряване на качество**: Технологиите за компоненти тестове като **XUnit** или **NUnit** ще позволят автоматизирани тестове на основните функционалности на системата (например добавяне на обяви, търсене на имоти, подаване на заявки за огледи). Това ще осигури висока степен на стабилност и надеждност на приложението.
5. **Microsoft SQL Server & SQL Server Management Studio:**
   * **Сигурност и надеждност:** Microsoft SQL Server е доказан релационен базов мениджър за данни, който осигурява висока производителност, мащабируемост и сигурност. Той е подходящ за обработка на големи обеми от данни и поддържа множество механизми за защита на чувствителна информация.
   * **Лесна администрация:** SQL Server Management Studio (SSMS) предоставя удобен графичен интерфейс за управление на базата данни, което улеснява администрирането, мониторинга и оптимизацията на базата данни.
   * **Поддръжка на транзакции и интегритет на данните:** Microsoft SQL Server предлага силна поддръжка на транзакции и механизми за осигуряване на целостта на данните, което е критично за коректното функциониране на системата.

### Заключение:

Избраните технологии (ASP.NET, Entity Framework, JavaScript, HTML, CSS и тестови инструменти) са подходящи за изграждане на стабилно, ефективно и мащабируемо уеб приложение, което отговаря на нуждите както на потребителите, така и на администраторите. Технологиите осигуряват мощност, сигурност и възможности за лесно управление и разширяване на системата в бъдеще, като същевременно поддържат добър потребителски интерфейс и оптимално представяне на сайта.

**2.1. Описание на изискванията към уеб сайта:**

**Описание на изискванията към уеб сайта:**

Уеб сайтът ще бъде платформа за намиране и наемане на жилища и офис сгради, която ще предоставя на потребителите удобен начин за търсене, разглеждане и запазване на имоти, както и за подаване на заявки за огледи. Сайтът ще включва и административен панел за управление на обяви, заявки и генериране на отчети. Изискванията към сайта са както следва:

### 1. Функционални изисквания:

#### 1.1. Каталог на имоти:

* Уеб сайтът трябва да предоставя списък с налични имоти (жилища и офис сгради), като за всеки имот ще се показва основна информация, включително:
  + Снимки на имота.
  + Локация (адрес или регион).
  + Цена.
  + Площ и етажност.
  + Описание на имота.
* Всеки имот трябва да има детайлен изглед, който ще включва допълнителни снимки и подробности за характеристиките му.

#### 1.2. Търсене и филтриране на имоти:

* Потребителите трябва да имат възможност да търсят имоти по различни критерии:
  + Ключови думи (например местоположение, тип имот).
  + Тип на имота (жилище, офис).
  + Местоположение (например регион или град).
  + Ценови диапазон (минимална и максимална цена).
  + Площ (минимум и максимум).
  + Други характеристики като етажност или наличност на паркоместа.
* Резултатите от търсенето трябва да могат да бъдат сортирани по:
  + Цена.
  + Площ.
  + Дата на публикуване.

#### 1.3. Профил на потребителя:

* Потребителите трябва да могат да създават лични профили, които ще включват:
  + Регистрация и логин.
  + Запазване на любими имоти.
  + История на подадените заявки за огледи.
  + Получаване на известия за нови оферти, отговарящи на техните критерии.
  + Персонализиране на предпочитанията за търсене и известия.

#### 1.4. Заявки за оглед:

* Потребителите трябва да могат да подадат заявки за огледи на избрани имоти. При подаването на заявка те трябва да могат да:
  + Посочат предпочитана дата и час за оглед.
  + Получат потвърждение или отказ от страна на агент или администратор.
  + Проследяват статус на заявките (например, дали са потвърдени или отложени).

#### 1.5. Административен панел:

* Администраторите и агентите трябва да имат достъп до административен панел, който позволява:
  + Добавяне на нови обяви за имоти (жилища и офис сгради).
  + Редактиране и премахване на съществуващи обяви.
  + Преглеждане на заявки за огледи и потвърждаване или отказване на тях.
  + Генериране на отчети за популярността на обявите, като най-гледани и най-често запазвани обяви.
  + Проследяване на текущите заявки за огледи и комуникацията с потребителите.

#### 1.6. Известия:

* Платформата трябва да има система за изпращане на известия към потребителите:
  + Известия за нови имоти, отговарящи на зададените критериите за търсене.
  + Известия за потвърдени или отказани заявки за огледи.
  + Известия за напомняне на среща.

### 2. Нефункционални изисквания:

#### 2.1. Производителност:

* Уеб сайтът осигурява бързо зареждане на страниците, като времето за зареждане не трябва да надвишава 3 секунди.
* Търсенето на имоти и филтрирането са бързи и ефективни, дори при голям брой обяви.

#### 2.2. Сигурност:

* Потребителските данни (например пароли) трябва да се съхраняват сигурно, като се използват съвременни технологии за криптиране и хеширане.
* Трябва да се осигури защита от основни уязвимости като SQL инжекции, XSS (Cross-Site Scripting) и CSRF (Cross-Site Request Forgery).
* Платформата трябва да използва HTTPS (HyperText Transfer Protocol) за сигурна комуникация между клиента и сървъра, това става когато стартираме уеб приложението от Visual Studio, тъй като има опция за стартиране с HTTPS.

#### 2.3. Отзивчивост:

* Уеб сайтът трябва да бъде адаптивен и да осигурява добро потребителско изживяване на различни устройства (настолни компютри, таблети и мобилни телефони).
* Дизайнът трябва да бъде съвременен и лесен за навигация, като се използват съвременни CSS фреймуъркове (framework) (например Bootstrap).

#### 2.4. Мащабируемост:

* Системата трябва да бъде проектирана така, че да може лесно да се разширява с нови функционалности или да поддържа повече потребители и обяви при увеличен трафик.

#### 2.5. Достъпност:

* Уеб сайтът трябва да отговаря на основни стандарти за достъпност, така че да бъде лесно използваем и от хора с увреждания, включително поддръжка на скрийн рийдъри (screen reader) и възможности за увеличаване на шрифтове.

#### 2.6. Поддръжка и обновления:

* Системата трябва да бъде лесна за поддръжка и обновление. Трябва да бъде проектирана така, че новите функции и подобрения да могат да се внедряват лесно без да се нарушава работата на сайта.

### 3. Технологични изисквания:

* **Фронтенд**: Уеб сайтът ще използва HTML, CSS и JavaScript за изграждане на потребителския интерфейс. За постигане на адаптивност и стилизиране ще се използват CSS фреймуъркове като **Bootstrap**. Тъй като работим с C#, използваме Razor syntax, Синтаксисът на Razor е синтаксис за маркиране, използван в ASP.NET за вграждане на C# код в HTML, което позволява генериране на динамично съдържание в уеб страници.
* **Сървърна част**: За изграждането на сървърната част ще се използва **ASP.NET Core**. ASP.NET Core е междуплатформена рамка с отворен код, разработена от Microsoft за изграждане на модерни, високопроизводителни уеб приложения и API, предлагаща подобрена мащабируемост, гъвкавост и скорост в сравнение с традиционната ASP.NET рамка.
* **База данни**: За съхранение на данни за имоти, потребители и заявки ще се използва релационна база данни, като **SQL Server**, с помощта на **Entity Framework Core и SQL Server Management Studio**.
* **Тестове**: За тестване на функционалностите на системата ще се използват **XUnit** или **NUnit**.

Тези изисквания осигуряват, че уеб сайтът ще бъде функционален, сигурен и лесен за използване както за потребителите, така и за администраторите.

**2.2. Проектиране на архитектурата:**

Проектирането на архитектурата на уеб сайта ще се базира на трислойния модел, който е широко използван за изграждане на мащабируеми и поддържани приложения. Трислойната архитектура ще включва три основни слоя:

1. **Слой за данни (Data Layer):**  
   Слоят за данни ще отговаря за съхранението и достъпа до данните в приложението. В този слой ще се използва релационна база данни (например SQL Server), като данните ще се съхраняват чрез таблици, които включват информация за имоти, потребители и заявки за огледи.
   * **Технологии:** Microsoft SQL Server, Entity Framework Core.
   * **Функционалности:**
     + Записване и извличане на данни за имоти, потребители и заявки.
     + Защита на данните чрез хеширане на чувствителни информации (като пароли). Хешираната парола се съхранява в таблицата User на базата данни.
2. **Слой за услуги (Business Logic Layer):**  
   Слоят за услуги ще съдържа логиката за обработка на заявките и бизнес процесите, свързани с функционирането на сайта. Този слой ще управлява взаимодействията между слоя за данни и слоя за потребителски интерфейс. Ще бъде отговорен за:
   * Обработката на търсене и филтриране на имоти.
   * Управление на заявки за огледи.
   * Изпращане на известия към потребителите и администраторите.
   * Генериране на статистики и отчети.
   * Осигуряване на безопасност и валидиране на входните данни.
   * **Технологии:** ASP.NET Core, C#.
3. **Слой за потребителски интерфейс (Presentation Layer):**  
   Слоят за потребителски интерфейс ще бъде отговорен за взаимодействието с потребителите и ще предоставя визуалната част на уеб сайта. Този слой ще осигури:
   * Интуитивен и лесен за използване интерфейс за търсене на имоти.
   * Регистрация и логин на потребителите.
   * Подаване на заявки за огледи и проследяване на тяхното състояние.
   * Управление на профила и получаване на известия.
   * Осигуряване на ефективно търсене и филтриране на данни чрез AJAX, което ни позволява да филтрираме имоти, без да е необходимо да презареждаме страницата.
   * Адаптивен дизайн за добро представяне на мобилни устройства и десктоп компютри.
   * **Технологии:** Razor Syntax, CSS, JavaScript, Bootstrap, ASP.NET Core (MVC).

### 2.2.1. ****Трислойна архитектура (Data Layer, Business Logic Layer, Presentation Layer)****

Архитектурата ще бъде изградена така, че всеки слой да бъде независим, което ще позволи лесна поддръжка и мащабиране на системата. Взаимодействието между слоевете ще се осъществява чрез ясно дефинирани интерфейси и API-та.

#### Пример за взаимодействие между слоевете:

1. **Слой за потребителски интерфейс**:
   * Потребителят извършва търсене на имоти чрез формата за търсене.
   * Изпраща заявка към **слоя за услуги**, който обработва заявката, изпълнява необходимите операции за филтриране (който става автоматично) на резултати и генерира отговор.
2. **Слой за услуги**:
   * Изпраща запитване към **слоя за данни** за извличане на имоти, отговарящи на критериите на потребителя.
   * Връща резултатите обратно в слоя за потребителски интерфейс.
3. **Слой за данни**:
   * Извлича необходимите данни от базата данни и ги предоставя на слоя за услуги. Включват таблиците:

- Users - Потребители  
- UserFavorites – Любими на потребители  
- Appointments - Срещи  
- Properties – Имоти или офиси  
- NotificationsAndPreferences - Известия

Тази архитектура ще осигури възможността за лесна бъдеща експанзия на системата. Например, нови функционалности като интеграция с външни API-та или добавяне на нови услуги ще могат да бъдат добавяни в слоя за услуги без да се променя останалата част от системата.

Използваме Google API за интегриране на функционалността за карти в нашия код. Тя работи по същия начин като JavaScript код, като единственото изискване е API ключ, който е безплатен.

### Предимства на трислойната архитектура:

* **Мащабируемост:** Лесно разширяване на функционалностите и инфраструктурата (например добавяне на нови сървъри за база данни). Ако решим да променим базата данни, ще е необходимо да актуализираме кода, включително моделите и контролерите. В противен случай кодът няма да работи, тъй като базата данни няма да бъде синхронизирана с кода, което означава, че тя няма да бъде актуализирана.
* **Поддръжка и модификации:** Лесна промяна и надграждане на всеки слой поотделно.
* **Сигурност:** Критичните бизнес процеси и логика, които изискват по-добра защита, могат да бъдат изолирани в слой за услуги, недостъпен за директен достъп от потребителите. За осигуряване на сигурността се използва HTTPS, който е интегриран в **Visual Studio** — интегрираната среда за разработка.

Проектирането на такава архитектура е ключово за осигуряване на висока производителност и устойчивост на системата, като същевременно се поддържа доброто потребителско изживяване и ефективността на администрацията.

**2.3. Проектиране на база данни**

**1. Въведение:**

Проектирането на базата данни за проекта има за цел да осигури ефективно управление на потребителите, имотите, резервациите и известията. В тази документация ще бъдат описани основните таблици, техните атрибути, релации и бизнес логиката, която поддържат.

**2. Структура на базата данни:**

В базата данни има следните основни таблици:

* **User** – съхранява информация на потребителите
* **Property** – съдържа данни за прегледаните имоти
* **Appointment** – управлява заявките за огледи (срещи)
* **UserFavorite** – позволява на потребителите да запазват любими имоти
* **NotificationsAndPreferences** – управлява известията и предпочитанията на потребителите.

**3. Описание на таблиците:**

**3.1. User**

Таблицата съхранява информация за всички регистрирани потребители.

**Структура:**

|  |  |  |
| --- | --- | --- |
| Колона | Тип | Описание |
| Id | INT (PK, Identity) | Уникален идентификатор на потребителя (първичен ключ) |
| Name | NVARCHAR(100) | Име на потребителя |
| Email | NVARCHAR(255) | Уникален емайл адрес на потребителя |
| PasswordHash | NVARCHAR(255) | Хеширана парола |
| Role | NVARCHAR(50) | Роля на потребителя (Admin, User) |
| CreatedAt | DATETIME | Дата на регистрация |
| IsEmailConfirmed | BIT | Индикация дали имейлът е потвърден |
| ResetToken | NVARCHAR(255) | Токен за нулиране на паролата |
| ResetTokenExpiration | DATETIME | Изтичане на валидността на токена за нулиране на паролата |

**SQL заявка за създаване на таблицата:**

1. CREATE TABLE User (

2. Id IDENTITY(1,1) PRIMARY KEY,

3. Name NVARCHAR(100) NOT NULL,

4. Email NVARCHAR(255) UNIQUE NOT NULL,

5. PasswordHash NVARCHAR(255) NOT NULL,

6. Role NVARCHAR(50) NOT NULL,

7. CreatedAt DATETIME DEFAULT GETDATE(),

8. IsEmailConfirmed BIT DEFAULT 0,

9. ResetToken NVARCHAR(255),

10. ResetTokenExpiration DATETIME

11. );

12.

**3.2. Property:**

Съхранява информация за предлаганите имоти.

**Структура:**

|  |  |  |
| --- | --- | --- |
| Колона | Тип | Описание |
| Id | INT (PK, Identity) | Уникален идентификатор на имотите (първичен ключ) |
| Title | NVARCHAR(200) | Заглавие на обявата, офиса |
| Description | NVARCHAR(MAX) | Описание на имота |
| Address | NVARCHAR(255) | Адрес на имота |
| PropertyType | NVARCHAR(255) | Вид имот (Апартамент, къща, офис) |
| Title\_bg | NVARCHAR(200) | Заглавие на обявата, офиса на български |
| Description\_bg | NVARCHAR(MAX) | Описание на имота на български |
| Address\_bg | NVARCHAR(255) | Адрес на имота на български |
| PropertyType\_bg | NVARCHAR(255) | Вид имот (Апартамент, къща, офис) на български |
| Price | DECIMAL(18,2) | Цена на имот |
| Area | INT | Квадратура в м² |
| Floor | INT | Етаж на имота (ако е приложимо) |
| IsAvailable | BIT | Статус на имот, дали е наличен или не |
| NumberOfRooms | INT | Брой стаи на даден имот |
| Photos | NVARCHAR(MAX) | Снимки на имота, използва се URL на дадена снимка |
| UpdatedAt | DATETIME | Дата на последна актуализация на имота |
| Latitude | DECIMAL(9,6) | Географка ширина на имота (къде се намира) |
| Longitude | DECIMAL(9,6) | Географска дължина на имота |
| Rating | FLOAT | Рейтинг на имота, оценка |

**SQL заявка за създаване на таблицата:**

1. CREATE TABLE Properties (

2. Id INT PRIMARY KEY IDENTITY(1,1),

3. Title NVARCHAR(200),

4. Description NVARCHAR(MAX),

5. Address NVARCHAR(255),

6. PropertyType NVARCHAR(255),

7. Title\_bg NVARCHAR(200),

8. Description\_bg NVARCHAR(MAX),

9. Address\_bg NVARCHAR(255),

10. PropertyType\_bg NVARCHAR(255),

11. Price DECIMAL(18,2),

12. Area INT,

13. Floor INT,

14. IsAvailable BIT,

15. NumberOfRooms INT,

16. Photos NVARCHAR(MAX),

17. UpdatedAt DATETIME,

18. Latitude DECIMAL(9,6),

19. Longitude DECIMAL(9,6),

20. Rating FLOAT

21. );

**3.3. UserFavorite:**

Позволява на потребителите да добавят имоти към “любими”.

**Структура:**

|  |  |  |
| --- | --- | --- |
| Колона | Тип | Описание |
| UserId | INT (FK) | Идентификатор на потребителя (външен ключ към таблицата "User") |
| PropertyId | INT(FK) | Идентификатор на имота (външен ключ към таблицата "Property") |
| CreatedAt | DATETIME | Дата на създаване на записа |

В тази таблица използваме външни ключове (Foreign Key) за навигация между User и Property, затова първичните ключове са UserId и PropertyId от другите таблици.

**SQL заявка за създаване на таблицата UserFavorite:**

1. CREATE TABLE UserProperty (

2. UserId INT,

3. PropertyId INT,

4. CreatedAt DATETIME,

5. PRIMARY KEY (UserId, PropertyId),

6. FOREIGN KEY (UserId) REFERENCES User(UserId),

7. FOREIGN KEY (PropertyId) REFERENCES Property(PropertyId)

8. );

**3.4. Appointments:**

Управлява заявките за огледи на даден имот.

Структура:

|  |  |  |
| --- | --- | --- |
| Колона | Тип | Описание |
| Id | INT(PK) | Уникален идентификатор на заявките за огледи (първичен ключ) |
| UserId | INT(FK) | Външен ключ към ID-то към потребителя |
| PropertyId | INT(FK) | Външен ключ към ID-то към имота |
| RequestedAt | DATETIME | Дата и час на огледа |
| IsConfirmed | BIT | Потвърден статус на заявката за оглед (true,false) |

**SQL заявка за създаване на таблицата Appointments (заявките за огледи):**

1. CREATE TABLE Appointment (

2. Id INT IDENTITY(1,1) PRIMARY KEY, -- Using INT as primary key with auto-increment

3. UserId UNIQUEIDENTIFIER NOT NULL,

4. PropertyId UNIQUEIDENTIFIER NOT NULL,

5. RequestedAt DATETIME DEFAULT GETDATE(),

6. IsConfirmed BIT DEFAULT 0,

7. FOREIGN KEY (UserId) REFERENCES [User](Id) ON DELETE CASCADE,

8. FOREIGN KEY (PropertyId) REFERENCES [Property](Id) ON DELETE CASCADE

9. );

**3.5. NotificationsAndPreferences:**

**4. Заключение:**

Съхранява предпочитанията и известията на потребителите

**Структура:**

|  |  |  |
| --- | --- | --- |
| Колона | Тип | Описание |
| PreferenceId | INT(PK) | Уникален идентификатор на заявките за предпочитанията (първичен ключ) |
| UserId | INT(FK) | Външен ключ на потребителя |
| NotificationType | INTB(FK) | Вид на известията |
| IsEnabled | BIT | Дали е активирано или деактивирано известията |
| CreatedAt | DATETIME | Дата на създаване |

**SQL заявка за създаване на таблицата NotificationsAndPreferences:**

1. CREATE TABLE NotificationsAndPreferences (

2. PreferenceId INT IDENTITY(1,1) PRIMARY KEY,

3. UserId UNIQUEIDENTIFIER NOT NULL,

4. NotificationType NVARCHAR(100) NOT NULL,

5. IsEnabled BIT DEFAULT 1,

6. CreatedAt DATETIME DEFAULT GETDATE(),

7. FOREIGN KEY (UserId) REFERENCES [User](Id) ON DELETE CASCADE

8. );

1. Заключение:

Тази структура осигурява гъвкавост и ефективност при работа с дипломния проект Rental Portal, като гарантира нормализирани и добре свързани данни.

**2.4. Разработка на основни функции на системата:**

**В този раздел ще разгледаме основните функционалности на системата, които ще бъдат разработени за уеб сайта. Тези функции са проектирани да отговарят на изискванията на потребителите и администраторите, като осигуряват лесно и ефективно управление на имоти, заявки за огледи и потребителски профили.**

**2.4.1. Каталог на имоти:**

* **Описание:**
  + Каталогът на имоти е основната функционалност на уеб сайта, която позволява на потребителите да разглеждат наличните имоти. Всеки имот ще бъде представен с подробна информация, включително снимки, описание, локация, цена и други характеристики.
* **Функционалности:**
  + **Преглед на имоти:** Потребителите ще могат да разглеждат списък с налични имоти, като всеки имот ще има своя страница с подробна информация.
  + **Детайлен изглед:** При кликване върху конкретен имот, потребителят ще бъде пренасочен към страница с подробна информация за имота, включително снимки, описание, цена, локация и други характеристики.
  + **Снимки и галерия:** Всеки имот ще има галерия със снимки, които ще бъдат показвани в слайдър или като миниатюри.
  + **Интерактивна карта:** Ще бъде интегрирана карта (например Google Maps), която показва точната локация на имота.
* **Технологии:**
  + **HTML, CSS, JavaScript** за визуализация на имотите.
  + **ASP.NET Core** за извличане на данните от базата данни и подаването им към потребителския интерфейс.
  + **Google Maps API** за показване на локацията на имотите.

**2.4.2. Търсене и филтриране на имоти:**

* **Описание:**
  + Функционалността за търсене и филтриране на имоти позволява на потребителите да намират имоти, които отговарят на техните критерии. Потребителите ще могат да филтрират имоти по различни параметри, като тип имот, локация, цена, площ и други.
* **Функционалности:**
  + **Филтриране по параметри:** Потребителите ще могат да филтрират имоти по тип (офис, апартамент, къща), локация, ценови диапазон, площ, брой стаи и други характеристики.
  + **Сортиране на резултати:** Резултатите от търсенето ще могат да бъдат сортирани по цена, площ, и други параметри.
  + **Търсене в реално време:** Филтрирането и сортирането ще се извършват в реално време, без да е необходимо презареждане на страницата, благодарение на използването на AJAX.
* **Технологии:**
  + **JavaScript** за реализация на търсенето и филтрирането в реално време.
  + **ASP.NET Core** за обработка на заявките за търсене и връщане на резултатите.
  + **Entity Framework Core** за извличане на данните от базата данни.

**2.4.3. Профил на потребителя**

* **Описание:**
  + Профилът на потребителя позволява на регистрираните потребители да управляват своите данни, да запазват любими имоти и да проследяват своите заявки за огледи.
* **Функционалности:**
  + **Регистрация и вход:** Потребителите ще могат да се регистрират и да влизат в системата чрез имейл и парола.
  + **Управление на профила:** Потребителите ще могат да редактират своя профил, като променят личните си данни, парола и предпочитания.
  + **Любими имоти:** Потребителите ще могат да запазват имоти в списък с любими, за да ги разгледат по-късно.
  + **История на заявките:** Потребителите ще могат да проследяват своите заявки за огледи и да виждат техния статус (потвърдени, отложени, отказани).
* **Технологии:**
  + **ASP.NET Core Identity** за управление на потребителските профили и автентикация.
  + **Entity Framework Core** за съхранение и извличане на потребителските данни.
  + **JavaScript** за подобряване на потребителското изживяване при управление на профила.

2.4.4. Заявки за огледи:

* **Описание:**
  + Функционалността за заявки за огледи позволява на потребителите да подават заявки за огледи на избрани имоти. Администраторите и агентите ще могат да управляват тези заявки и да ги потвърдят или отказват.
* **Функционалности:**
  + **Подаване на заявка**: Потребителите ще могат да подават заявки за огледи на избрани имоти, като посочват предпочитана дата и час.
  + **Потвърждение на заявка**: Администраторите и агентите ще могат да преглеждат и потвърдят заявките за огледи, като изпращат известие до потребителя.
  + **Проследяване на статуса:** Потребителите ще могат да проследяват статуса на своите заявки (потвърдени, отложени, отказани).
* **Технологии:**
  + **ASP.NET Core** за обработка на заявките и изпращане на известия.
  + **Entity Framework Core** за съхранение и управление на заявките за огледи.
  + **JavaScript** за подобряване на потребителското изживяване при подаване на заявки.

**2.4.5. Административен панел:**

* **Описание:**
  + Административният панел е предназначен за администраторите и агентите, които управляват обявите за имоти и заявките за огледи. Той предоставя инструменти за добавяне, редактиране и изтриване на обяви, както и за управление на заявките за огледи.
* **Функционалности:**
  + **Управление на обяви:** Администраторите и агентите ще могат да добавят, редактират и изтриват обяви за имоти.
  + **Управление на заявки:** Администраторите ще могат да преглеждат и управляват заявките за огледи, като ги потвърдят или отказват.
  + **Генериране на отчети:** Администраторите ще могат да генерират отчети за популярността на обявите, като най-гледани и най-често запазвани имоти.
* **Технологии:**
  + **ASP.NET Core** за изграждане на административния панел.
  + **Entity Framework Core** за управление на данните за обяви и заявки.
  + **JavaScript** за подобряване на потребителското изживяване в административния панел.

**2.5. Технологии и инструменти за реализация**

* В този раздел ще разглеждаме технологиите и инструментите, който ще бъдат използвани за реализация на проекта:

**2.5.1. Избор на технологии**

* **ASP.NET Core:**
  + **Сървърна част:** ASP.NET Core е избрана за сървърната част на приложението, тъй като предлага висока производителност, мащабируемост и сигурност. Той е подходящ за изграждане на трислойни приложения и предлага вградени механизми за автентикация и авторизация.
* **Entity Framework Core:** 
  + **База данни:** За работа с базата данни е избран Entity Framework Core, който действа като ORM (Object-Relational Mapping) и улеснява взаимодействието с релационните бази данни. Той предоставя удобен начин за управление на данните чрез модели, като позволява изпълнение на заявки, съхранение и извличане на информация без директна работа със SQL.
* **Microsoft SQL Server/SQL Server Management Studio**
  + За създаването, управлението и съхранението на данни в базата данни използвах Microsoft SQL Server, който осигурява надеждна и мащабируема среда за работа с релационни бази данни. Чрез SQL Server Management Studio (SSMS) извърших структуриране на базата данни, създаване на таблици, индекси и връзки между тях, както и изпълнение на сложни SQL заявки за ефективно управление на информацията. Комбинирането на тези технологии гарантира стабилна и производителна система за управление на данните в платформата.
* **JavaScript, HTML, CSS (Razor syntax):**
  + **Фронтенд:** JavaScript, HTML и CSS са избрани за изграждане на потребителския интерфейс. JavaScript ще се използва за добавяне на интерактивност и динамично поведение на сайта, докато HTML и CSS ще се използват за структуриране и стилизиране на съдържанието.
* **Bootstrap:**
  + **Адаптивен дизайн:** Bootstrap е избран за създаване на адаптивен дизайн, който ще осигури добро потребителско изживяване на различни устройства, включително настолни компютри, таблети и мобилни телефони.

**2.5.2. Описание на използваните библиотеки и фреймуърци:**

* **ASP.NET Core:**
  + **MVC (Model-View-Controller):** ASP.NET Core MVC ще се използва за изграждане на архитектурата на приложението. Той позволява разделяне на логиката на приложението на три слоя: модел (данни), изглед (потребителски интерфейс) и контролер (бизнес логика).
* **Entity Framework Core:**
  + **ORM (Object-Relational Mapping):** Entity Framework Core ще се използва за управление на базата данни. Той позволява използването на C# за извършване на SQL операции, което улеснява работата с данни.
* **Bootstrap:**
  + **CSS фреймуърк:** Bootstrap ще се използва за създаване на адаптивен и модерен дизайн на сайта. Той предоставя готови компоненти и стилове, които улесняват разработката на потребителския интерфейс.
* **JavaScript библиотеки:**
  + **jQuery**: jQuery ще се използва за опростяване на работата с DOM (Document Object Model) и добавяне на интерактивност към сайта.
  + **AJAX:** AJAX ще се използва за извършване на асинхронни заявки към сървъра, което ще позволи на потребителите да филтрират и търсят имоти без презареждане на страницата.

**2.6. Тестване на основните функционалности:**

**Описание на тестовете за PropertiesController**

Тестовете за реализирани с цел проверка на създаване на имоти и коректността на филтриране по всички критерии. Използва се InMemory база данни за изолиране на тестовете и Moq (Mocking) за симулиране на услуги (напр. IEmailService, интерфейс който съдържа методите за изпращане на имейл.)

1. Филтриране по заглавие (Filter\_ByTitle\_ReturnsCorrectProperties):
   * Цел: Проверка дали филтърът връща имоти със зададено заглавие.
   * Стъпки:
     + Добавят се 3 имота със заглавия: “Home”, “Apartment”, “Villa”.
     + Филтрира се по заглавие “Home”.
   * Очакван резултат: Връща се точно 1 имот със заглавие “Home”.

Ето е кодът на теста Filter\_ByTitle\_ReturnsCorrectProperties:

1. [Fact]

2. public async Task Filter\_ByTitle\_ReturnsCorrectProperties()

3. {

4. // Arrange

5. using (var context = new AppDbContext(\_dbContextOptions))

6. {

7. context.Properties.AddRange(new List<Property>

8. {

9. new Property { Id = 1, Title = "Home", Address = "Address 1", Description = "Description 1", Photos = "Photos 1", PropertyType = "Residential" },

10. new Property { Id = 2, Title = "Apartment", Address = "Address 2", Description = "Description 2", Photos = "Photos 2", PropertyType = "Residential" },

11. new Property { Id = 3, Title = "Villa", Address = "Address 3", Description = "Description 3", Photos = "Photos 3", PropertyType = "Residential" }

12. });

13. await context.SaveChangesAsync();

14.

15. var mockEmailService = new Mock<IEmailService>();

16. var controller = new PropertiesController(context, mockEmailService.Object);

17. controller.ControllerContext.HttpContext = new DefaultHttpContext();

18. controller.ControllerContext.HttpContext.Request.Headers["Cookie"] = "lang=en";

19.

20. // Act

21. var result = await controller.Filter("Home", null, null, null, null, null, null, null, null);

22.

23. // Assert

24. var viewResult = Assert.IsType<ViewResult>(result);

25. var properties = Assert.IsAssignableFrom<IEnumerable<Property>>(viewResult.Model);

26. Assert.Single(properties);

27. Assert.Equal("Home", properties.First().Title);

28. }

29. }

30.

1. Филтриране по местоположение (Filter\_ByLocation\_ReturnsCorrectProperties):
   * **Цел:** Проверка на филтъра по адрес.
   * **Стъпки:** 
     + Добавят се 3 имота с различни адреси.
     + Филтрира се по адрес "Test Address 1".
   * **Очакван резултат:** Връща се 1 имот с точния адрес.

Ето е кодът на теста Filter\_ByLocation\_ReturnsCorrectProperties:

1. [Fact]

2. public async Task Filter\_ByLocation\_ReturnsCorrectProperties()

3. {

4. // Arrange

5. using (var context = new AppDbContext(\_dbContextOptions))

6. {

7. // Изчистете базата данни, за да не се оставят данни

8. context.Properties.RemoveRange(context.Properties);

9. await context.SaveChangesAsync();

10.

11. // Добавете данни за тест

12. context.Properties.AddRange(new List<Property>

13. {

14. new Property

15. {

16. Id = 1,

17. Title = "Home",

18. Price = 1000,

19. Area = 60,

20. IsAvailable = true,

21. Address = "Test Address 1",

22. Description = "Nice home",

23. Photos = "home.jpg",

24. PropertyType = "Residential"

25. },

26. new Property

27. {

28. Id = 2,

29. Title = "Apartment",

30. Price = 1200,

31. Area = 70,

32. IsAvailable = true,

33. Address = "Test Address 2",

34. Description = "Spacious apartment",

35. Photos = "apt.jpg",

36. PropertyType = "Residential"

37. },

38. new Property

39. {

40. Id = 3,

41. Title = "Villa",

42. Price = 2000,

43. Area = 150,

44. IsAvailable = false,

45. Address = "Test Address 3",

46. Description = "Luxury villa",

47. Photos = "villa.jpg",

48. PropertyType = "Residential"

49. }

50. });

51. await context.SaveChangesAsync();

52.

53. // Услуга за електронна поща (за целите на тестване)

54. var mockEmailService = new Mock<IEmailService>();

55. var controller = new PropertiesController(context, mockEmailService.Object);

56.

57. // Задайте език на английски, преди да извикате Filter метода

58. controller.ControllerContext.HttpContext = new DefaultHttpContext();

59. controller.ControllerContext.HttpContext.Request.Headers["Cookie"] = "lang=en"; // Гарантиране на английски контекст

60.

61. // Акт: Филтър за имоти, разположени на "Тестов адрес 1"

62. var result = await controller.Filter(null, new List<string> { "Test Address 1" }, null, null, null, null, null, null, null);

63.

64. // Assert

65. var viewResult = Assert.IsType<ViewResult>(result);

66. var properties = Assert.IsAssignableFrom<IEnumerable<Property>>(viewResult.Model);

67. Assert.Single(properties);

68. Assert.Equal("Test Address 1", properties.First().Address);

69. }

70. }

71.

1. Филтриране по площ (Filter\_ByArea\_ReturnsCorrectProperties):
   * **Цел:** Проверка на филтъра за диапазон площ (60–100 кв.м).
   * **Стъпки:**

* Добавят се 3 имота с площи 60, 70, 150 кв.м.
* Прилага се филтър за площ между 60 и 100.
  + **Очакван резултат:** Връщат се 2 имота (60 и 70 кв.м).

Ето е кодът на теста Filter\_ByArea\_Returns\_CorrectProperties:

1. [Fact]

2. public async Task Filter\_ByArea\_ReturnsCorrectProperties()

3. {

4. // Подготовка

5. using (var context = new AppDbContext(\_dbContextOptions))

6. {

7. // Изчистване на базата данни, за да се избегнат остатъчни данни

8. context.Properties.RemoveRange(context.Properties); // Необходимо за избягване на нарушение на уникалното ограничение - изтриване на всички имоти преди добавяне на нови

9. context.SaveChanges();

10.

11. // Добавяне на тестови данни

12. context.Properties.AddRange(new List<Property>

13. {

14. new Property

15. {

16. Id = 1,

17. Title = "Home",

18. Price = 1000,

19. Area = 60,

20. IsAvailable = true,

21. Address = "Test Address 1",

22. Description = "Nice home",

23. Photos = "home.jpg",

24. PropertyType = "Residential"

25. },

26. new Property

27. {

28. Id = 2,

29. Title = "Apartment",

30. Price = 1200,

31. Area = 70,

32. IsAvailable = true,

33. Address = "Test Address 2",

34. Description = "Spacious apartment",

35. Photos = "apt.jpg",

36. PropertyType = "residential"

37. },

38. new Property

39. {

40. Id = 3,

41. Title = "Villa",

42. Price = 2000,

43. Area = 150,

44. IsAvailable = false,

45. Address = "Test Address 3",

46. Description = "Luxury villa",

47. Photos = "villa.jpg",

48. PropertyType = "residential"

49. }

50. });

51. context.SaveChanges(); // Запазване на промените в контекста

52.

53. // Мокване на имейл услугата (за тестови цели)

54. var mockEmailService = new Mock<IEmailService>();

55. var controller = new PropertiesController(context, mockEmailService.Object);

56.

57. // Задаване на езика на английски за теста

58. controller.ControllerContext.HttpContext = new DefaultHttpContext();

59. controller.ControllerContext.HttpContext.Request.Headers["Cookie"] = "lang=en"; // Осигуряване на английски контекст

60.

61. // Действие: Филтриране на имоти с площ между 60 и 100

62. var result = await controller.Filter(null, null, null, null, null, 60, 100, null, null);

63.

64. // Проверка

65. var viewResult = Assert.IsType<ViewResult>(result);

66. var properties = Assert.IsAssignableFrom<IEnumerable<Property>>(viewResult.Model);

67. Assert.Equal(2, properties.Count());

68. Assert.True(properties.All(p => p.Area >= 60 && p.Area <= 100));

69. }

70. }

71.

1. Филтриране по наличност (Filter\_ByAvailability\_ReturnsCorrectProperties):
   * **Цел:** Проверка дали се връщат само налични имоти.
   * **Стъпки:**
   * Добавят се 2 налични и 1 неналичен имот.
   * Филтрира се по наличност (IsAvailable = true).
   * **Очакван резултат:** Връщат се 2 налични имота.

Ето е кодът на теста Filter\_ByAvailability\_ReturnsCorrectProperties:

1. [Fact]

2. public async Task Filter\_ByAvailability\_ReturnsCorrectProperties()

3. {

4. // Подготовка

5. using (var context = new AppDbContext(\_dbContextOptions))

6. {

7. // Добавяне на тестови данни

8. context.Properties.AddRange(new List<Property>

9. {

10. new Property

11. {

12. Id = 1,

13. Title = "Home",

14. Price = 1000,

15. Area = 60,

16. IsAvailable = true,

17. Address = "Test Address 1",

18. Description = "Nice home",

19. Photos = "home.jpg",

20. PropertyType = "Residential"

21. },

22. new Property

23. {

24. Id = 2,

25. Title = "Apartment",

26. Price = 1200,

27. Area = 70,

28. IsAvailable = false,

29. Address = "Test Address 2",

30. Description = "Spacious apartment",

31. Photos = "apt.jpg",

32. PropertyType = "Residential"

33. },

34. new Property

35. {

36. Id = 3,

37. Title = "Villa",

38. Price = 2000,

39. Area = 150,

40. IsAvailable = true,

41. Address = "Test Address 3",

42. Description = "Luxury villa",

43. Photos = "villa.jpg",

44. PropertyType = "Residential"

45. }

46. });

47. context.SaveChanges(); // Запазване на промените в контекста

48.

49. // Мокване на имейл услугата (за тестови цели)

50. var mockEmailService = new Mock<IEmailService>();

51. var controller = new PropertiesController(context, mockEmailService.Object);

52.

53. // Задаване на езика на английски за теста

54. controller.ControllerContext.HttpContext = new DefaultHttpContext();

55. controller.ControllerContext.HttpContext.Request.Headers["Cookie"] = "lang=en";

56.

57. // Действие: Филтриране на налични имоти (предаване на null за minRooms и използване на bool за isAvailable)

58. var result = await controller.Filter(

59. null, // searchTitle

60. null, // addresses

61. null, // propertyTypes

62. null, // minPrice

63. null, // maxPrice

64. null, // minArea

65. null, // maxArea

66. null, // minRooms

67. true // isAvailable (bool?)

68. );

69.

70. // Проверка

71. var viewResult = Assert.IsType<ViewResult>(result);

72. var properties = Assert.IsAssignableFrom<IEnumerable<Property>>(viewResult.Model);

73. Assert.Equal(2, properties.Count());

74. Assert.True(properties.All(p => p.IsAvailable));

75. }

76. }

77.

1. Филтриране по тип имот (Filter\_ByPropertyType\_ReturnsCorrectProperties):
   * **Цел:** Проверка на филтъра за тип имот (напр. "Residential").
   * **Стъпки:**
   * Добавят се 2 жилищни и 1 комерсиален имот.
   * Филтрира се по тип "Residential".
   * **Очакван резултат:** Връщат се 2 жилищни имота.

1. [Fact]

2. public async Task Filter\_ByPropertyType\_ReturnsCorrectProperties()

3. {

4. // Arrange

5. using (var context = new AppDbContext(\_dbContextOptions))

6. {

7. context.Properties.AddRange(new List<Property>

8. {

9. new Property { Id = 1, Title = "Home", PropertyType = "Residential", Address = "Address 1", Description = "Description 1", Photos = "Photos 1" },

10. new Property { Id = 2, Title = "Apartment", PropertyType = "Residential", Address = "Address 2", Description = "Description 2", Photos = "Photos 2" },

11. new Property { Id = 3, Title = "Office", PropertyType = "Commercial", Address = "Address 3", Description = "Description 3", Photos = "Photos 3" }

12. });

13. await context.SaveChangesAsync();

14.

15. var mockEmailService = new Mock<IEmailService>();

16. var controller = new PropertiesController(context, mockEmailService.Object);

17. controller.ControllerContext.HttpContext = new DefaultHttpContext();

18. controller.ControllerContext.HttpContext.Request.Headers["Cookie"] = "lang=en";

19.

20. // Act

21. var result = await controller.Filter(null, null, new List<string> { "Residential" }, null, null, null, null, null, null);

22.

23. // Assert

24. var viewResult = Assert.IsType<ViewResult>(result);

25. var properties = Assert.IsAssignableFrom<IEnumerable<Property>>(viewResult.Model);

26. Assert.Equal(2, properties.Count());

27. Assert.All(properties, p => Assert.Equal("Residential", p.PropertyType));

28. }

29. }

30.

1. Филтриране по ценови диапазон: (Filter\_ByPriceRange\_ReturnsCorrectProperties):
   * **Цел:** Проверка на филтъра за цена (1000–1500).
   * **Стъпки:**
   * Добавят се 3 имота с цени 1000, 1200, 2000.
   * Прилага се филтър за цена в диапазона.
   * **Очакван резултат:** Връщат се 2 имота (1000 и 1200).

1. [Fact]

2. public async Task Filter\_ByPriceRange\_ReturnsCorrectProperties()

3. {

4. // Arrange

5. using (var context = new AppDbContext(\_dbContextOptions))

6. {

7. context.Properties.AddRange(new List<Property>

8. {

9. new Property { Id = 1, Title = "Home", Price = 1000, Address = "Address 1", Description = "Description 1", Photos = "Photos 1", PropertyType = "Residential" },

10. new Property { Id = 2, Title = "Apartment", Price = 1200, Address = "Address 2", Description = "Description 2", Photos = "Photos 2", PropertyType = "Residential" },

11. new Property { Id = 3, Title = "Villa", Price = 2000, Address = "Address 3", Description = "Description 3", Photos = "Photos 3", PropertyType = "Residential" }

12. });

13. await context.SaveChangesAsync();

14.

15. var mockEmailService = new Mock<IEmailService>();

16. var controller = new PropertiesController(context, mockEmailService.Object);

17. controller.ControllerContext.HttpContext = new DefaultHttpContext();

18. controller.ControllerContext.HttpContext.Request.Headers["Cookie"] = "lang=en";

19.

20. // Act

21. var result = await controller.Filter(null, null, null, 1000, 1500, null, null, null, null);

22.

23. // Assert

24. var viewResult = Assert.IsType<ViewResult>(result);

25. var properties = Assert.IsAssignableFrom<IEnumerable<Property>>(viewResult.Model);

26. Assert.Equal(2, properties.Count());

27. Assert.All(properties, p => Assert.InRange(p.Price, 1000, 1500));

28. }

29. }

30.

1. Филтриране на брой стаи (Filter\_ByNumberOfRooms\_ReturnsCorrectProperties):
   * **Цел:** Проверка на филтъра за минимален брой стаи.
   * **Стъпки:**
   * Добавят се 3 имота с 2, 3, 4 стаи.
   * Филтрира се за минимум 3 стаи.
   * **Очакван резултат:** Връщат се 2 имота (3 и 4 стаи).

1. [Fact]

2. public async Task Filter\_ByNumberOfRooms\_ReturnsCorrectProperties()

3. {

4. // Arrange

5. using (var context = new AppDbContext(\_dbContextOptions))

6. {

7. context.Properties.AddRange(new List<Property>

8. {

9. new Property { Id = 1, Title = "Home", NumberOfRooms = 2, Address = "Address 1", Description = "Description 1", Photos = "Photos 1", PropertyType = "Residential" },

10. new Property { Id = 2, Title = "Apartment", NumberOfRooms = 3, Address = "Address 2", Description = "Description 2", Photos = "Photos 2", PropertyType = "Residential" },

11. new Property { Id = 3, Title = "Villa", NumberOfRooms = 4, Address = "Address 3", Description = "Description 3", Photos = "Photos 3", PropertyType = "Residential" }

12. });

13. await context.SaveChangesAsync();

14.

15. var mockEmailService = new Mock<IEmailService>();

16. var controller = new PropertiesController(context, mockEmailService.Object);

17. controller.ControllerContext.HttpContext = new DefaultHttpContext();

18. controller.ControllerContext.HttpContext.Request.Headers["Cookie"] = "lang=en";

19.

20. // Act

21. var result = await controller.Filter(null, null, null, null, null, null, null, 3, null);

22.

23. // Assert

24. var viewResult = Assert.IsType<ViewResult>(result);

25. var properties = Assert.IsAssignableFrom<IEnumerable<Property>>(viewResult.Model);

26. Assert.Equal(2, properties.Count());

27. Assert.All(properties, p => Assert.True(p.NumberOfRooms >= 3));

28. }

29. }

30.

1. Създаване на нов имот (Create\_NewProperty\_SavesProperty):
   * **Цел:** Проверка дали новият имот се записва коректно в базата.
   * **Стъпки:**
   * Създава се нов имот с предварително зададени параметри.
   * Проверява се дали имотът е добавен успешно.
   * **Очакван резултат:** Имотът се намира в базата със съответните данни.

1. [Fact]

2. public async Task Create\_NewProperty\_SavesProperty()

3. {

4. // Arrange

5. using (var context = new AppDbContext(\_dbContextOptions))

6. {

7. var mockEmailService = new Mock<IEmailService>();

8. var controller = new PropertiesController(context, mockEmailService.Object);

9.

10. // Set language to English before calling Create

11. controller.ControllerContext.HttpContext = new DefaultHttpContext();

12. controller.ControllerContext.HttpContext.Request.Headers["Cookie"] = "lang=en";

13.

14. var property = new Property

15. {

16. Id = 4,

17. Title = "New Property",

18. Price = 3000,

19. Area = 100,

20. IsAvailable = true,

21. Address = "New Address",

22. Description = "Newly added property",

23. Photos = "newproperty.jpg",

24. PropertyType = "Commercial",

25. NumberOfRooms = 3,

26. Longitude = 34.0522m,

27. Latitude = -118.2437m,

28. Rating = 5

29. };

30.

31. // Act

32. var result = await controller.Create(property);

33.

34. // Assert

35. var redirectToActionResult = Assert.IsType<RedirectToActionResult>(result);

36. Assert.Equal(nameof(PropertiesController.Index), redirectToActionResult.ActionName);

37.

38. var savedProperty = await context.Properties.FindAsync(property.Id);

39. Assert.NotNull(savedProperty);

40. Assert.Equal(property.Title, savedProperty.Title);

41. Assert.Equal(property.Address, savedProperty.Address);

42. }

43. }

44.

**Описание на тестовете за AppointmentsController**

Тестовете са реализирани с цел проверка на създаването на заявки за оглед (appointments), валидацията на входните данни и коректната работа на системата при липса на потребителски контекст. Използва се InMemory база данни и Moq за симулиране на услуги (IUserService, IEmailService).

Създаване на ангажимент (GET) – Зареждане на форма с имоти (Create\_Get\_ReturnsViewWithProperties):

**Цел:** Проверка дали формата за създаване на ангажимент зарежда списък с имоти в падащото меню.

**Стъпки:**

Добавя се тестов имот в базата.

Извиква се действието Create() (GET).

**Очакван резултат:** Връща се изглед със списък от имоти, който не е празен.

Ето е кодът на теста Create\_Get\_ReturnsViewWithProperties:

1. [Fact]

2. public async Task Create\_Get\_ReturnsViewWithProperties()

3. {

4. // Arrange

5. using (var context = new AppDbContext(\_dbContextOptions))

6. {

7. // Seed one property so the dropdown isn’t empty

8. context.Properties.Add(new Property

9. {

10. Id = 1,

11. Title = "Test Property",

12. Price = 1000,

13. Area = 50,

14. Address = "123 Test St.",

15. Description = "A test property.",

16. Photos = "photo.jpg",

17. PropertyType = "Residential"

18. });

19. context.SaveChanges();

20.

21. var mockUserService = new Mock<IUserService>();

22. var mockEmailService = new Mock<IEmailService>();

23. var mockLogger = new Mock<ILogger<AppointmentsController>>();

24.

25. var controller = new AppointmentsController(context, mockUserService.Object, mockEmailService.Object, mockLogger.Object);

26.

27. // Act

28. var result = await controller.Create();

29.

30. // Assert

31. var viewResult = Assert.IsType<ViewResult>(result);

32. var viewModel = Assert.IsType<AppointmentCreateViewModel>(viewResult.Model);

33. Assert.NotEmpty(viewModel.Properties);

34. }

35. }

36.

Създаване на ангажимент (POST) – Липса на потребителски имейл (Create\_Post\_NoUserEmail\_RedirectsToLoginRequired):

**Цел:** Проверка дали системата препраща към страница за вход при опит за създаване на ангажимент без потребителски контекст.

**Стъпки:**

Симулира се HTTP заявка без имейл claim в кукитата.

Изпраща се форма с валидни данни за ангажимент.

**Очакван резултат:** Препраща към LoginRequired действие.

Ето е кодът на теста Create\_Post\_NoUserEmail\_RedirectsToLoginRequired:

1. [Fact]

2. public async Task Create\_Post\_NoUserEmail\_RedirectsToLoginRequired()

3. {

4. // Arrange

5. using (var context = new AppDbContext(\_dbContextOptions))

6. {

7. // Създаване на мокове за зависимостите

8. var mockUserService = new Mock<IUserService>();

9. var mockEmailService = new Mock<IEmailService>();

10. var mockLogger = new Mock<ILogger<AppointmentsController>>();

11.

12. var controller = new AppointmentsController(context, mockUserService.Object, mockEmailService.Object, mockLogger.Object);

13.

14. // Настройка на HttpContext без email claim

15. var httpContext = new DefaultHttpContext();

16. controller.ControllerContext = new ControllerContext() { HttpContext = httpContext };

17.

18. // Създаване на тестов модел за заявка за среща

19. var viewModel = new AppointmentCreateViewModel { Appointment = new Appointment { PropertyId = 1, RequestedAt = DateTime.Now } };

20.

21. // Act

22. var result = await controller.Create(viewModel);

23.

24. // Assert

25. var redirectResult = Assert.IsType<RedirectToActionResult>(result);

26. Assert.Equal("LoginRequired", redirectResult.ActionName);

27. }

28. }

29.

Създаване на ангажимент (POST) – Невалидни данни (Create\_Post\_InvalidAppointment\_ReturnsViewWithErrors):

**Цел:** Проверка дали системата обработва невалидни данни коректно (напр. липсващ Appointment обект).

**Стъпки:**

Симулира се потребител с валиден имейл claim.

Изпраща се форма с Appointment = null.

**Очакван резултат:** Връща се изглед с грешки в модела (ModelState.IsValid == false).

Ето е кодът на теста Create\_Post\_InvalidAppointment\_ReturnsViewWithErrors:

1. [Fact]

2. public async Task Create\_Post\_InvalidAppointment\_ReturnsViewWithErrors()

3. {

4. // Arrange

5. using (var context = new AppDbContext(\_dbContextOptions))

6. {

7. var mockUserService = new Mock<IUserService>();

8. // Симулиране на съществуващ текущ потребител.

9. mockUserService.Setup(s => s.GetUserByEmailAsync(It.IsAny<string>()))

10. .ReturnsAsync(new User { Id = 10, Email = "user@test.com" });

11.

12. var mockEmailService = new Mock<IEmailService>();

13. var mockLogger = new Mock<ILogger<AppointmentsController>>();

14.

15. var controller = new AppointmentsController(context, mockUserService.Object, mockEmailService.Object, mockLogger.Object);

16.

17. // Настройка на фалшив потребителски claim

18. var claims = new List<Claim> { new Claim(ClaimTypes.Email, "user@test.com") };

19. var identity = new ClaimsIdentity(claims, "TestAuth");

20. controller.ControllerContext.HttpContext = new DefaultHttpContext { User = new ClaimsPrincipal(identity) };

21.

22. // Симулиране на невалидни данни за среща (null среща)

23. var viewModel = new AppointmentCreateViewModel { Appointment = null };

24.

25. // Act

26. var result = await controller.Create(viewModel);

27.

28. // Assert

29. var viewResult = Assert.IsType<ViewResult>(result);

30. Assert.False(controller.ModelState.IsValid);

31. }

32. }

33.

Създаване на ангажимент (POST) – Успешно запазване и изпращане на имейл (Create\_Post\_ValidAppointment\_SavesAndSendsEmail):

**Цел:** Проверка дали валиден ангажимент се запазва в базата и изпраща имейл до потребителя.

**Стъпки:**

Добавя се тестов имот и потребител в базата.

Симулира се успешно влизане на потребител (чрез ClaimTypes.Email).

Изпраща се форма с валиден ангажимент.

**Очакван резултат:**

Ангажиментът се записва в базата със съответния потребител (UserId) и имот (PropertyId)

Имейл услуга (IEmailService) се извиква веднъж.

Препраща към **Index** действие.

Ето е кодът на теста Create\_Post\_ValidAppointment\_SavesAndSendsEmail:

1. [Fact]

2. public async Task Create\_Post\_ValidAppointment\_SavesAndSendsEmail()

3. {

4. // Подготовка

5. using (var context = new AppDbContext(\_dbContextOptions))

6. {

7. // Добавяне на имот и потребител

8. // Добавяне на имот с необходимите полета

9. var property = new Property

10. {

11. Id = 1,

12. Title = "Cool Pad",

13. Price = 1500,

14. Area = 80,

15. IsAvailable = true,

16. Address = "123 Test St.", // Добавяне на липсващ задължителен имот

17. Description = "A cool pad in a great location.", // Добавяне на липсващо задължително описание

18. Photos = "photo1.jpg", // Добавяне на липсваща снимка

19. PropertyType = "Residential" // Добавяне на липсващ тип имот

20. };

21. context.Properties.Add(property);

22. context.SaveChanges();

23.

24. var user = new User { Id = 10, Email = "user@test.com" };

25. var mockUserService = new Mock<IUserService>();

26. mockUserService.Setup(s => s.GetUserByEmailAsync("user@test.com"))

27. .ReturnsAsync(user);

28.

29. var mockEmailService = new Mock<IEmailService>();

30. mockEmailService.Setup(s => s.SendEmailAsync(It.IsAny<string>(), It.IsAny<string>(), It.IsAny<string>()))

31. .Returns(Task.CompletedTask);

32.

33. var mockLogger = new Mock<ILogger<AppointmentsController>>();

34.

35. var controller = new AppointmentsController(context, mockUserService.Object, mockEmailService.Object, mockLogger.Object);

36.

37. // Настройка на потребителски claim

38. var claims = new List<Claim> { new Claim(ClaimTypes.Email, "user@test.com") };

39. var identity = new ClaimsIdentity(claims, "TestAuth");

40. controller.ControllerContext.HttpContext = new DefaultHttpContext { User = new ClaimsPrincipal(identity) };

41.

42. // Изграждане на валиден модел за заявка за среща

43. var viewModel = new AppointmentCreateViewModel

44. {

45. Appointment = new Appointment { PropertyId = 1, RequestedAt = DateTime.Now }

46. };

47.

48. // Действие

49. var result = await controller.Create(viewModel);

50.

51. // Проверка

52. var redirectResult = Assert.IsType<RedirectToActionResult>(result);

53. Assert.Equal("Index", redirectResult.ActionName);

54. // Също така, проверяваме дали срещата е добавена в базата данни

55. Assert.True(context.Appointments.Any(a => a.PropertyId == 1 && a.UserId == user.Id));

56. // Проверка дали услугата за имейл е била извикана

57. mockEmailService.Verify(s => s.SendEmailAsync("user@test.com", It.IsAny<string>(), It.IsAny<string>()), Times.Once);

58. }

59. }

60.

**Общи бележки:**

* **Симулация на потребителски контекст:**
  + За тестове, изискващи потребителски контекст, се използват ClaimTypes.Email за симулиране на влязъл потребител.
  + При липса на имейл claim, системата препраща към логин страница.
* **Валидация на данни:**
  + При невалиден модел (напр. липсващ Appointment), контролерът връща грешки във View-то.
* **Изпращане на имейл:**
  + Използва се Mock<IEmailService> за проверка дали SendEmailAsync() се извиква с правилните параметри.
* **Изчистване на данни:**
  + Всеки тест използва нова InMemory база с GUID име, за да избегне конфликти между тестове.
* **Логиране:**
  + Добавен е Mock<ILogger> за симулиране на логер, но не се проверява активно в тестовете.

**Описание на тестовете за UsersController:**

1. Създаване на потребител с валидни данни (Create\_User\_ValidModel\_HashesPasswordAndSavesUser):
   * **Цел:** Проверка дали потребител с валиден имейл се запазва коректно, и ако паролата се хешира.
   * **Стъпки:**
   * Симулира се хеширане на парола чрез IPasswordHasher
   * Изпраща се заявка за създаване на потребител с валиден имейл (test@test.com)
   * **Очакван резултат:**
   * Потребителят се записва в базата.
   * Паролата се започва с hashed\_ (демек, е хеширана)
   * Пренасочва към действие Index.

1. [Fact]

2. public async Task Create\_User\_ValidModel\_HashesPasswordAndSavesUser()

3. {

4. // Подготовка

5. using (var context = new AppDbContext(\_dbContextOptions))

6. {

7. // Създаване на mock за IUserService и IEmailService

8. var mockUserService = new Mock<IUserService>();

9. var mockEmailService = new Mock<IEmailService>();

10.

11. // Създаване на fake password hasher

12. var mockPasswordHasher = new Mock<IPasswordHasher<User>>();

13. mockPasswordHasher.Setup(h => h.HashPassword(It.IsAny<User>(), It.IsAny<string>()))

14. .Returns((User u, string password) => $"hashed\_{password}");

15.

16. // Създаване на контролер и подаване на необходимите зависимости

17. var controller = new UsersController(context, mockPasswordHasher.Object, mockUserService.Object, mockEmailService.Object);

18.

19. // Уверяване се, че ModelState е валиден

20. controller.ModelState.Clear();

21.

22. var user = new User { Name = "Test User", Email = "test@test.com", Password = "secret", Role = "User" };

23.

24. // Действие

25. var result = await controller.Create(user);

26.

27. // Проверка

28. var redirectResult = Assert.IsType<RedirectToActionResult>(result);

29. Assert.Equal("Index", redirectResult.ActionName);

30.

31. var savedUser = context.Users.FirstOrDefault(u => u.Email == "test@test.com");

32. Assert.NotNull(savedUser);

33. Assert.StartsWith("hashed\_", savedUser.Password);

34. }

35. }

1. Създаване на потребител с невалиден имейл (Create\_User\_InvalidEmail\_ReturnsError):
   * **Цел:** Проверка дали системата отхвърля потребители с некоректен имейл формат.
   * **Стъпки:**
   * Изпраща се заявка за създаване на потребител с имейл invalid-email (невалиден формат).
   * **Очакван резултат:**
   * Връща се изглед с грешки във ModelState.
   * Състоянието на модела (ModelState.IsValid) е невалидно.

1. [Fact]

2. public async Task Create\_User\_InvalidEmail\_ReturnsError()

3. {

4. // Подготовка

5. using (var context = new AppDbContext(\_dbContextOptions))

6. {

7. var mockUserService = new Mock<IUserService>();

8. var mockEmailService = new Mock<IEmailService>();

9. var mockPasswordHasher = new Mock<IPasswordHasher<User>>();

10.

11. var controller = new UsersController(context, mockPasswordHasher.Object, mockUserService.Object, mockEmailService.Object);

12.

13. controller.ModelState.Clear(); // Изчистване на ModelState за теста

14.

15. // Създаване на невалиден потребител (например, невалиден имейл)

16. var user = new User { Name = "Test User", Email = "invalid-email", Password = "secret", Role = "User" };

17.

18. // Действие

19. var result = await controller.Create(user);

20.

21. // Проверка

22. Assert.IsType<ViewResult>(result); // Трябва да се върне View с валидирани грешки

23. Assert.False(controller.ModelState.IsValid); // Model state трябва да бъде невалиден заради валидирането на имейла

24. }

25. }

26.

1. Създаване на потребител със съществуващ имейл (Create\_User\_ExistingEmail\_ReturnsError):
   * **Цел:** Проверка дали системата блокира регистрация с вече зает имейл.
   * **Стъпки:**
   * Добавя се потребител с имейл [test@test.com](mailto:test@test.com) в базата.
   * Опит за регистрация на втори потребител със същия имейл.
   * **Очакван резултат:**
   * Връща се изглед с грешка “Имейлът вече е взет.“
   * Грешката се вижда в ModelState.

1. [Fact]

2. public async Task Create\_User\_ExistingEmail\_ReturnsError()

3. {

4. // Arrange

5. using (var context = new AppDbContext(\_dbContextOptions))

6. {

7. var mockUserService = new Mock<IUserService>();

8. var mockEmailService = new Mock<IEmailService>();

9. var mockPasswordHasher = new Mock<IPasswordHasher<User>>();

10.

11. var controller = new UsersController(context, mockPasswordHasher.Object, mockUserService.Object, mockEmailService.Object);

12.

13. controller.ModelState.Clear();

14.

15. // Добавяне на потребител със същия имейл към контекста (за симулиране на съществуващ потребител)

16. context.Users.Add(new User { Name = "Existing User", Email = "test@test.com", Password = "secret", Role = "User" });

17. await context.SaveChangesAsync();

18.

19. var user = new User { Name = "New User", Email = "test@test.com", Password = "secret", Role = "User" };

20.

21. // Act

22. var result = await controller.Create(user);

23.

24. // Assert

25. Assert.IsType<ViewResult>(result); // Трябва да върне изгледа с грешки при валидиране

26. Assert.Contains("Email is already taken.", controller.ModelState.Values.SelectMany(v => v.Errors).Select(e => e.ErrorMessage)); // Съобщението за грешка трябва да бъде видимо

27. }

28. }

29.

Общи бележки:

* Хеширане на пароли:
  + Паролите се обработват чрез IPasswordHasher, като тестовете симулират хеширане с добавката hashed\_.
* Валидация на имейл:
  + Системата проверява както формата на имейла, така и уникалността му в базата.
* Симулация на услуги:
  + IEmailService и IUserService се заменят с Mock обекти, за да се избегнат реални изпращания на имейли или зависимост от външни системи.
* Изчистване на контекст:
  + Всеки тест използва нова InMemory база с уникално име (GUID), за да гарантира изолираност.
* Обработка на грешки:
  + При грешки, контролерът връща изгледа с информация за проблемите в ModelState.

3. Заключение:

3.1. Резултати от проекта:

Проектът за създаване на уеб сайт за намиране и наемане на жилища и офис сгради успешно постигна основните си цели. Бяха разработени ключови функционалности като каталог на имоти, търсене и филтриране, потребителски профили, заявки за огледи и административен панел. Уебсайтът предоставя интуитивен и адаптивен интерфейс, който гарантира добро потребителско изживяване на различни устройства. Използването на трислойна архитектура (слой за данни, слой за услуги и слой за потребителски интерфейс) осигури стабилност, сигурност и лесна поддръжка на системата.

3.2. Проблеми при и предизвикателства при разработката:

* Проектиране на базата данни : Правилното моделиране на релациите между таблиците беше критично за осигуряване на целостта на данните.
* Интеграция на AJAX и динамични функционалности : Добавянето на интерактивни елементи, като динамично търсене и филтриране, изискваше прецизно координиране между клиентската и сървърната част.
* Сигурност : Внедряването на механизми за защита на потребителските данни и тяхното управление беше комплексен процес, който изискваше специално внимание.
* Мащабируемост : Проектирането на системата така, че да може да се разширява в бъдеще, изискваше допълнително планиране и анализ.

3.3. Препоръки и възможности за доразвиване на проекта:

За да се подобри и разшири функционалността на платформата, могат да бъдат внедрени следните подобрения:

* Интеграция с външни API-та : Добавяне на връзки с социални медии или плащания чрез популярни платформи (например PayPal, Stripe).
* Усъвършенстване на аналитиката : Разширена система за генериране на отчети, която ще позволи на администраторите да взимат по-информирани решения.
* Мобилно приложение : Разработване на мобилна версия или хибридно приложение, което ще предостави още по-голяма достъпност за потребителите.
* AI-базирано препоръчително табло : Интегриране на алгоритми за машинно обучение, които да предвиждат интересите на потребителите и да предлагат персонализирани предложения за имоти.

3.4. Потенциал за бъдещо приложение и разширения на системата:

* Глобално мащабиране : Системата може да бъде адаптирана за работа в различни региони, като се добавят многоезична поддръжка и локализация, като се използва бисквитки и JSON файлове за превод.
* Добавяне на нови функционалности : Може да се включат допълнителни услуги, като онлайн подписване на договори, виртуални огледи на имоти и автоматизирани напомняния за важни събития и също така плащания.
* Интеграция с IoT устройства : Свързване на платформата с умни домове или офиси, което ще позволи на потребителите да управляват имотите си дистанционно, напр. включване на светлините вируатлно.
* Разширяване на бизнес модела : Платформата може да се превърне в пазарна площадка, която не само помага за наемане на имоти, но и предлага допълнителни услуги като ремонт, дизайнерски решения и т.н.

4. Списък на използваната литература:

4.1. [Трислойна архитектура](https://thereaderwiki.com/bg/%D0%A2%D1%80%D0%B8%D1%81%D0%BB%D0%BE%D0%B9%D0%BD%D0%B0_%D0%B0%D1%80%D1%85%D0%B8%D1%82%D0%B5%D0%BA%D1%82%D1%83%D1%80%D0%B0)

1. Visual Studio:

* Официален сайт за сваляне на Visual Studio:
  + [Изтеглете Visual Studio](https://visualstudio.microsoft.com/downloads/?spm=a2ty_o01.29997173.0.0.7fccc921Qit4AO)
* Документация на Visual Studio:
  + [ASP.NET Core Документация](https://learn.microsoft.com/en-us/aspnet/core/?spm=a2ty_o01.29997173.0.0.7fccc921Qit4AO&view=aspnetcore-7.0)
* Уроци за стартиране с ASP.NET Core:
  + [Първи стъпки с ASP.NET Core](https://learn.microsoft.com/en-us/aspnet/core/getting-started/?spm=a2ty_o01.29997173.0.0.7fccc921Qit4AO&view=aspnetcore-7.0&tabs=windows)

2. SQL Server & SQL Server Management Studio (SMSS)

* Изтегляне на SQL Server:
  + [Изтеглете SQL Server](https://www.microsoft.com/en-us/sql-server/sql-server-downloads?spm=a2ty_o01.29997173.0.0.7fccc921Qit4AO)
* Изтегляне на SQL Server Management Studio (SSMS):
  + [Изтеглете SMSS](https://learn.microsoft.com/en-us/sql/ssms/download-sql-server-management-studio-ssms?spm=a2ty_o01.29997173.0.0.7fccc921Qit4AO&view=sql-server-ver16)
* Документация за SQL Server:
  + [SQL Server Документация](https://learn.microsoft.com/en-us/sql/sql-server/?spm=a2ty_o01.29997173.0.0.7fccc921Qit4AO&view=sql-server-ver16)
* Ръководство за начинаещи в SQL Server:
  + [Урок за SQL Server](https://www.sqlservertutorial.net/?spm=a2ty_o01.29997173.0.0.7fccc921Qit4AO) --