

Министерство образования Республики Беларусь
Учреждение образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей
Кафедра информатики
Дисциплина: Объектно-ориентированное программирование

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
к курсовой работе
на тему

ВЕБ-ПРИЛОЖЕНИЕ
ДЛЯ ОРГАНИЗАЦИИ ПУТЕШЕСТВИЙ

БГУИР 1-40 04 01

Студент: гр. 253502 Канавальчик А.Д.

Руководитель: ассистент кафедры
информатики, Рогов М.Г.

Минск 2024

СОДЕРЖАНИЕ

| | |
|--|----|
| ОПРЕДЕЛЕНИЯ И ТЕРМИНЫ | 5 |
| ВВЕДЕНИЕ | 6 |
| 1 ПОСТАНОВКА ЗАДАЧИ | 7 |
| 1.1 Исходная постановка задачи | 7 |
| 1.2 Описание средств разработки..... | 8 |
| 1.3 Спецификация функциональных требований..... | 9 |
| 1.4 Анализ рынка и существующих аналогов | 10 |
| 2 ПРОЕКТИРОВАНИЕ И РАЗРАБОТКА ПРОГРАММНОГО СРЕДСТВА.. | 12 |
| 2.1 Описание общей архитектуры | 12 |
| 2.1.1 ASP.NET Core | 12 |
| 2.1.2 Entity Framework Core | 13 |
| 2.1.3 MVC и Repository в разработке приложений..... | 13 |
| 2.1.4 Технологии программирования..... | 14 |
| 2.2 Структура данных | 15 |
| 2.2.1 Структура типов программы | 15 |
| 2.2.1.1 Изначальная схема классов..... | 16 |
| 2.2.1.2 Итоговая схема классов | 17 |
| 2.2.2 Структура данных программы..... | 18 |
| 3 РЕАЛИЗАЦИЯ ПРОГРАММНОГО СРЕДСТВА..... | 19 |
| 3.1 Реализация основного функционала | 28 |
| 3.2 Реализация системы рекомендаций | 28 |
| 4 ФУНКЦИОНАЛЬНЫЕ ВОЗМОЖНОСТИ..... | 19 |
| 4.1 Графический интерфейс..... | 28 |
| ЗАКЛЮЧЕНИЕ | 49 |
| СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ | 51 |
| Приложение А Исходный текст кода для визуализации Главной страницы.. | 52 |
| Приложение Б Исходный код..... | 63 |

ОПРЕДЕЛЕНИЯ И ТЕРМИНЫ

База данных (БД) – это упорядоченный набор структурированной информации или данных, которые обычно хранятся в электронном виде в компьютерной системе. Сама структура БД хранится в виде набора файлов, и единственный способ получить доступ к данным в этих файлах – через СУБД.

Интерфейс – граница между двумя функциональными объектами, требования к которой определяются стандартом; совокупность средств, методов и правил взаимодействия (управления, контроля и т. д.) между элементами системы.

Система управления базами данных (СУБД) – это набор программ, которые управляют структурой БД и контролируют доступ к данным, хранящимся в БД. СУБД служит посредником между пользователем и БД.

Веб-фреймворк – программная платформа, определяющая структуру программной системы; программное обеспечение, облегчающее разработку и объединение разных компонентов большого программного проекта (*ASP.NET Core Blazor*).

Entity Framework – это решение для работы с базами данных, которое используется в программировании на языках семейства .NET. Оно позволяет взаимодействовать с СУБД с помощью сущностей (entity), а не таблиц.

HTTPS (HyperText Transfer Protocol Secure) – это протокол, который обеспечивает целостность и конфиденциальность данных при их передаче между сайтом и устройством пользователя, используя шифрование.

Объектно-ориентированное программирование (ООП) – методология программирования, основанная на представлении программы в виде совокупности взаимодействующих объектов, каждый из которых является экземпляром определенного класса, а классы образуют иерархию наследования.

Паттерн проектирования – это многократно используемая архитектурная схема в разработке программного обеспечения, предлагающая стандартное решение для часто возникающих задач проектирования в определенном контексте.

Диаграмма вариантов использования (use-case диаграмма) – это диаграмма, показывающая взаимодействие между группами пользователей и системой, являющаяся частью модели прецедентов, которая позволяет описать систему на концептуальном уровне.

UML-диаграмма – это схема, нарисованная с применением символов UML. Она может содержать множество элементов и соединений между ними.

ВВЕДЕНИЕ

В современном мире путешествия стали неотъемлемой частью жизни многих людей. Возросшая мобильность и доступность различных видов транспорта, а также разнообразие туристических услуг создают необходимость в удобных и интегрированных решениях для планирования и организации поездок. Одним из таких решений являются веб-приложения, которые позволяют пользователям бронировать авиабилеты, жилье, арендовать автомобили и бронировать экскурсии и достопримечательности в одном месте.

Данная курсовая работа посвящена разработке инновационного веб-приложения для организации путешествий, которое объединяет все перечисленные функции. Основная цель приложения - упростить процесс планирования путешествий, предоставляя пользователю удобный и интуитивно понятный интерфейс для управления всеми аспектами поездки. В условиях стремительного развития информационных технологий и возрастающего спроса на удобные сервисы, такое приложение имеет значительный потенциал востребованности на рынке.

Основной функционал веб-приложения включает аутентификацию пользователей, бронирование жилья, авиабилетов, экскурсий и достопримечательностей, а также аренду автомобилей. Дополнительные функции для авторизованных пользователей включают возможность добавления понравившихся объектов в категорию «Любимые» и формирование системы рекомендаций на основе этих предпочтений. Это не только делает процесс планирования более персонализированным, но и значительно улучшает пользовательский опыт, предлагая индивидуализированные предложения и варианты.

При разработке веб-приложения будут применяться современные технологии и методы создания приложений, а также принципы проектирования. Особое внимание будетделено интеграции различных сервисов бронирования, обеспечению безопасности данных пользователей и удобству использования приложения. Для достижения высоких стандартов качества будут использованы принципы UX/UI-дизайна, что позволит создать интуитивно понятный и привлекательный интерфейс.

В заключении будут представлены результаты работы, а также обсуждены перспективы дальнейшего развития и улучшения приложения.

Таким образом, создание комплексного веб-приложения для организации путешествий не только способствует упрощению процесса планирования поездок для пользователей, но и открывает новые возможности для развития туристической индустрии в целом. Это приложение станет незаменимым помощником для всех, кто ценит своё время и стремится сделать свои путешествия максимально комфортными и запоминающимися.

1 ПОСТАНОВКА ЗАДАЧИ

1.1 Исходная постановка задачи

Веб-приложение "TravelHaven" представляет собой интегрированную систему, предназначенную для упрощения процесса планирования и организации путешествий. Приложение предоставляет следующие ключевые возможности:

1. Поиск и бронирование жилья: пользователи могут искать и бронировать отели, апартаменты и другие типы жилья, соответствующие их предпочтениям и бюджету.
2. Поиск и бронирование авиабилетов: удобный интерфейс позволяет быстро найти и забронировать авиабилеты на желаемые направления.
3. Аренда автомобилей: возможность поиска и бронирования автомобилей в пунктах назначения для комфортного передвижения.
4. Поиск и бронирование экскурсий и активностей: пользователи могут выбирать и бронировать экскурсии, мероприятия и другие виды активного отдыха.

"TravelHaven" также предоставляет расширенные функции для владельцев туристических объектов и компаний. Зарегистрированные владельцы могут добавлять свои объекты на сайт, делая их доступными для бронирования пользователями. Владельцы объектов обязаны подтверждать или отклонять брони, поступившие от пользователей, обеспечивая актуальность информации и доступность объектов.

Администраторы системы играют ключевую роль в обеспечении качества и надежности сервиса. Они имеют возможность подтверждать или отклонять объекты, предложенные для размещения на сайте различными компаниями, гарантируя соответствие объектов стандартам качества и безопасности.

Для удобства и персонализации пользовательского опыта в "TravelHaven" реализована категория "Любимые объекты". Пользователи могут добавлять понравившиеся объекты в этот список, а система рекомендаций на основе этих предпочтений предлагает наиболее выгодные и подходящие предложения, улучшая качество планирования поездки.

Кроме того, авторизованные пользователи могут легко отслеживать статус своих бронирований через удобный и интуитивно понятный интерфейс. Таким образом, "TravelHaven" объединяет все необходимые функции для комфортного и эффективного планирования путешествий, делая этот процесс простым и приятным.

Категория приложения: туризм, бизнес.

Тематика: планирование и организация путешествий.

Цель: цель проекта "TravelHaven" заключается в создании универсальной платформы, которая объединяет все аспекты планирования и организации путешествий, обеспечивая удобство, эффективность и безопасность для пользователей и владельцев туристических объектов. Для путешествующих пользователей "TravelHaven" предоставляет интегрированное решение для поиска и бронирования авиабилетов, жилья, автомобилей, экскурсий и других активностей, а также предлагает персонализированные рекомендации на основе предпочтений. Для владельцев туристических объектов и компаний платформа предоставляет возможности для продвижения своих услуг, управления бронированиями и взаимодействия с клиентами, что способствует росту бизнеса и улучшению качества обслуживания.

Целевая аудитория:

1. Туристы, планирующие отпуска и короткие поездки.
2. Деловые путешественники, часто находящиеся в командировках.
3. Семьи и группы друзей, координирующие совместные поездки.
4. Любители приключений, ищащие уникальные экскурсии и активные виды отдыха.
5. Отельеры и арендодатели жилья.
6. Авиакомпании и билетные агентства.
7. Компании по аренде автомобилей.
8. Туристические агентства и экскурсионные компании.
9. Малые и средние предприятия в сфере туризма.

1.2 Описание средств разработки

Для разработки проекта я использовала современные и мощные технологии, обеспечивающие высокую производительность и масштабируемость. Основным языком программирования стал C#, который позволил создать надежное и эффективное приложение. Для разработки серверной части я выбрала фреймворк ASP.NET Core, который славится своей скоростью, кроссплатформенностью и широкими возможностями для построения веб-приложений.

Для работы с базой данных я применила MS SQL Server, который известен своей стабильностью, безопасностью и производительностью. MS SQL Server идеально подходит для обработки больших объемов данных и обеспечения быстрого доступа к ним.

Для взаимодействия с базой данных я использовала Entity Framework Core. Этот ORM-фреймворк значительно упростил процесс работы с данными, позволяя использовать удобные и интуитивно понятные методы для выполнения запросов, управления транзакциями и обработки данных. Entity Framework Core также обеспечил высокую производительность и гибкость в работе с различными типами данных.

1.3 Спецификация функциональных требований

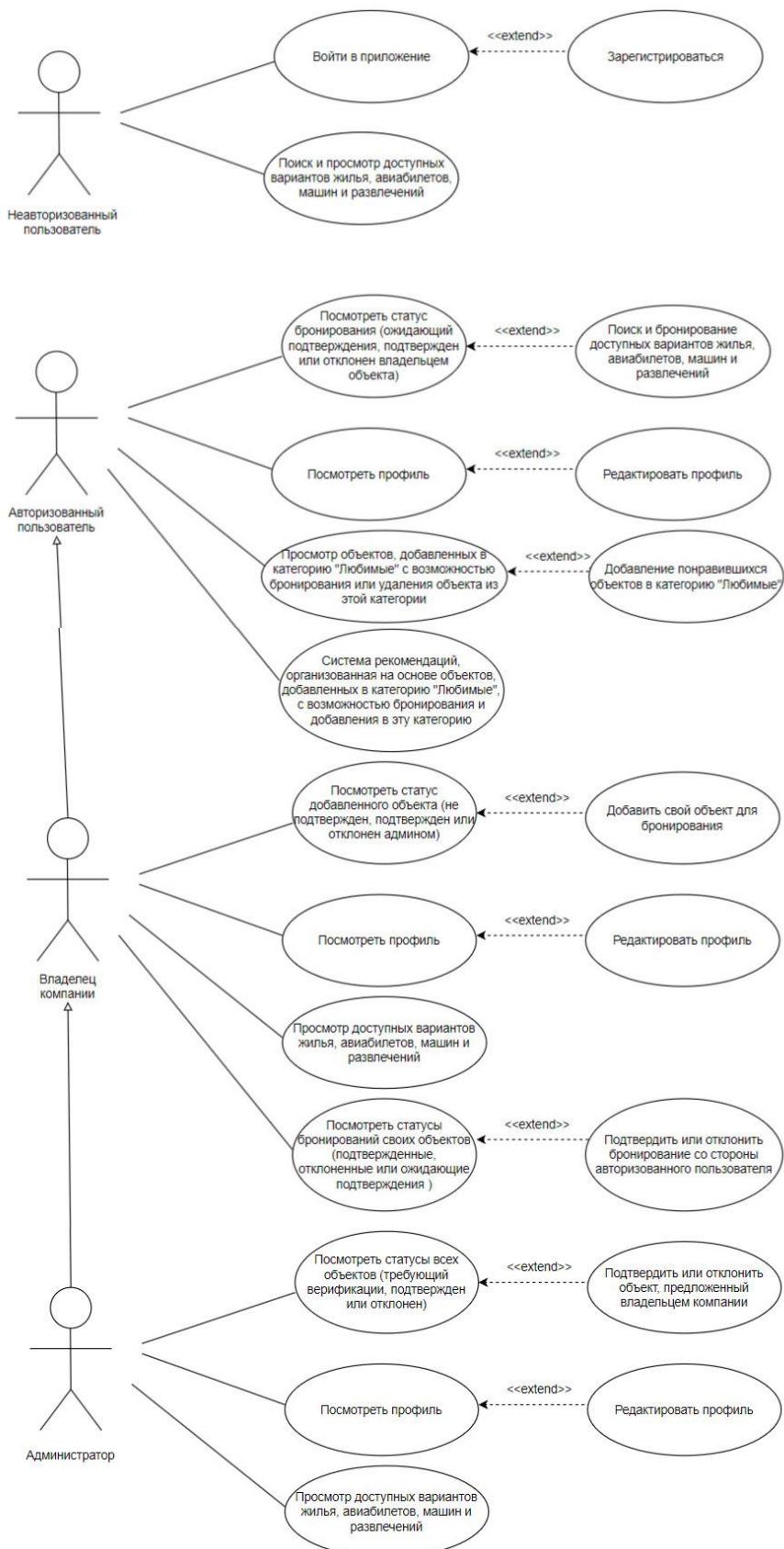


Рисунок 1 – Диаграмма вариантов использования

Неавторизованный пользователь – физическое лицо, имеющее возможность посмотреть варианты бронирований жилья, авиабилетов, автомобилей и развлечений.

Авторизованный пользователь – авторизованный пользователь, который может забронировать любой выбранный объект; посмотреть статус бронирования; добавить понравившийся объект в категорию «Любимые»; просматривать объекты, находящиеся в категории «Любимые» с возможностью брони; посмотреть, добавить в категорию «Любимые» или забронировать объекты, предложенные пользователю в качестве самых выгодных предложений.

Владелец компании – авторизованный пользователь, который имеет доступ к данным пользователя, забронировавшего его объект. Владелец также может предлагать к публикации своего объекты. После того, как предложенные объекты подтверждены админом, они будут предоставлены пользователям для бронирования. Данный тип пользователя имеет возможность просмотра бронирований своих объектов, а также уже ранее подтвержденных или отклоненных бронирований. Имеется возможность отслеживать статус своего объекта, предложенного к публикации на сайте.

Администратор – авторизованный пользователь, который имеет доступ ко всем объектам сайта. Администратор подтверждает или отклоняет объекты, предложенные к публикации владельцами компаний. Также он имеет возможность просматривать варианты жилья, авиабилетов, машин и развлечений по заданным критериям.

1.4 Анализ рынка и существующих аналогов

Рынок туристических веб-приложений динамично развивается благодаря увеличению числа людей, предпочитающих самостоятельно планировать свои путешествия. Цифровизация индустрии туризма и рост онлайн-бронирований привели к появлению множества платформ, которые стремятся удовлетворить разнообразные потребности путешественников. Основными тенденциями на рынке являются:

1. Интеграция различных услуг: Современные приложения стремятся предоставить пользователям возможность бронирования авиабилетов, жилья, аренды автомобилей и экскурсий в одном месте.
2. Персонализация: Использование данных пользователей для предоставления персонализированных рекомендаций и предложений.
3. Мобильность: Разработка мобильных приложений для удобного доступа к услугам в любом месте и в любое время.
4. Безопасность данных: Обеспечение высокого уровня защиты личной информации пользователей.

Рассмотрим основных конкурентов и аналоги. Сравнительный анализ нескольких веб-приложений для организации путешествий приведен в таблице 1.1.

Таблица 1.1 – Сравнительный анализ веб-приложений для организации путешествий

| Название | Описание | Ключевые функции | Преимущества | Недостатки |
|-------------|---|---|--|---|
| Booking.com | Один из крупнейших сервисов онлайн-бронирования жилья, включая отели, апартаменты, хостелы и другие типы размещения | Поиск и бронирование жилья, аренда автомобилей, бронирование авиабилетов, рекомендации на основе прошлых бронирований | Широкий выбор жилья, удобный интерфейс, мобильное приложение, гибкая система фильтров | Комиссии для владельцев объектов, ограниченные возможности для бронирования экскурсий и мероприятий |
| Airbnb | Платформа для аренды частного жилья, включая квартиры, дома, и уникальные варианты размещения | Поиск и бронирование частного жилья, возможность выбора уникальных объектов (например, дом на дереве), экскурсии и впечатления от местных жителей | Уникальные предложения, интеграция местных впечатлений, активное сообщество пользователей | Вариативность качества жилья, сервисные сборы, отсутствие аренды автомобилей и бронирования авиабилетов |
| Expedia | Крупный онлайн-тревел-агент, предлагающий широкий спектр услуг по бронированию | Бронирование авиабилетов, жилья, аренды автомобилей, круизов и пакетных туров | Комплексные предложения, пакетные туры, мобильное приложение | Сложный интерфейс, возможные скрытые сборы, ограниченные возможности для бронирования экскурсий |
| TripAdvisor | Платформа, известная своими отзывами и рекомендациями, а также услугами бронирования | Отзывы и рейтинги пользователей, бронирование отелей, ресторанов, экскурсий и активностей | Доверие пользователей к отзывам, разнообразие бронирования активностей, форум путешественников | Недостаточная интеграция с авиакомпаниями, сложности с навигацией на сайте |

2 ПРОЕКТИРОВАНИЕ И РАЗРАБОТКА ПРОГРАММНОГО СРЕДСТВА

2.1 Описание общей архитектуры

В этом разделе представляется общая архитектура разрабатываемого программного средства. Это включает в себя обзор основных технологий, используемых в проекте, а также описание основных компонентов и их взаимосвязей. Рассматриваются выбранные архитектурные паттерны и подходы, которые обеспечивают эффективное взаимодействие между различными частями системы. Кроме того, в этом разделе также могут быть описаны принципы проектирования, лежащие в основе архитектуры приложения, и решения, принятые для обеспечения его гибкости, масштабируемости и производительности.

2.1.1 ASP.NET Core

ASP.NET Core - это кросс-платформенный фреймворк для разработки веб-приложений, который предлагает разработчикам широкий спектр инструментов и возможностей для создания мощных и масштабируемых приложений. Одним из ключевых преимуществ ASP.NET Core является его кроссплатформенность, что позволяет разрабатывать приложения на различных операционных системах, таких как Windows, Linux и macOS. Это делает его идеальным выбором для разработчиков, которые предпочитают использовать различные платформы или развертывать свои приложения в различных окружениях.

ASP.NET Core также обладает высокой производительностью и эффективностью, благодаря использованию новейших технологий и оптимизациям, таким как встроенная поддержка асинхронного программирования и улучшенная система управления памятью. Это позволяет создавать быстрые и отзывчивые приложения, способные обрабатывать большие нагрузки даже при высоких нагрузках.

Другим важным преимуществом ASP.NET Core является его модульная архитектура и открытый исходный код, что обеспечивает гибкость и расширяемость при разработке приложений. Можно легко добавлять новые функции и компоненты, используя сторонние библиотеки или создавая их самостоятельно.

ASP.NET Core предлагает обширный набор инструментов и библиотек для разработки разнообразных типов приложений, включая веб-сайты, веб-приложения, API и многое другое. Это делает его универсальным инструментом для разработки различных проектов, независимо от их сложности и требований.

2.1.2 Entity Framework Core

Entity Framework Core (EF Core) - это легкий, расширяемый и мощный инструмент для работы с базами данных в .NET-приложениях. Он предоставляет удобный объектно-ориентированный подход к взаимодействию с данными, позволяя разработчикам работать с базами данных, используя объекты и LINQ-запросы, вместо языка SQL. EF Core обеспечивает абстракцию уровня данных, что делает код более чистым, поддерживаемым и масштабируемым.

Одним из главных преимуществ EF Core является его способность работать с различными провайдерами баз данных, такими как SQL Server, PostgreSQL, MySQL, SQLite и другими. Это делает его универсальным и позволяет разработчикам выбирать подходящий провайдер для своих конкретных потребностей.

EF Core также обеспечивает механизм миграций, который позволяет автоматически обновлять схему базы данных в соответствии с изменениями в модели данных, что существенно упрощает процесс развертывания и обновления приложений.

Кроме того, EF Core поддерживает асинхронные операции, что позволяет повысить производительность приложений за счет эффективного использования ресурсов.

Благодаря своей открытой сущности и активному сообществу разработчиков, EF Core постоянно развивается и обновляется, добавляя новые функции и улучшения, что делает его одним из наиболее популярных инструментов для работы с данными в .NET-экосистеме.

2.1.3 MVC и Repository в разработке приложений

MVC (Model-View-Controller) – это паттерн проектирования, который используется для разделения пользовательского интерфейса, бизнес-логики и данных в приложении. Он помогает организовать код и сделать его более структурированным и поддерживаемым. MVC широко применяется в различных технологиях разработки, включая веб-разработку, настольные и мобильные приложения.

Основные компоненты MVC:

1 Модель (Model): Представляет бизнес-логику и данные приложения. В модели определены классы, которые отвечают за получение и обработку данных, а также за взаимодействие с сервером или базой данных.

2 Представление (View): Отвечает за отображение пользовательского интерфейса. Включает в себя элементы управления, макеты и стилизацию. Представление не содержит бизнес-логики и должно быть максимально независимым от остальных компонентов.

3 Контроллер (Controller): Служит связующим звеном между моделью и представлением. Он отвечает за обработку пользовательского ввода. Контроллер получает входные данные от пользователя, вызывает

соответствующие методы модели для обработки данных и обновляет представление, чтобы отобразить изменения.

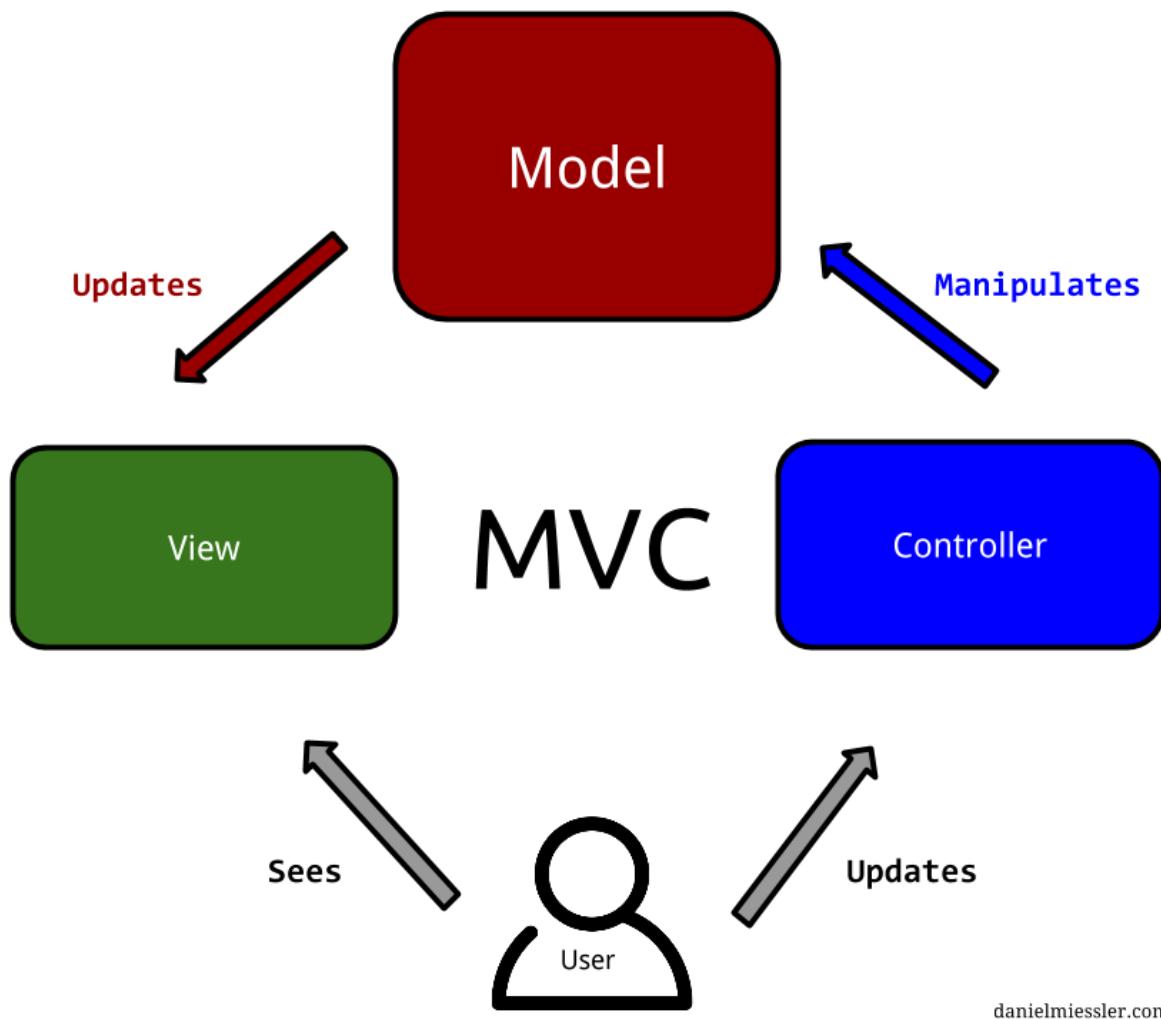


Рисунок 1 – Схема паттерна MVC

Repository – это паттерн проектирования, который используется для абстрагирования доступа к данным. Он предоставляет единый интерфейс для получения данных из различных источников, таких как базы данных, веб-сервисы или файлы. Репозиторий предоставляет методы для создания, чтения, обновления и удаления данных, скрывая детали их хранения и предоставляя простой и удобный интерфейс для работы с данными.

2.1.4 Технологии программирования

Рассмотрим применение конкретных принципов объектно-ориентированного программирования (ООП) и SOLID для достижения целей проекта.

В проекте применяется принцип абстракции, который позволяет скрыть детали реализации объектов и предоставить четкий интерфейс для

взаимодействия с ними. Внутренние механизмы объектов остаются невидимыми для внешнего кода, что позволяет изменять реализацию без влияния на клиентский код, который работает только с абстракциями.

Инкапсуляция используется для скрытия внутреннего состояния и функциональности объектов, предоставляя доступ только через открытые методы. Это повышает безопасность и надежность взаимодействия с объектами, защищая их данные от прямого доступа и неправильного использования.

Принцип единственной ответственности (SRP) соблюдается путем того, что каждый класс выполняет одну конкретную задачу и отвечает только за свою часть функциональности. Это упрощает сопровождение и расширение системы.

Принцип открытости/закрытости (OCP) реализован через создание классов, которые можно расширять, не изменяя существующий код. Это позволяет добавлять новые функции без риска внесения ошибок в уже работающий код.

Принцип подстановки Лисков (LSP) гарантирует, что объекты подклассов могут заменять объекты базового класса без изменения поведения программы. Это достигается строгим соблюдением контрактов, определенных в базовых классах и интерфейсах.

Таким образом, использование принципов абстракции, наследования и инкапсуляции в сочетании с принципами SOLID позволяет создать гибкую и расширяемую архитектуру. Каждый класс имеет четко определенные обязанности и может быть легко изменен или заменен при необходимости, что способствует улучшению поддерживаемости и масштабируемости проекта.

2.2 Структура данных

В этом пункте описывается, какие данные используются в программе, как они организованы, а также какие структуры данных применяются для их хранения и управления.

2.2.1 Структура типов программы

В этом подпункте описываются ключевые типы данных, используемые в программе, их свойства, методы и взаимосвязи. Типы данных организованы в виде классов и интерфейсов, которые представляют различные сущности и их взаимодействия.

Связи между объектами (Наименование отношений между объектами) представлены в таблице 2.1.

Таблица 2.1 – Связи между объектами

| Объект 1 | Объект 2 | Тип связи |
|---|--|--|
| Владелец | Публикуемый объект (жилье, авиабилет, машина, развлечение) | 1:M. Один Владелец может опубликовать много Объектов (или ни одного). Наименование отношения – публикует |
| Опубликованный объект (жилье, авиабилет, машина, развлечение) | Бронь | 1:M. Один Объект может иметь много Бронирований . Наименование отношения – бронирует |
| Авторизованный пользователь | Бронь | 1: M. Один Пользователь может делать много Броней(или ни одного). Наименование отношения – бронирует |
| Авторизованный пользователь | Объект в категории «Любимые» | 1: M. Один Пользователь может иметь много Объектов в категории «Любимые». Наименование отношения – добавляет |
| Администратор | Объект, предложенный Владельцем | 1:M. Один Администратор может иметь много объектов, предложенных Владельцем(или ни одного). Наименование отношения – подтверждает/ отклоняет |
| Владелец | Бронируемый Пользователем объект | 1:M. Один Владелец может иметь много Бронируемых Пользователем объектов (или ни одного). Наименование отношения – подтверждает/отклоняет |

2.2.1.1 Изначальная схема классов

Здесь представлена диаграмма классов (рисунок 2), показывающая первоначальную структуру типов программы. На данной диаграмме можно увидеть ключевые классы и их отношения на начальном этапе разработки. Основные компоненты включают:

Классы моделей: Представляют сущности, используемые в бизнес-логике. Диаграмма включает классы Hostel, Apartment, Hotel, Accommodation, Entertainment, Car, Admin, AirTicket, Seller, Traveller.

Связи между классами: Показывают ассоциации и зависимости между различными моделями.

Основные свойства и методы: Включают ключевые поля и методы для каждой модели, отображающие их функциональность.

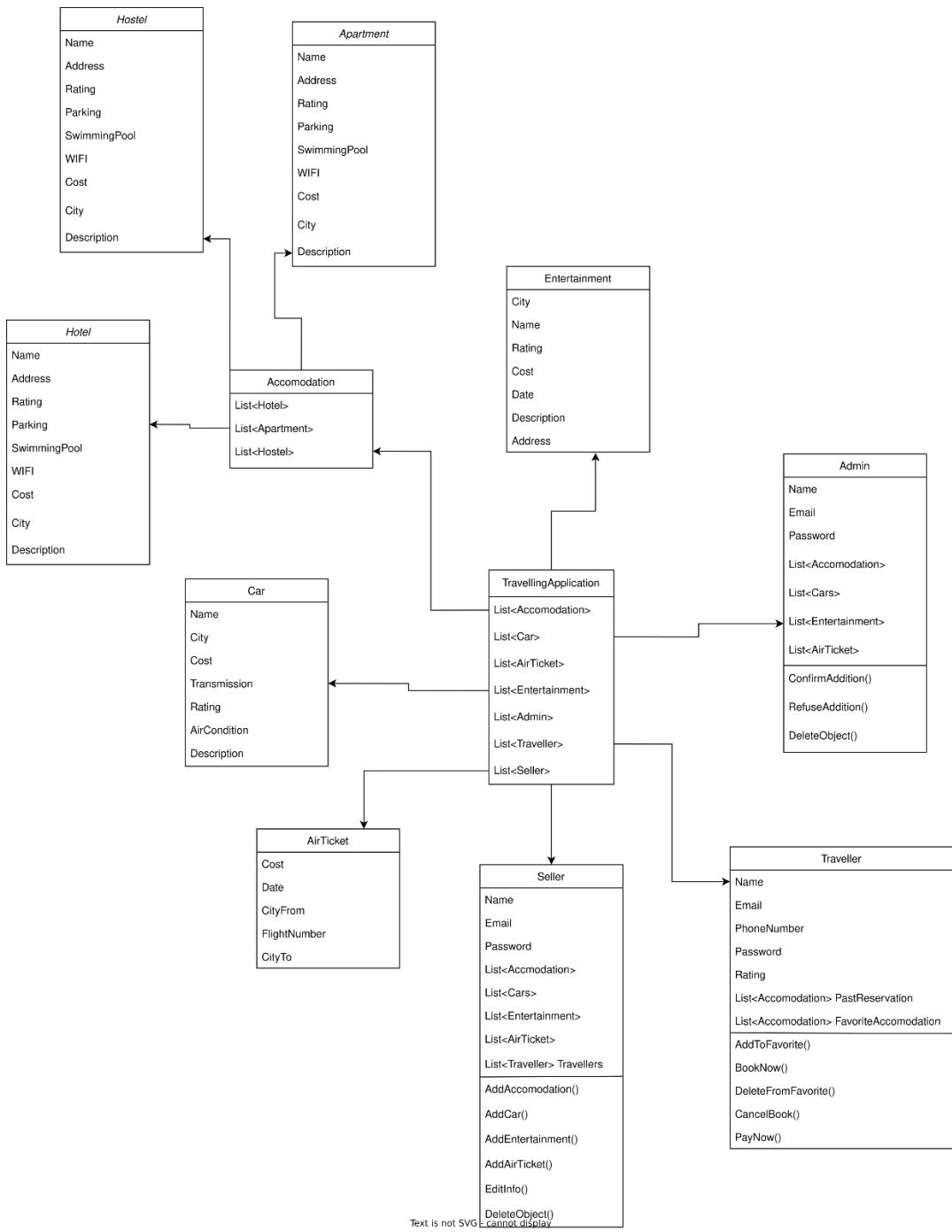


Рисунок 2 – Изначальная диаграмма классов

2.2.1.2 Итоговая схема классов

В приложение В помещена диаграмма классов, отражающая окончательную структуру типов программы после внесения всех изменений и доработок. Итоговая схема классов включает:

Обновленные классы моделей: Классы, модифицированные на основе требований проекта и улучшенные для большей эффективности. Например, в классы Car и Entertainment были добавлены новые атрибуты, а в некоторых других классах, наоборот, убраны ненужные.

Новые связи и зависимости: Дополненные и измененные отношения между моделями для оптимизации данных и бизнес-логики.

Изменение классов и интерфейсов: Оптимизация структуры кода путем удаления избыточных или неиспользуемых компонентов, что улучшает производительность и упрощает поддержку кода.

Итоговая диаграмма классов показывает завершенную и отлаженную структуру типов программы, готовую для использования в производственной среде.

2.2.2 Структура данных программы

Для соединения с базами данных мы используем Entity Framework Core (EF Core), который представляет собой ORM-технологию (object-relational mapping – отображения данных на реальные объекты). Позволяет абстрагироваться от самой базы данных и ее таблиц и работать с данными как с объектами класса независимо от типа хранилища (т.е есть возможность достаточно быстро поменять один из провайдеров, предложенных Microsoft, таких как MS SQL Server, SQLite, PostgreSQL).

В моем случае, я использую MS SQL Server в качестве провайдера базы данных. База данных создается на основе моделей и контекста данных (DbContext). Каждая модель соответствует таблице в базе данных, а контекст данных (DbContext) представляет общую схему базы данных.

В настоящий момент наша база данных состоит из набора таблиц, которые были созданы на основе наших моделей (рисунок 1). Для инициализации данных в приложении отвечает DbContext, который является входной точкой для работы с базой данных. Разделение сущностей (entity splitting) также было реализовано для облегчения доступа к таблицам базы данных.

В разработке проекта используется подход Code First в Entity Framework, что означает, что база данных генерируется на основе классов. EF Core предоставляет набор базовых методов для работы с базой данных, таких как добавление и удаление данных, которые могут быть использованы в вашем приложении.

Миграции в Entity Framework Core (EF Core) играют ключевую роль в управлении схемой базы данных при использовании подхода Code First. Они позволяют легко изменять структуру базы данных, отслеживая и применяя изменения, внесенные в модели классов. Миграция представляет собой класс, который описывает эти изменения. Они создаются с помощью командной строки или интерфейса управления пакетами (Package Manager Console) и содержат инструкции для обновления базы данных до новой версии схемы.

3 РЕАЛИЗАЦИЯ ПРОГРАММНОГО СРЕДСТВА

3.1 Реализация основного функционала

Рассмотрим реализацию основного функционала веб-приложения “TravelHaven”, а именно: добавление объектов со стороны владельца, изменение объектов, верификация админом предложенных объектов, бронирование объектов, отмена бронирования объектов, добавление понравившихся объектов в категорию «Любимые».

Для выполнения всех этих функций используются два ключевых контроллера: HomeController и AccountController.

Владельцы могут добавлять новые объекты, такие как автомобили, на сайт с помощью метода SaveForm() контроллера AccountController (рис. 1).

```
// Преобразуем модель представления в объект Car и сохраним его в базе данных
var car = new Car
{
    Title = model.Title,
    City = model.City,
    Country = model.Country,
    Address = model.Address,
    IsAirCondition = model.IsAirCondition,
    Transmission = model.Transmission,
    Cost = model.Cost,
    CarPhoto = imageDataa,
    FreeCancellation = model.FreeCancellation,
    AmountOfPassengers = model.AmountOfPassengers,
    TheftCoverage = model.TheftCoverage,
    CollisionDamageWaiver = model.CollisionDamageWaiver,
    LiabilityCoverage = model.LiabilityCoverage,
    UnlimitedMileage = model.UnlimitedMileage,
    CarCategory = model.CarCategory,
    ElectricCar = model.ElectricCar,
    Description = model.Description,
    StartDates = model.StartDates,
    EndDates = model.EndDates,
    VerifiedByAdmin = false,
    PublisherId = currentUser.Id,
    RejectedByAdmin = false,
    RejectedMessage = string.Empty
};
// добавить сохранение dat в другую БД
_context.Car.Add(car);

await _context.SaveChangesAsync();

ViewBag.CarPhoto = fileBytes;

// Устанавливаем сообщение об успешном изменении в TempData
TempData["SuccessMessage"] = "The object Car was successfully saved and sent for moderation.";
```

Рисунок 1 – Исходный код функции SaveForm()

Функция SaveForm() вызывается, когда владелец кликает на кнопку "Save" для добавления нового автомобиля, заполнив все необходимые данные в форме. В метод передается модель, сформированная на основе заполненной владельцем формы. Внутри метода создается объект класса Car, свойства которого инициализируются значениями, полученными из формы. Затем новый объект машины добавляется в базу данных и изменения сохраняются. После успешного сохранения и отправки объекта на модерацию, владельцу отображается сообщение, подтверждающее успешное добавление его автомобиля и отправку на проверку.

Добавление объектов жилья, авиабилетов и достопримечательностей осуществляется аналогичным образом. Внутри метода создаются объекты соответствующих классов: Accommodation, AirTicket и Entertainment. Эти объекты инициализируются данными, введенными владельцем в форму, после чего сохраняются в базе данных, обеспечивая их доступность на платформе.

Владельцы также имеют возможность изменять ранее добавленные объекты. Это осуществляется через соответствующие методы: UpdateForm(), UpdateFlightForm(), UpdateEntertainmentForm() и UpdateAccomodationForm(), в зависимости от типа редактируемого объекта.

Рассмотрим код функции UpdateForm(), предназначенной для сохранения изменений в редактируемом объекте (рис. 2). Этот метод позволяет владельцам обновлять информацию об объекте, обеспечивая актуальность и точность данных, представленных на платформе.

```
    }

    var car = _context.Car.Find(editCarId);

    car.Title = model.Title;
    car.City = model.City;
    car.Country = model.Country;
    car.Address = model.Address;
    car.IsAirCondition = model.IsAirCondition;
    car.Transmission = model.Transmission;
    car.Cost = model.Cost;
    car.CarPhoto = imageDataa;
    car.FreeCancellation = model.FreeCancellation;
    car.AmountOfPassengers = model.AmountOfPassengers;
    car.TheftCoverage = model.TheftCoverage;
    car.CollisionDamageWaiver = model.CollisionDamageWaiver;
    car.LiabilityCoverage = model.LiabilityCoverage;
    car.UnlimitedMileage = model.UnlimitedMileage;
    car.CarCategory = model.CarCategory;
    car.ElectricCar = model.ElectricCar;
    car.Description = model.Description;
    car.StartDates = model.StartDates;
    car.EndDates = model.EndDates;
    car.PublisherId = currentUser.Id;
    car.VerifiedByAdmin = false;
    car.RejectedByAdmin = false;

    await _context.SaveChangesAsync(); // Сохраняем изменения в базе данных
    return await UnverifiedObjects();
```

Рисунок 2 – Исходный код функции UpdateForm()

Функция UpdateForm() вызывается, когда владелец кликает по кнопке "Save" для сохранения внесённых изменений в объект машины. В метод передаётся идентификатор редактируемого автомобиля, по которому осуществляется его поиск в базе данных Car. После нахождения нужного объекта класса Car, в него вносятся соответствующие изменения, которые затем сохраняются в базе данных. Функции UpdateFlightForm(), UpdateEntertainmentForm() и UpdateAccomodationForm() реализованы подобным образом.

Администратор имеет возможность отклонять или подтверждать объекты, предложенные владельцами. Подтверждение объектов реализовано с помощью методов VerifyCar(), VerifyAttraction(), VerifyAccommodation(), VerifyAirTicketEC(), VerifyAirTicketBC() и VerifyAirTicketFC(). Для отклонения объектов используются методы RejectCar(), RejectAttraction(), RejectAccommodation(), RejectAirTicketEC(), RejectAirTicketBC() и RejectAirTicketFC(). Выбор конкретного метода зависит от типа объекта, который необходимо подтвердить/отклонить.

Рассмотрим код функции VerifyCar(), предназначеннной для подтверждения объекта машины (рис. 3).

```
Ссылок: 0
public async Task<IActionResult> VerifyCar(int verifyCarId)
{
    var car = _context.Car.Find(verifyCarId);

    car.VerifiedByAdmin = true;

    await _context.SaveChangesAsync();

    return await AllUnverifiedObjects();
}
```

Рисунок 3 – Исходный код функции VerifyCar()

Функция VerifyCar() вызывается, когда администратор нажимает на кнопку "Verify" для подтверждения предложенного владельцем объекта машины. В метод передаётся идентификатор автомобиля, который необходимо подтвердить. После этого выполняется поиск соответствующего объекта класса Car в базе данных по переданному id. Найденный объект обновляется: его свойство VerifiedByAdmin устанавливается в значение true, и изменения сохраняются в базе данных. Функции VerifyAttraction(), VerifyAccommodation(), VerifyAirTicketEC(), VerifyAirTicketBC() и VerifyAirTicketFC() реализованы аналогичным образом.

Рассмотрим код функции RejectCar(), предназначеннной для отклонения объекта машины (рис. 4).

```

Ссылок: 0
public async Task<IActionResult> RejectCar(int rejectCarId, IFormCollection form)
{
    var car = _context.Car.Find(rejectCarId);
    var message = form["RejectedMessageCar"];

    car.RejectedByAdmin = true;
    car.RejectedMessage = message;

    await _context.SaveChangesAsync();

    return await AllUnverifiedObjects();
}

```

Рисунок 4 – Исходный код функции RejectCar()

Функция RejectCar() вызывается, когда администратор нажимает на кнопку "Reject" для отклонения предложенного владельцем объекта машины, предварительно указав причину отклонения. В метод передаётся идентификатор автомобиля, который необходимо отклонить, а также объект типа IFormCollection. После этого выполняется поиск соответствующего объекта класса Car в базе данных по переданному id. Из объекта класса IFormCollection извлекаем причину отклонения объекта, чтобы проинформировать владельца. Найденный объект обновляется: его свойство RejectedByAdmin устанавливается в значение true, а свойство RejectedMessage инициализируется причиной отклонения, изменения сохраняются в базе данных. Функции RejectAttraction(), RejectAccommodation(), RejectAirTicketEC(), RejectAirTicketBC() и RejectAirTicketFC() реализованы аналогичным образом.

Авторизованный пользователь может забронировать любой доступный объект. Для этого используются функции BookCar(), BookFlightEC(), BookFlightBC(), BookFlightFC(), BookAccommodation() и BookAttraction(). Выбор конкретного метода зависит от типа объекта, который пользователь хочет забронировать.

Рассмотрим код функции BookCar(), предназначеннной для брони автомобиля по заданным параметрам (рис. 5).

Функция BookCar() вызывается после того, как пользователь кликнет на кнопку “Book” с целью брони текущего объекта машины в соответствии с заданными параметрами. В метод передается идентификатор бронируемой машины и даты, на которые пользователь хочет ее забронировать. В самом методе мы находим текущего пользователя, текущий объект машины по id. После чего создаем бронируемую модель с передачей в нее некоторых свойств: Id пользователя, который бронирует объект; Id бронируемой машины; свойство подтверждения бронирования VerifiedBooking; свойство отказа бронирования RejectedBooking и свойство сообщения об отклонении

RejectedMessage; даты бронирования и свойство отклоненного пользователем бронирования CanceledBooking. После чего созданная модель сохраняется в базу данных BookingCars.

```
[HttpPost]
Ссылок 0
public async Task<IActionResult> BookCar(string carId, DateTime dateIn, DateTime dateOut)
{
    var currentUser = _context.Users.FirstOrDefault(u => u.UserName == User.Identity.Name);
    int CarId = int.Parse(carId);
    var car = _context.Car.Find(CarId);

    var model = new BookingCar
    {
        UserId = currentUser.Id,
        CarId = CarId,
        VerifiedBooking = false,
        RejectedBooking = false,
        RejectedMessage = string.Empty,
        DateOfDeparture = dateIn,
        ReturnDate = dateOut,
        CanceledBooking = false
    };

    _context.BookingCars.Add(model);
}
```

Рисунок 5 – Исходный код функции BookCar()

Функции BookFlightEC(), BookFlightBC(), BookFlightFC(), BookAccommodation() и BookAttraction() работают подобным образом.

Авторизованный пользователь имеет возможность отменить бронь любого ранее забронированного объекта. Реализуется это посредством функций CancelCarBooking(), CancelAttractionBooking(), CancelAccomodationBooking(), CancelAirTicketECBooking(), CancelAirTicketBCBooking(), CancelAirTicketFCBooking(). Выбор конкретной функции зависит от типа объекта, на который пользователь хочет отменить бронирование.

Рассмотрим код функции CancelCarBooking(), предназначенный для отмены бронирования автомобиля (рис. 6).

```
[HttpPost]
Ссылок 0
public async Task<IActionResult> CancelCarBooking(int cancelCarId)
{
    var car = _context.BookingCars.Find(cancelCarId);
    DateTime startDate = car.DateOfDeparture;
    DateTime endDate = car.ReturnDate;

    int carId = car.CarId;
    var originalCar = _context.Car.Find(carId);
    originalCar.StartDates.Add(startDate);
    originalCar.EndDates.Add(endDate);
    _context.Car.Update(originalCar);

    car.CanceledBooking = true;
    await _context.SaveChangesAsync();

    return await ShowUnverifiedBookingItems();
}
```

Рисунок 6 – Исходный код функции CancelCarBooking()

Функция CancelCarBooking() вызывается после того, как пользователь кликнет по кнопке “Cancel Booking” с целью отменить бронирование автомобиля. Функция принимает на вход идентификатор автомобиля, на который пользователь хочет отменить бронирование. После чего по идентификатору находим машину в базе данных BookingCars. Делаем доступными для брони автомобиля ранее забронированные даты, после чего устанавливаем свойство машины CanceledBooking в true, обновляем объект автомобиля и сохраняем все изменения в базе данных. Функции CancelAttractionBooking(), CancelAccomodationBooking(), CancelAirTicketECBooking(), CancelAirTicketBCBooking(), CancelAirTicketFCBooking() реализованы подобным образом.

Особенность авторизованного пользователя заключается в том, что он может добавить понравившиеся объекты в категорию «Любимые» с возможностью дальнейшего бронирования. Данный функционал реализуется посредством функций AddToFavoriteCar(), AddToFavoriteAttraction(), AddToFavoriteAirTicketEC(), AddToFavoriteAirTicketBC(), AddToFavoriteAirTicketFC(), AddToFavoriteAccomodation(). Выбор конкретной функции зависит от того, какой тип объект пользователь хочет добавить в категорию «Любимые».

Рассмотрим код функции AddToFavoriteCar(), предназначенной для добавления в категорию «Любимые» объекта машины (рис. 7).

```
[HttpPost]
Ссылок 0
public async Task<IActionResult> AddToFavoriteCar(int carId, DateTime dateIn, DateTime dateOut)
{
    var currentUser = _context.Users.FirstOrDefault(u => u.UserName == User.Identity.Name);

    var model = new FavoriteCar
    {
        UserId = currentUser.Id,
        CarId = carId,
        DateOfDeparture = dateIn,
        ReturnDate = dateOut
    };

    _context.FavoriteCars.Add(model);
    await _context.SaveChangesAsync();

    return Json(new { isFavorite = true });
}
```

Рисунок 7 – Исходный код функции AddToFavoriteCar()

Функция AddToFavoriteCar() вызывается, когда пользователя кликает по иконке сердечка с целью добавления объекта машины в категорию «Любимые». Функция принимает на вход идентификатор машины и выбранные пользователем даты. После чего получаем текущего пользователя, формируем объект типа FavoriteCar и инициализируем его свойства UserId, CarId, DateOfDeparture, ReturnDate. Созданную модель добавляем в базу данных FavoriteCars с последующим сохранением

изменений в базе данных. Функции AddToFavoriteAttraction(), AddToFavoriteAirTicketEC(), AddToFavoriteAirTicketBC(), AddToFavoriteAirTicketFC(), AddToFavoriteAccomodation() реализованы подобным образом.

3.2 Реализация системы рекомендаций

Веб-приложение "TravelHaven" включает в себя продвинутую систему рекомендаций, которая значительно улучшает пользовательский опыт. Авторизованные пользователи могут добавлять понравившиеся объекты в категорию «Любимые», и на основе этих предпочтений формируется персонализированная система рекомендаций.

Система рекомендаций анализирует выбор пользователя и предлагает индивидуализированные варианты жилья, авиабилетов, экскурсий, достопримечательностей и автомобилей. Благодаря этому каждый пользователь получает уникальные предложения, которые соответствуют его интересам.

Эта функциональность делает процесс планирования путешествий не только более удобным, но и вдохновляющим, предоставляя пользователям идеи и предложения, которые они, возможно, не нашли бы самостоятельно.

Суть системы рекомендаций состоит в поиске самых выгодных предложений для пользователей в городах, в которых находятся объекты, которые они добавляли в категорию «Любимые».

Система рекомендаций реализована посредством функции AttractiveOffers(), которая возвращает список выгодных объектов.

Рассмотрим детальную реализацию системы (рис. 8).

```
var currentUser = _context.Users.FirstOrDefault(u => u.UserName == User.Identity.Name);

var favoriteCarIds = _context.FavoriteCars
    .Where(fc => fc.UserId == currentUser.Id)
    .Select(fc => fc.CarId)
    .ToList();

var favoriteCarCities = _context.Cars
    .Where(c => favoriteCarIds.Contains(c.Id))
    .Select(c => c.City)
    .ToList();

var favoriteAttractionIds = _context.FavoriteAttractions
    .Where(fc => fc.UserId == currentUser.Id)
    .Select(fc => fc.AttractionId)
    .ToList();

var favoriteAttractionCities = _context.Entertainment
    .Where(c => favoriteAttractionIds.Contains(c.Id))
    .Select(c => c.City)
    .ToList();

var favoriteAirTicketIds = _context.FavoriteAirTickets
    .Where(fc => fc.UserId == currentUser.Id)
    .Select(fc => fc.AirTicketId)
    .ToList();

var favoriteAirTicketCities = _context.AirTicket
    .Where(c => favoriteAirTicketIds.Contains(c.Id))
    .Select(c => new { c.CityFrom, c.CityTo })
    .ToList();

var favoriteAccommodationIds = _context.FavoriteAccomodations
    .Where(fc => fc.UserId == currentUser.Id)
    .Select(fc => fc.AccommodationId)
    .ToList();

var favoriteAccommodationCities = _context.Accomodation
    .Where(c => favoriteAccommodationIds.Contains(c.Id))
    .Select(c => c.City)
    .ToList();
```

Рисунок 8 – Детальная реализация системы рекомендаций

Первое, что необходимо сделать, это получить текущего пользователя. После чего выведем список идентификаторов объектов машин, добавленных текущим пользователем в категорию «Любимые». Выведем все города, в которых находятся любимые объекты автомобилей. Аналогичным образом выведем список идентификаторов объектов авиабилетов, достопримечательностей и жилья. После чего получим соответствующие списки городов, в которых находятся данные любимые объекты. Таким образом, у нас есть 4 списка городов и 4 списка идентификаторов объектов соответствующих типов.

Рассмотрим дальнейшую реализацию системы рекомендаций (рис. 9).

```

var allFavoriteCities = new HashSet<string>();
// Добавление городов в HashSet
allFavoriteCities.UnionWith(favoriteCarCities);
allFavoriteCities.UnionWith(favoriteAttractionCities);
foreach (var city in favoriteAirTicketCities)
{
    allFavoriteCities.Add(city.CityFrom);
    allFavoriteCities.Add(city.CityTo);
}
allFavoriteCities.UnionWith(favoriteAccommodationCities);

var cars = _context.Car
    .Where(car => allFavoriteCities.Contains(car.City) && car.VerifiedByAdmin) // Фильтруем машины, чтобы выбрать
    .GroupBy(car => car.City) // Группируем машины по городу
    .Select(group => group.OrderBy(car => car.Cost).FirstOrDefault()) // Из каждой группы выбираем машину с минимальной ценой
    .ToList();

var attractions = _context.Entertainment
    .Where(attraction => allFavoriteCities.Contains(attraction.City) && attraction.VerifiedByAdmin) // Фильтруем достопримечательности
    .GroupBy(attraction => attraction.City) // Группируем достопримечательности по городу
    .Select(group => group.OrderBy(attraction => attraction.Cost).FirstOrDefault()) // Из каждой группы выбираем самую выгодную
    .ToList();

var accommodations = _context.Accommodation
    .Where(accommodation => allFavoriteCities.Contains(accommodation.City) && accommodation.VerifiedByAdmin) // Фильтруем жилье
    .GroupBy(accommodation => accommodation.City) // Группируем жилье по городу
    .Select(group => group.OrderBy(accommodation => accommodation.MinCost).FirstOrDefault()) // Из каждой группы выбираем самое дешевое
    .ToList();

var airticketsFrom = _context.AirTicket
    .Where(airticket => allFavoriteCities.Contains(airticket.CityFrom) && airticket.VerifiedByAdmin)
    .GroupBy(airticket => airticket.CityFrom)
    .Select(group => group.OrderBy(airticket => airticket.CostEC).FirstOrDefault())
    .ToList();

var airticketsTo = _context.AirTicket
    .Where(airticket => allFavoriteCities.Contains(airticket.CityTo) && airticket.VerifiedByAdmin)
    .GroupBy(airticket => airticket.CityTo)
    .Select(group => group.OrderBy(airticket => airticket.CostEC).FirstOrDefault())
    .ToList();

var uniqueAirtickets = airticketsFrom.Union(airticketsTo, new AirTicketComparer()).ToList();

```

Рисунок 9 – Детальная реализация системы рекомендаций

Следующим шагом мы объединим 4 списка городов в один список allFavoriteCities, который будет содержать только уникальные города. После чего начнем поиск самых выгодных предложений для каждого из типов объектов. Получим список выгодных вариантов машин, проверяя свойство city, значение которого обязательно должно содержаться в списке allFavoriteCities, а также определяя самый дешевый объект по стоимости в текущем городе. Таким образом мы формируем список объектов типа “Car”, которые содержат самые выгодные предложения для каждого из городов.

Аналогичным образом формируем списки выгодных предложений для достопримечательностей и жилья. В случае авиабилетов мы формируем два

списка. Первый список будет перебирать города, из которых вылетает самолет, а второй список будет перебирать города, в которые летит самолет. Однако в таком случае есть вероятность того, что некоторые билеты в этих двух списках будут совпадать. Чтобы избежать этого, создадим список uniqueAirtickets, который будет содержать только уникальные выгодные авиабилеты.

Завершающим этапом реализации системы рекомендаций станет формирование единого объекта, включающего все четыре списка объектов с выгодными предложениями (рис. 10). Этот объект объединит персонализированные рекомендации по жилью, авиабилетам, экскурсиям и достопримечательностям, а также аренде автомобилей, предоставляя пользователю комплексное и удобное решение для планирования путешествий.

```
var model = new VerifiedObjectsViewModel()
{
    VerifiedCars = cars,
    VerifiedAirTickets = airTickets,
    VerifiedAttractions = attractions,
    VerifiedAccomodation = accomodations
};

return View(model);
```

Рисунок 10 – Завершающий этап реализации системы рекомендаций

Таким образом, была реализована система рекомендаций, предлагающая самые выгодные и подходящие варианты для различных типов объектов. Эта система позволяет пользователям легко находить оптимальные предложения, соответствующие их личным предпочтениям и потребностям. В результате пользователи могут наслаждаться более удобным и персонализированным процессом планирования, находя идеальные варианты жилья, авиабилетов, экскурсий, достопримечательностей и автомобилей, специально подобранные для них.

4 ФУНКЦИОНАЛЬНЫЕ ВОЗМОЖНОСТИ

4.1 Графический интерфейс

Навигация по проекту осуществляется с помощью меню, расположенного сверху страницы и доступного на каждой странице. Содержимое меню меняется в зависимости от роли текущего пользователя.

Стартовая страница для неавторизованного пользователя проиллюстрирована на рисунке 1.

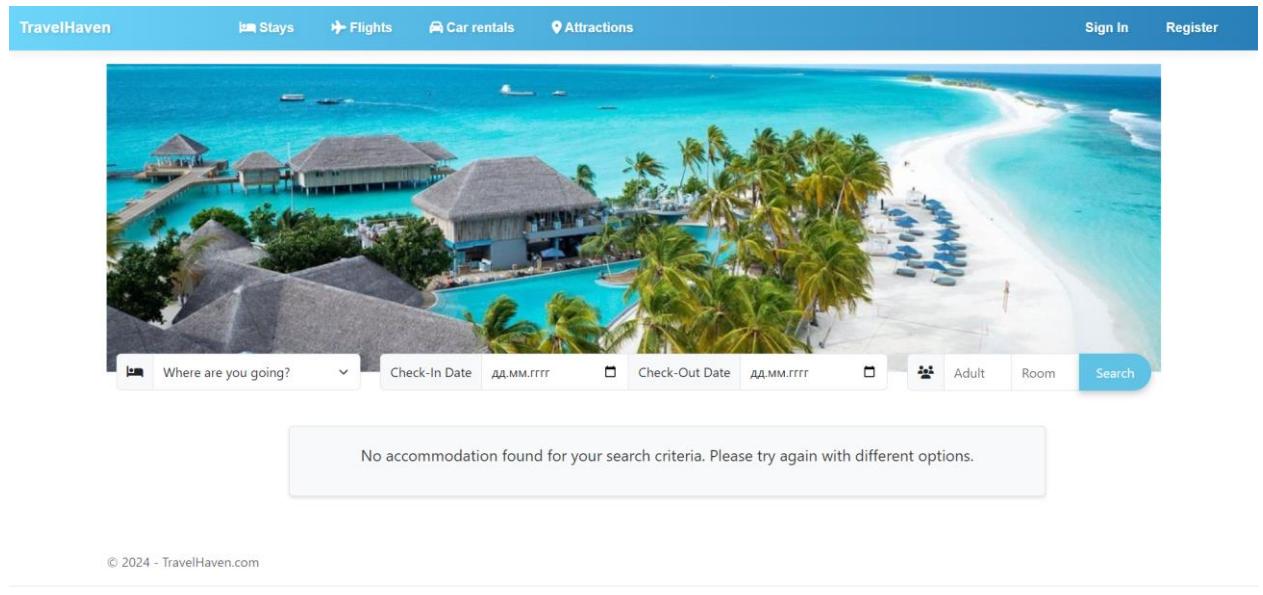


Рисунок 1 – Стартовая страница неавторизованного пользователя

Данная страница соответствует функционалу Stays шапки сайта. Она позволяет подобрать соответствующие заданным параметрам объекты жилья. В случае, если пользователь введет некорректные данные для поиска объекта или не введет их вовсе, ошибочные поля приобретут красные границы (рис. 2).

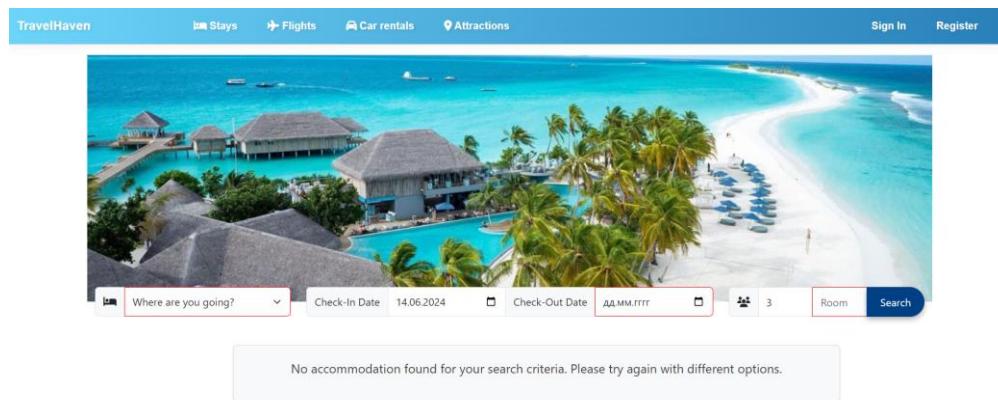


Рисунок 2 – Стартовая страница с некорректно заполненными полями

После корректного заполнения всех требуемых полей в случае существования объектов по заданным критериям они будут выведены в виде списка (рис. 3).

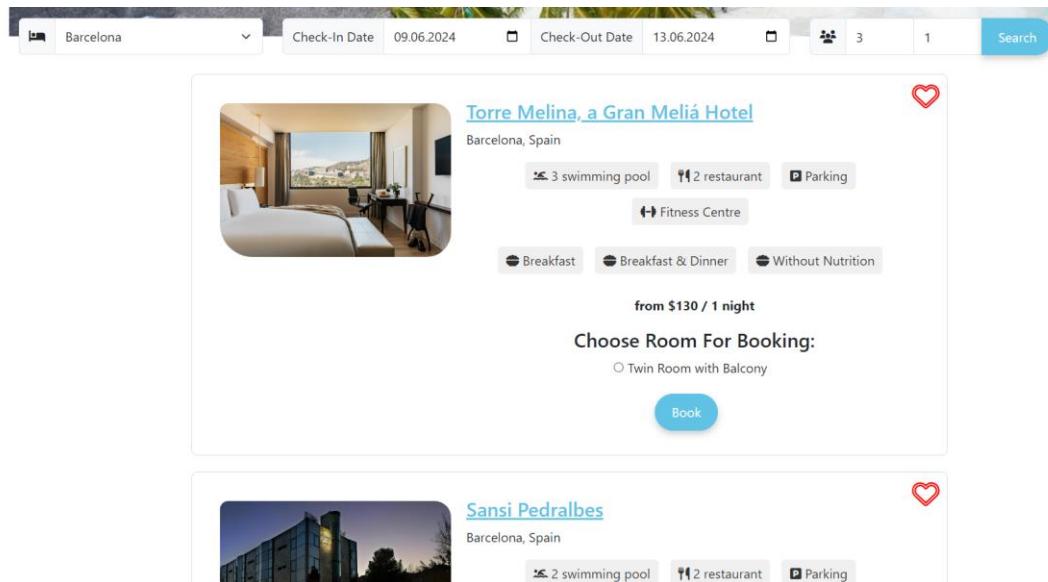


Рисунок 3 – Объекты жилья

Каждый объект содержит краткую информацию в виде иконок, а также основную информацию в виде названия объекта, города и страны нахождения, стоимости. Только для пользователей с ролью “Traveller” есть возможность забронировать объект, а также добавить его в категорию «Любимые», нажав на иконку сердечка. Кликнув по названию объекта, откроется страница с подробной информацией об самом объекте жилья, а также о всех комнатах, свободных в соответствии с заданными критериями. (рис. 4).

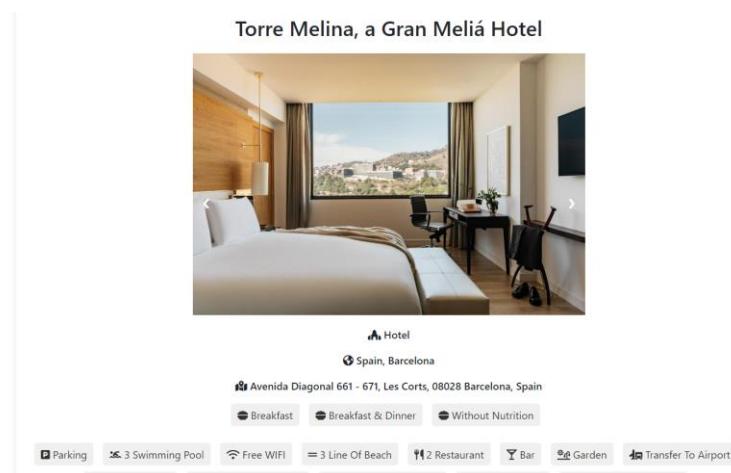


Рисунок 4 – Подробная информация об объекте

В случае, если пользователь не выбрал тип комнаты для бронирования перед добавлением объекта в категорию «Любимые» или непосредственно бронирования, он будет об этом проинформирован (рис. 5).

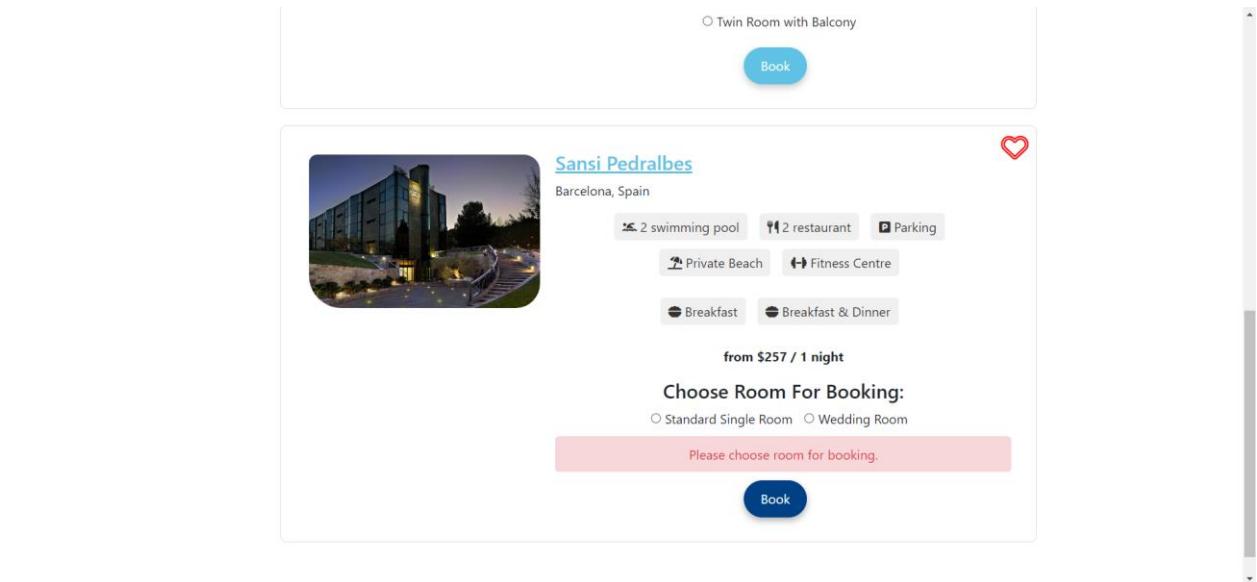


Рисунок 5 – Обработка некорректного бронирования

После успешного бронирования текст кнопки будет изменен на “Booked” и она перестанет быть кликабельной (рис. 6).

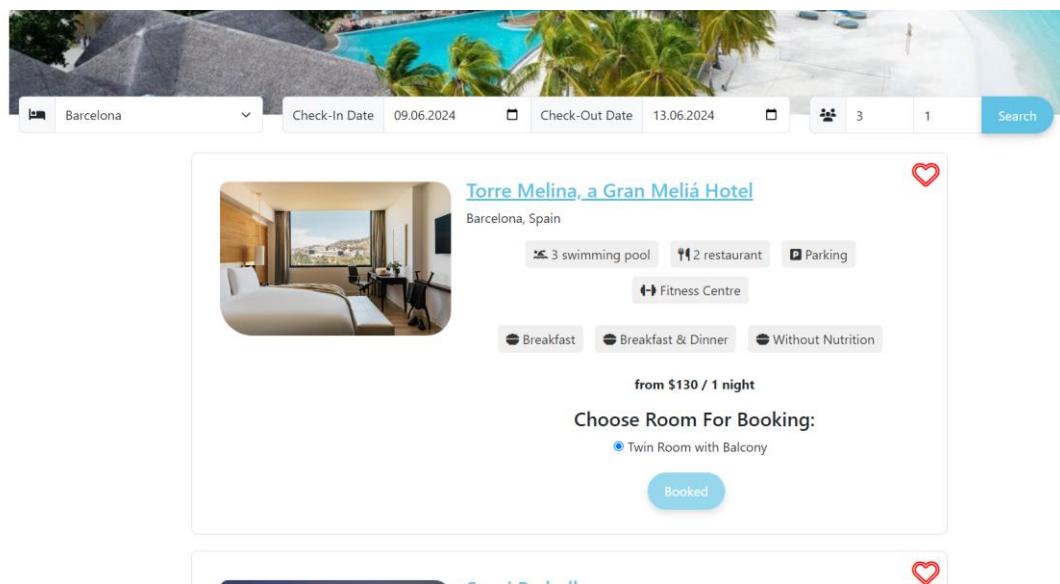


Рисунок 6 – Бронирование объекта жилья

После успешного добавления объекта в категорию «Любимые» иконка сердечка заполнится красным цветом (рис. 7).

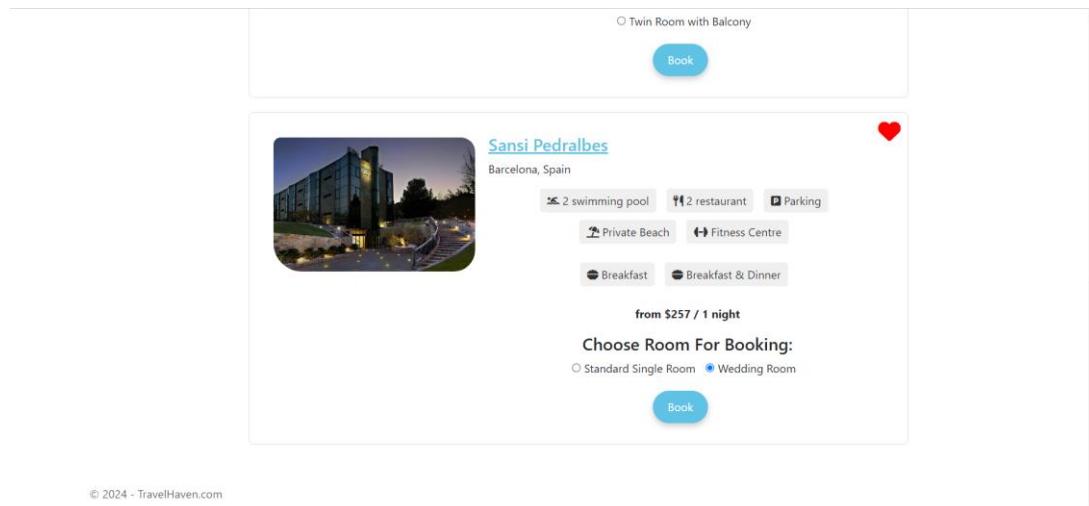


Рисунок 7 – Добавление объекта в категорию «Любимые»

Неавторизованный пользователь, а также пользователи с ролями “Admin” и “Owner” имеют возможность только найти и посмотреть объекты по заданным критериям. Возможность бронирования и добавления объекта в категорию «Любимые» у них отсутствует.

Рассмотрим функционал страницы, на которую мы попадаем, кликнув по Flights шапки сайта (рис. 8). Обработка некорректного ввода параметров для бронирования реализована для всех страниц для поиска объектов для бронирования, включая Car rentals, Flights, Attractions.

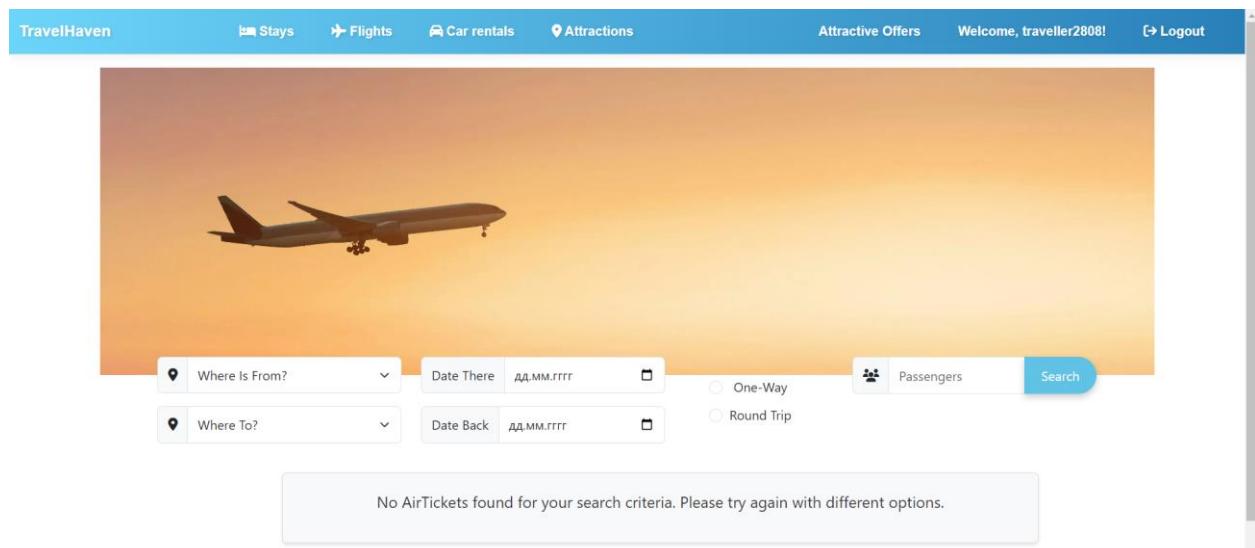


Рисунок 8 – Страница для бронирования авиабилетов

Данная страница предназначена для поиска авиабилетов, соответствующих заданным параметрам. В случае, если билет нужен только в одну сторону, можно выбрать пункт “One-Way”, после чего поле Date Back исчезнет.

В случае, если существуют билеты по заданным критериям, они будут выведены (рис. 9).

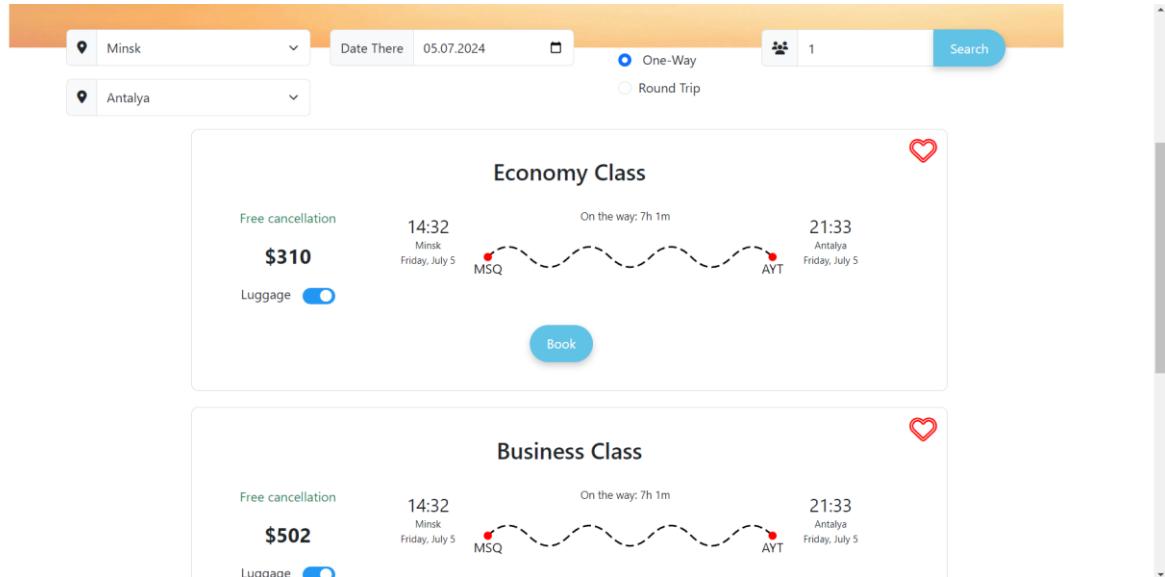


Рисунок 9 – Доступные для брони авиабилеты

Объект любой категории можно как забронировать, так и добавить в категорию «Любимые» по ранее рассмотренным принципам.

Для поиска машин для аренды соответствует Car rentals в шапке страницы, кликнув по которой мы попадем на соответствующую страницу (рис. 10).

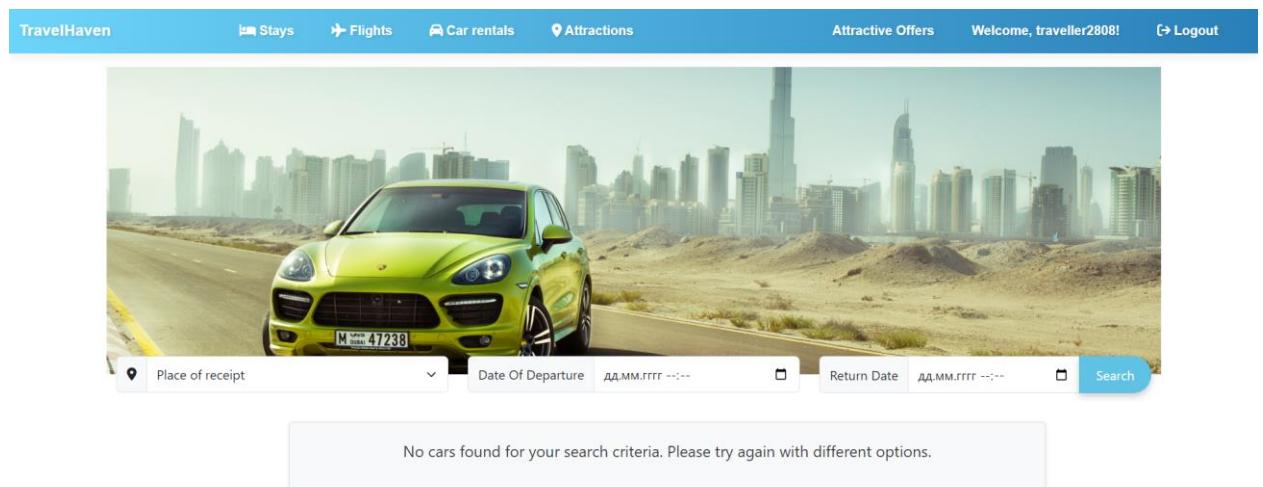


Рисунок 10 – Страница для аренды машин

Если существуют свободные для брони автомобили по заданным пользователем критериям, то они будут выведены (рис. 11).

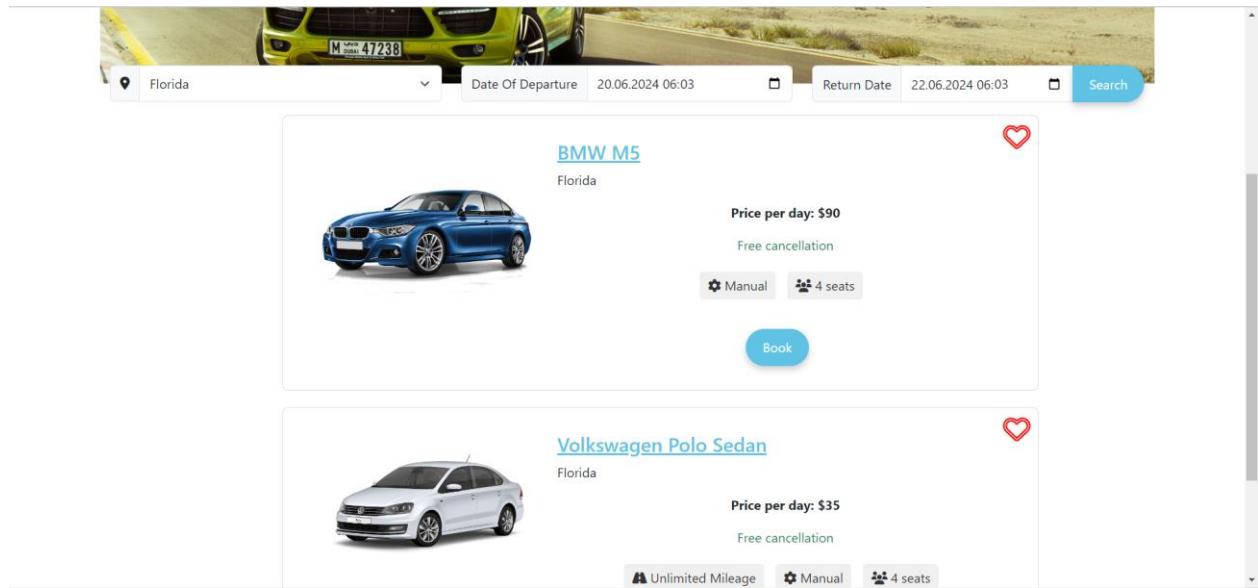


Рисунок 11 – Список машин, доступных для аренды

Кликнув по названию машины, мы попадем на страницу подробной информации о ней (рис. 12).

A screenshot of a detailed car information page for a BMW M5. The top navigation bar includes links for 'TravelHaven', 'Stays', 'Flights', 'Car rentals', 'Attractions', 'Attractive Offers', 'Welcome, traveller2808!', and 'Logout'. The main content area features a large image of a blue BMW M5 sedan. Above the car, the model name 'BMW M5' is displayed. Below the car, the location 'USA, Florida' and address '1600 W. New Haven Avenue' are shown. A series of buttons provide options for 'Limited Mileage', 'Manual', '4 seats', 'Air Condition', 'Electric Car', and 'Large Car'. A section titled 'Included in the price' lists 'Theft Protection with €950 excess' and 'Free cancellation up to 48 hours before pick-up'. A descriptive paragraph at the bottom explains the unique characteristics of the BMW M5.

Рисунок 12 – Страница подробной информации об автомобиле

Кликнув по Attractions в шапке сайта, мы попадем на страницу для поиска доступных развлечений для бронирования (рис. 13).

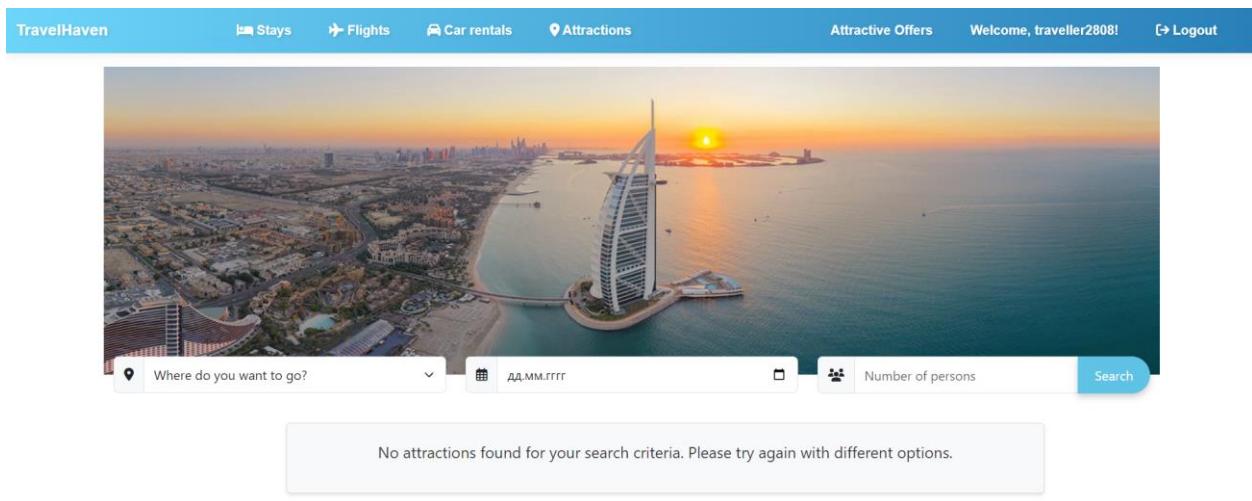


Рисунок 13 – Страница подробной информации об автомобиле

В случае, если по заданным критериям объекты развлечений для бронирования были найдены, они будут выведены (рис. 14).

Museum Of The Future
Dubai, UAE
Price per 1 person: \$65
Free cancellation
All Day
English Spanish Italian Chinese Arabic
Book

The View at The Palm | Dubai Skyline Marvel
Dubai, UAE
Price per 1 person: \$70

Рисунок 14 – Доступные объекты развлечений для бронирования

Кликнув по названию объекта развлечений, мы попадем на страницу подробной информации об объекте (рис. 15).



▲ Museum, arts & culture
📍 UAE, Dubai
📍 Sheikh Zayed Rd - Trade Centre - Trade Centre 2 - Dubai
✓ Free cancellation up to 48 hours before pick-up
🕒 English 🕒 Spanish 🕒 French 🕒 Italian 🕒 Chinese 🕒 Arabic
🕒 All Day

Museum of the Future officially opened its doors to be an incubator for ideas and innovation.^[6] On the evening of 22 February 2022, the museum was inaugurated by His Highness Sheikh Mohammed bin Rashid Al Maktoum, UAE Vice President, Prime Minister and Ruler of Dubai, with a vibrant lightshow to mark the occasion.^[7] The opening ceremony saw His Highness Sheikh Mohammed bin Rashid Al Maktoum; alongside Sheikh Hamdan bin Mohammed bin Rashid Al Maktoum, Crown Prince of Dubai, and Sheikh Maktoum bin Mohammed bin Rashid Al Maktoum initiate the projections together at the press of a button.^[8] The goal of this museum is to be the first to look to the future, rather than the past.^[9] It is said that everything in the museum is predicted to be in 2071.

Available Dates & Amount Of Tickets:

Рисунок 15 – Страница подробной информации об объекте развлечений

Кликнув по «Welcome, @Username!» в шапке сайта пользователь с ролью “Owner” попадет на страницу, проиллюстрированную на рисунке 16.

The screenshot shows the 'ACCOUNT SETTINGS' section of the TravelHaven website. On the left, there is a sidebar with links: Account, Add Object, My Verified Objects, My Unverified Objects, My Rejected Objects, Bookings For Verification, Verified Bookings, and Rejected Bookings. The main area displays the company name 'Tour Company "Amazing Tours"' above a circular profile picture placeholder. Below the placeholder are three buttons: 'Upload', 'Save', and 'Delete'. Further down are input fields for 'Name & Surname' (containing 'Tour Company "Amazing Tours"'), 'Email' (containing 'Enter your Email'), and another 'Enter your Email' field.

Рисунок 16 – Основной функционал пользователя с ролью “Owner”

Данная страница представляет собой персональные настройки профиля. Они аналогичны для всех авторизованных пользователей.

Кликнув по “Add Object”, мы переходим на страницу добавления объекта (рис. 17).

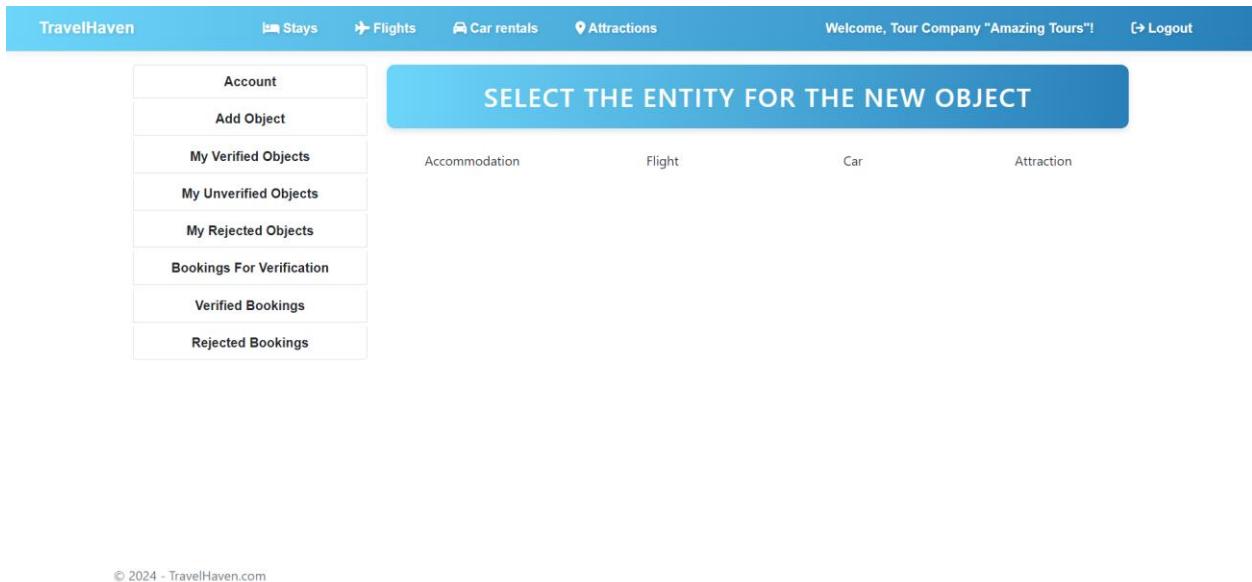


Рисунок 17 – Страница добавления объекта

В зависимости от того, какой объект мы хотим добавить, откроются соответствующие текстовые поля. Все текстовые поля предусматривают проверку на вводимые данные (рис. 18).

The form is for adding an accommodation object. It includes:

- A sidebar with options: My Verified Objects, My Unverified Objects, My Rejected Objects, Bookings For Verification, Verified Bookings, and Rejected Bookings.
- A header with tabs: Accommodation (selected), Flight, Car, and Attraction.
- A red message box: "Please add accommodation photos."
- An "Upload Photos" button.
- Text input fields for "Accommodation Name", "Country Of Accommodation", "City Of Accommodation", "Address Of Accommodation", and "Time Of Accommodation".

Рисунок 18 – Обработка вводимых данных

К особому случаю можно отнести добавление объекта жилья, так как изначально вы заполняем данные об отеле, а после, кликнув по кнопке “Add Room” откроется форма для добавления комнаты в текущий вид жилья (рис. 19).

TravelHaven

Stays Flights Car rentals Attractions

Welcome, Tour Company "Amazing Tours!" Logout

New Room

Upload Photos

Room Name

Enter the room name

Washing Machine
 Yes No

Kitchen
 Yes No

Wheelchair Accessible
 Yes No

Toilet With Grab Bars
 Yes No

Bathtub With Grabbars

Рисунок 19 – Страница добавления комнаты

Минимальная стоимость за ночь в конкретном виде жилья формируется в соответствии с самой дешевой комнатой, предлагаемой этим отелем (рис. 20).

Free cancellation

\$890

12:40 Minsk Friday, June 14 MSQ

On the way: 6h 5m

18:45 Hurghada Friday, June 14 HRG

Luggage

Edit Delete

Rixos Premium JBR Hotel

Dubai, UAE

4 swimming pool 5 restaurant Parking Private Beach SPA Fitness Centre Breakfast Breakfast & Dinner All Inclusive

from \$590 / 1 night

Add Room Edit Delete

© 2024 - TravelHaven.com

Рисунок 20 – Неподтвержденный объект жилья

После того, как объект будет добавлен, он будет отображаться в категории “My Unverified Objects”, тем самым ожидая подтверждения или отклонения со стороны администратора (рис. 21).

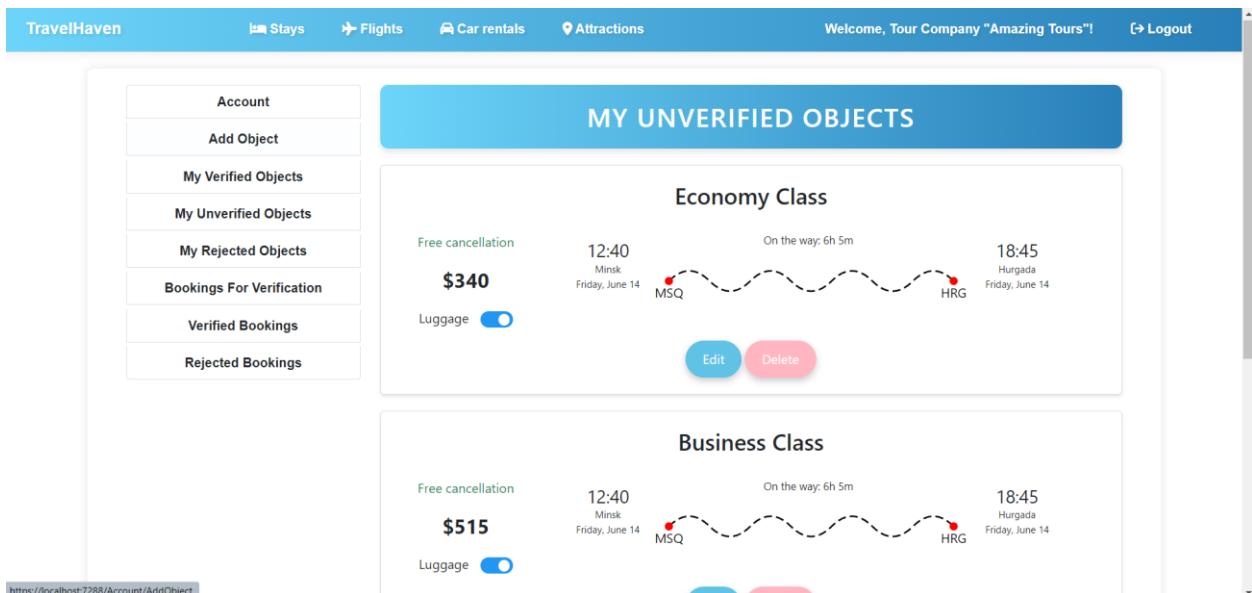


Рисунок 21 – Страница неподтвержденных админом объектов

Каждый неподтвержденный объект можно отредактировать или вовсе удалить. Страница редактирования изображена на рисунке 22. Некорректно заполненные поля также обрабатываются.

| | |
|----------------------|---------|
| Departure Country | Belarus |
| Departure City Code | MSQ |
| Departure City | Minsk |
| Country of Arrival | Egypt |
| City Code of Arrival | HRG |
| City of Arrival | Hurgada |

Рисунок 22 – Страница редактирования объекта

Страница подтвержденных объектов работает по тому же принципу. Выводятся все подтвержденные администратором объекты. Также есть возможность отредактировать объект и удалить. Однако после редактирования объекта он снова отправится в категорию “Unverified Objects”, ожидая подтверждения или отклонения от админа. Кликнув по названию объекта, мы попадем на страницу подробной информации. Страница подтвержденных объектов продемонстрирована на рисунке 23.

Рисунок 23 – Страница подтвержденных объектов

В случае, если объект был отклонен администратором, он отобразится в категории “My Rejected Objects” с информационным сообщением о причине отклонения (рис. 24). Также есть возможность отредактировать объект, учитывая пожелания администратора и снова отправить на верификацию админом. Можно удалить объект или, кликнув по его названию, посмотреть подробную информацию. В случае объекта жилья на странице подробной информации есть возможность отредактировать комнату.

Рисунок 24 – Страница отклоненных объектов

Страница “Bookings For Verification” предназначена для того, что владелец компании смог отклонять или подтверждать бронирования на его объекты (рис. 25). Каждый объект дополнительно выводит информацию о пользователе, который его забронировал.

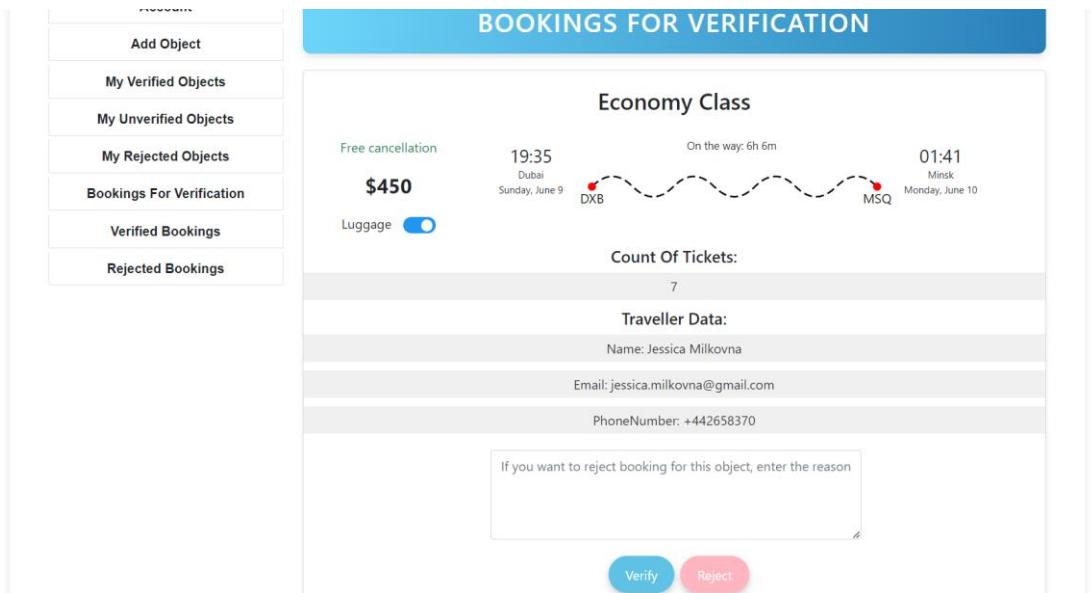


Рисунок 25 – Страница броней объектов

Если владелец компании забудет или не захочет указывать причину отклонения, то ему вовсе не удастся отклонить данное бронирование (рис. 26).

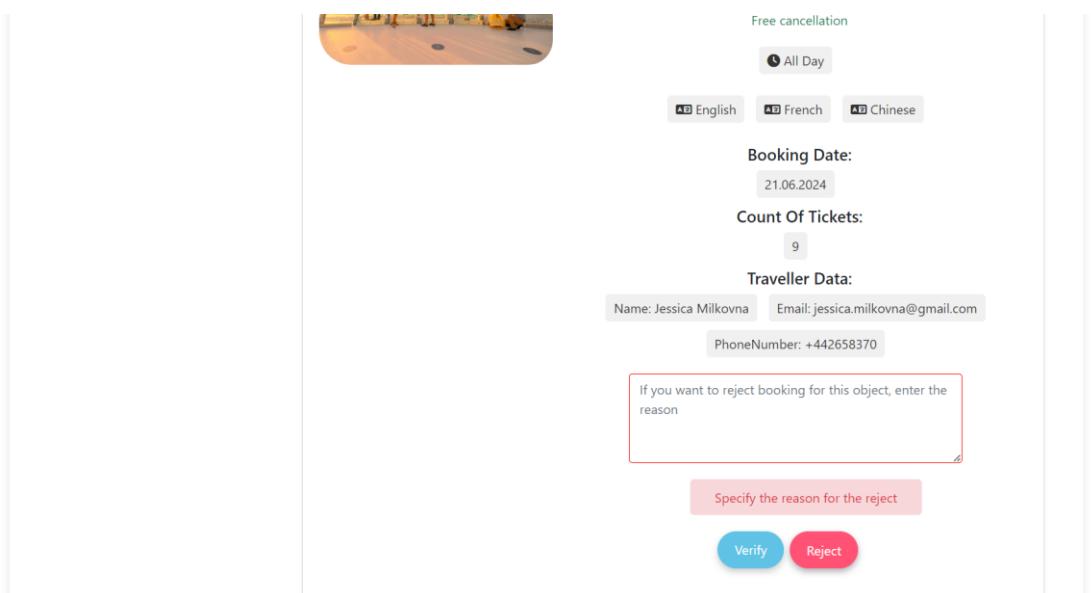


Рисунок 26 – Обработка некорректного отклонения брони

Все подтвержденные владельцем бронирования с данными пользователей, которые их забронировали, располагаются на странице “Verified Bookings” (рис. 27).

Рисунок 27 – Страница Verified Bookings

Все отклоненные бронирования соответственно располагаются на странице “Rejected Bookings” (рис. 28).

Рисунок 28 – Страница Rejected Bookings

Рассмотрим возможности, которые предоставлены для роли “Admin”. На рисунке 29 представлен основной функционал администратора.

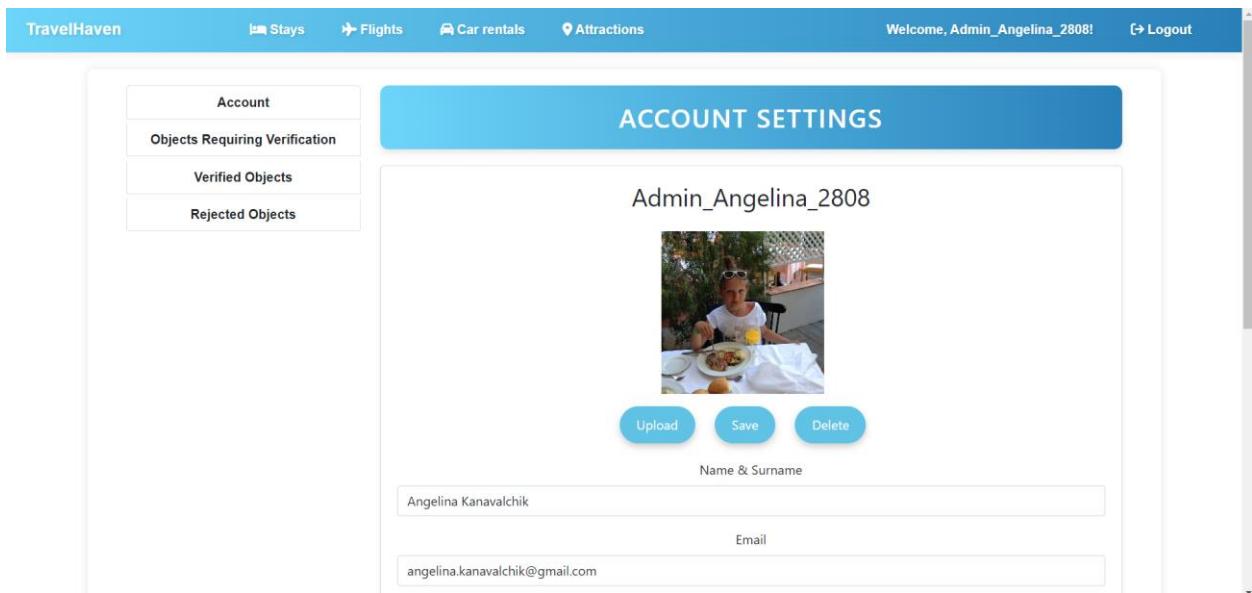


Рисунок 29 – Функциональные возможности администратора

Объекты, которые были предложены различными владельцами компаний и ожидающие подтверждение со стороны админа, располагаются на странице “Objects Requiring Verification” (рис. 30). Обработка некорректной причины отклонения объекта также предусмотрена.

Рисунок 30 – Страница объектов, ждающих подтверждение/отклонение админа

Все подтвержденные админом объекты располагаются на странице “Verified Objects” (рис. 31). Админ имеет возможность удалить объект с сайта.

The screenshot shows the 'Verified Objects' section of the TravelHaven platform. On the left, a sidebar menu includes 'Account', 'Objects Requiring Verification' (which is currently selected), 'Verified Objects', and 'Rejected Objects'. The main area is titled 'VERIFIED OBJECTS' and lists two items:

- BMW M5** (Florida)
 - Price per day: \$90
 - Free cancellation
 - Transmission: Manual
 - Seating: 4 seats
- Mercedes-Benz G-Class** (Florida)
 - Price per day: \$32
 - Transmission: Automatic
 - Seating: 100 seats

Рисунок 31 – Страница подтвержденных объектов

Страница отклоненных админом объектов структурирована подобным образом. Указывается причина отклонения. В случае, если админ передумал, он может подтвердить этот объект или вовсе удалить (рис. 32).

The screenshot shows the 'REJECTED OBJECTS' section of the TravelHaven platform. On the left, a sidebar menu includes 'Account', 'Objects Requiring Verification' (which is currently selected), 'Verified Objects', and 'Rejected Objects'. The main area is titled 'REJECTED OBJECTS' and lists two items:

- Mercedes** (Barcelona)
 - Price per day: \$105
 - Unlimited Mileage
 - Automatic
 - 4 seats

Cause of Reject:
Just I dont like this caaaaaaaaaarrrrrrrrrr
- Rixos Premium JBR Hotel** (Dubai, UAE)
 - 4 swimming pool
 - 5 restaurant
 - Parking
 - Private Beach

Рисунок 32 – Страница отклоненных объектов

Рассмотрим возможности, предоставляемые для роли “Traveller”. На рисунке 33 представлен основной функционал указанной роли.

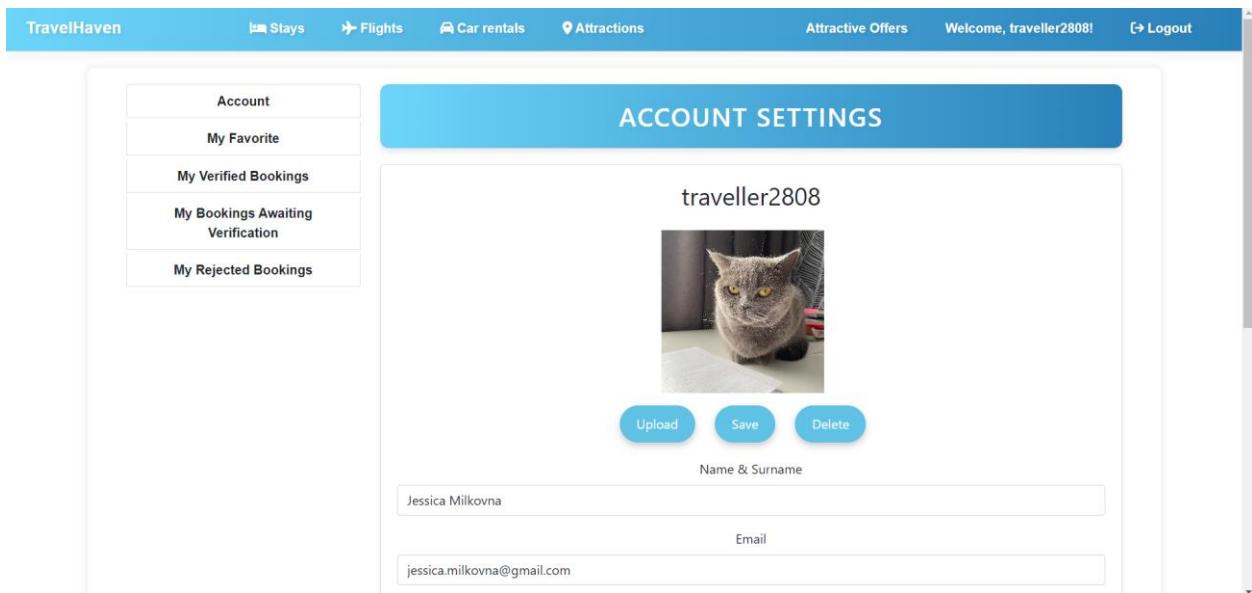


Рисунок 33 – Основной функционал роли “Traveller”

Бронирования, сделанные данным пользователем и ждущие подтверждения/отклонения со стороны владельца объекта, находятся на странице “My Bookings Awaiting Verification” (рис. 34). Выводятся данные о бронировании, а также есть возможность отменить текущее бронирование.

| Car Model | Location | Price per day | Booking Dates | Action |
|-----------|-----------|---------------|-------------------------------------|--------------------------------|
| BMW M5 | Florida | \$90 | 03.06.2024 05:46 - 07.06.2024 02:46 | Cancel Booking |
| Red Tesla | Barcelona | \$310 | | |

Рисунок 34 – Страница бронирований, ожидающих подтверждения/отклонения

Страница с подтвержденными бронированиями обладает аналогичным функционалом (рис. 35).

TravelHaven

Stays Flights Car rentals Attractions Attractive Offers Welcome, traveller2808! Logout

Account My Favorite My Verified Bookings My Bookings Awaiting Verification My Rejected Bookings

MY VERIFIED BOOKINGS

BMW M5
Florida
Price per day: \$90
Free cancellation
Manual 4 seats

Booking Dates:
31.05.2024 10:24 - 31.05.2024 12:24

Cancel Booking

Business Class
Free cancellation 19:35 On the way: 6h 6m 01:41

Рисунок 35 – Страница подтвержденных бронирований

На странице с отклоненными бронированиями каждый объект предусматривает причину отклонения (рис. 36).

TravelHaven

Stays Flights Car rentals Attractions Attractive Offers Welcome, traveller2808! Logout

Account My Favorite My Verified Bookings My Bookings Awaiting Verification My Rejected Bookings

MY REJECTED BOOKINGS

Mercedes-Benz G-Class
Florida
Price per day: \$32
Automatic 100 seats

Booking Dates:
30.05.2024 10:24 - 31.05.2024 10:24

Cause of Reject:
This car has a new price

The View at The Palm | Dubai Skyline Marvel
Dubai, UAE
Price per 1 person: \$70

Рисунок 36 – Страница отклоненных бронирований

Особенным функционалом для пользователей с ролью “Traveller” является добавление понравившихся объектов в категорию «Любимые» с возможностью последующего бронирования (если это возможно) или удалением текущего объекта из данной категории (рис. 37).

The screenshot shows the 'FAVORITE ITEMS' section of the TravelHaven website. On the left, a sidebar menu includes 'Account', 'My Favorite', 'My Verified Bookings', 'My Bookings Awaiting Verification', and 'My Rejected Bookings'. The main area displays two favorite items:

- Range Rover Black** (Moscow): Price per day: \$240. Includes options for Unlimited Mileage, Automatic transmission, and 5 seats. Selected Dates: 14.06.2024 04:39 - 15.06.2024 04:40. A 'Book' button is present.
- Red Tesla** (Barcelona): Price per day: \$310. Includes 'Free cancellation'. A 'Book' button is present.

Рисунок 37 – Страница объектов в категории «Любимые»

В случае, если бронирование любимого объекта невозможно, пользователь будет проинформирован и кнопка Book станет некликабельной (рис. 38).

The screenshot shows the 'Red Tesla' listing from Figure 37. A message at the bottom states: 'Unfortunately, this car is not available for booking on the specified dates'. Below this, a flight section is shown:

| Business Class | |
|---|----------------------------------|
| Free cancellation | |
| \$502 | 14:32 Minsk Friday, July 5 MSQ |
| Luggage <input checked="" type="checkbox"/> | On the way: 7h 1m |
| | 21:33 Antalya Friday, July 5 AYT |

Рисунок 38 – Обработка возможности бронирования любимого объекта

Чтобы удалить объект из категории «Любимые», необходимо кликнуть по иконке сердечка, после чего она перестанет быть заполненной и объект успешно удалится из категории.

С целью сделать процесс планирования путешествий более персонализированным была создана система рекомендаций на основании объектов, добавленных в категорию «Любимые». Ее суть заключается в

нахождении самых выгодных предложений для брони всех видов объектов в городах, в которых располагаются любимые объекты пользователя.

Для начала пользователю предлагается выбрать категорию объектов, для которой он бы хотел посмотреть выгодные предложения (рис. 39).

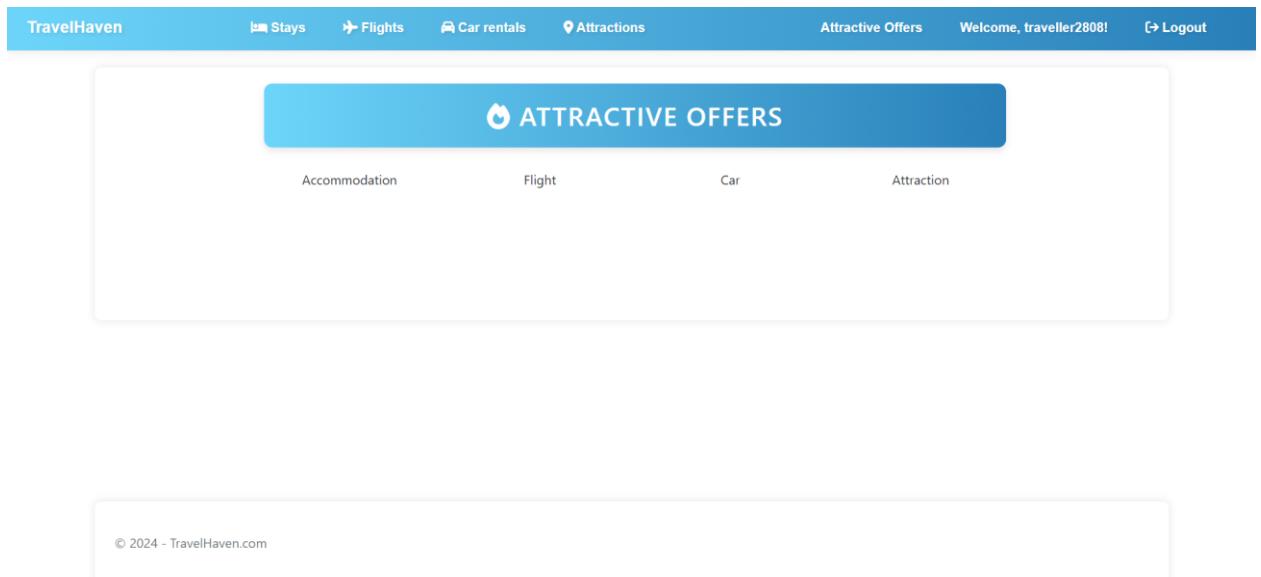


Рисунок 39 – Стартовая страница системы рекомендаций

После выбора конкретного вида объекта выводится список самых выгодных предложений с возможностью бронирования, а также добавления объекта в категорию «Любимые» (рис. 40).

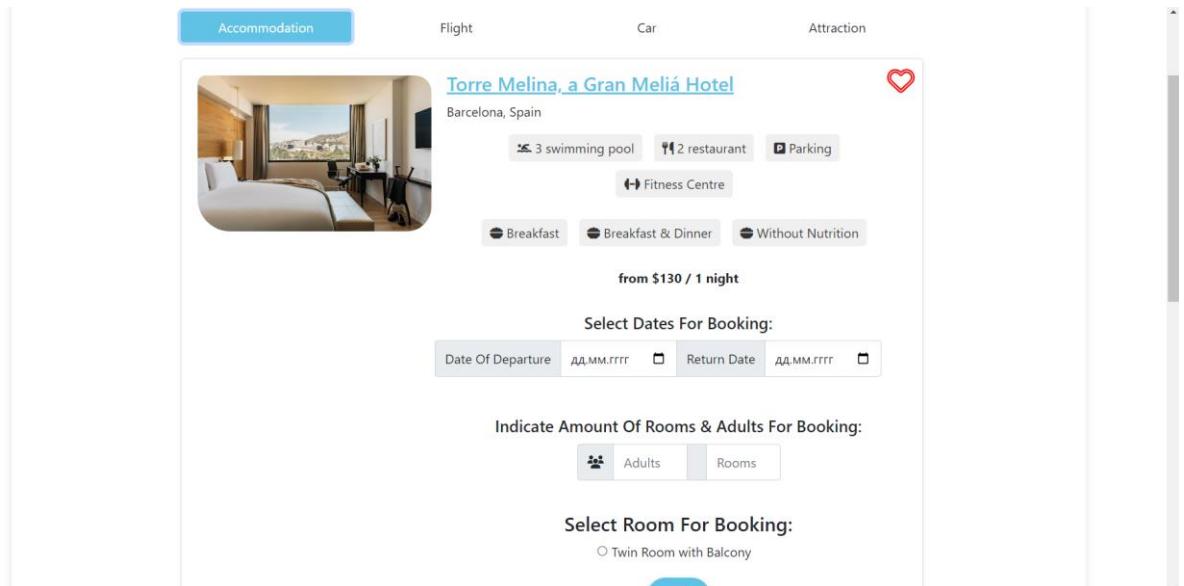


Рисунок 40 – Выгодные предложения для объектов жилья

Прежде, чем забронировать объект или добавить его в категорию «Любимые», необходимо заполнить требуемые данные. Обработка некорректного заполнения данных также предусмотрена (рис. 41).

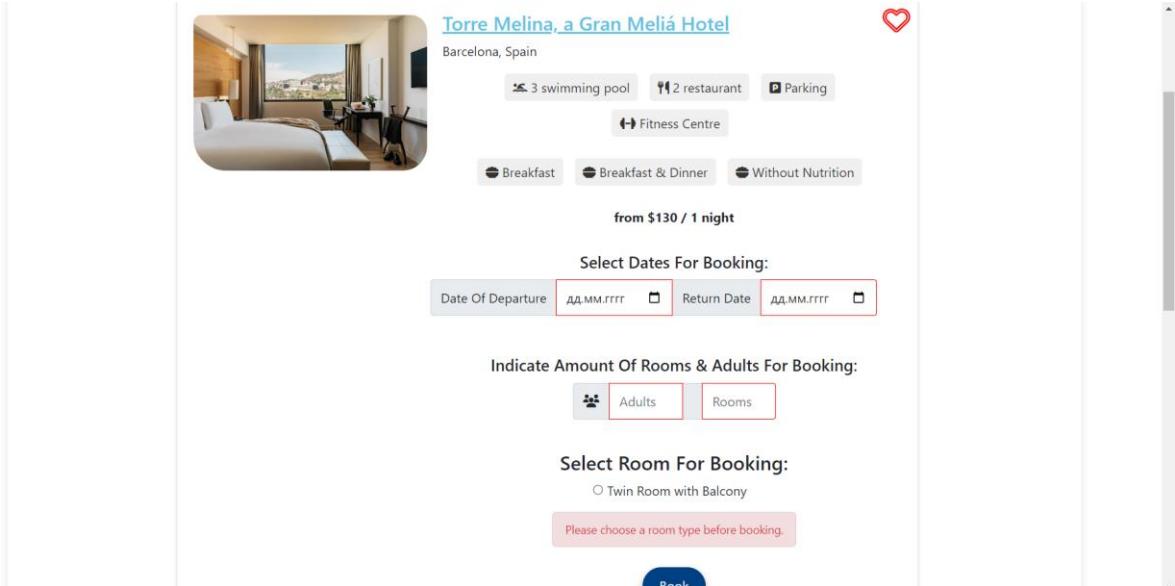


Рисунок 41 – Обработка некорректного ввода данных

Страница регистрации продемонстрирована на рисунке 42.

Please register

UserName
traveller2008

Password

Confirm Password

Select your account ▾

Register

© 2024 - TravelHaven.com

Рисунок 42 – Страница регистрации

Обработка некорректно введенных данных также присутствует (рис. 43).

TravelHaven

Stays Flights Car rentals Attractions

Please register

• Password is not correct

UserName
traveller2808

Password

Confirm Password

Select your account ▾

Register

This screenshot shows the registration page of the TravelHaven website. At the top, there's a navigation bar with links for Stays, Flights, Car rentals, and Attractions. Below the navigation is a form for creating a new account. The 'UserName' field contains 'traveller2808'. The 'Password' field contains '*****'. A red error message 'Password is not correct' is displayed above the 'Confirm Password' field. Below the form is a button labeled 'Select your account ▾' and a large blue 'Register' button.

© 2024 - TravelHaven.com

Рисунок 43 – Обработка некорректного ввода данных

Страница входа проиллюстрирована на рисунке 44.

TravelHaven

Stays Flights Car rentals Attractions

Please sign in

User Name
traveller2808

Password

Remember me

[Create new account](#)

Sign in

This screenshot shows the login page of the TravelHaven website. At the top, there's a navigation bar with links for Stays, Flights, Car rentals, and Attractions. Below the navigation is a form for logging in. The 'User Name' field contains 'traveller2808'. The 'Password' field contains '*****'. There is a checkbox for 'Remember me' and a link for 'Create new account'. Below the form is a large blue 'Sign in' button.

© 2021–2024

© 2024 - TravelHaven.com

Рисунок 44 – Страница входа

ЗАКЛЮЧЕНИЕ

В ходе данной курсовой работы было разработано веб-приложение для организации путешествий «TravelHaven» на языке программирования C# с применением концепций объектно-ориентированного программирования. Были использованы принципы ООП, а также принципы SOLID и паттерны проектирования. Приложение позволяет авторизованным пользователям бронировать жилье, авиабилеты, автомобили и различные виды активностей. Владельцы компаний могут размещать свои объекты для бронирования. Особое внимание было уделено вопросам удобства использования и безопасности данных, что позволяет обеспечить высокий уровень удовлетворенности пользователей.

Перспективы дальнейшего развития веб-приложения "TravelHaven" включают внедрение новых функций, направленных на повышение удобства и выгодности использования. Одной из таких функций является система скидок, основанная на количестве ранее забронированных объектов пользователем. Это позволит не только вознаграждать лояльных пользователей, но и стимулировать их к повторным бронированиям.

Кроме того, планируется реализация усовершенствованной системы фильтрации по свойствам объектов во время поиска доступных бронирований. Это позволит пользователям быстрее находить именно те варианты, которые наиболее полно соответствуют их требованиям и предпочтениям, повышая общую эффективность использования приложения.

В ходе работы над проектом я изучила специфику языка программирования C# и технологию фреймворка ASP.NET Core, приобрела навыки работы с базами данных, используя MS SQL Server и Entity Framework Core для управления данными. На практике был применен паттерн MVC, что позволило разделить приложение на независимые компоненты, облегчая их тестирование и поддержку.

Итоговый продукт удовлетворяет как техническим, так и функциональным требованиям. Был реализован интуитивно понятный и удобный пользовательский интерфейс. Новый функционал можно легко добавлять благодаря модульной архитектуре и применению принципов ООП, что позволяет улучшать и расширять возможности системы без необходимости переписывать или усложнять существующий код.

Таким образом, данная курсовая работа демонстрирует возможности и преимущества использования современных технологий и ООП-подходов при разработке веб-приложения для организации путешествий. Полученные навыки и опыт помогут мне в будущем создавать более качественные и функциональные программные продукты.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

- [1] ГОСТ 19.201-78. Единая система программной документации. Техническое задание. Требования к содержанию и оформлению [Электронный ресурс]. – Режим доступа: <https://internet-law.ru/gosts/gost/31884/>.
- [2] Методология разработки [Электронный ресурс]. – Режим доступа: <https://scrumtrek.ru/blog/agile-scrum/4029/metodologiya-agile/>.
- [3] Сайт о программировании [Электронный ресурс]. – Режим доступа: <https://metanit.com/sharp/mvc.php>.
- [4] Документация по интерфейсной веб-платформе .NET – ASP.NET Core Blazor [Электронный ресурс]. – Режим доступа: <https://learn.microsoft.com/ru-ru/aspnet/core/blazor/>.
- [5] EntityFramework: (анти)паттерн Repository [Электронный ресурс]. – Режим доступа: <https://habr.com/ru/articles/335856/>
- [6] Документация по интерфейсной веб-платформе ASP.NET – Веб-приложения – Blazor – [Электронный ресурс]. – Режим доступа: <https://dotnet.microsoft.com/ru-ru/apps/aspnet/web-apps/blazor>
- [7] Создание веб-приложений с помощью Blazor – [Электронный ресурс]. – Режим доступа: <https://learn.microsoft.com/ru-ru/training/paths/build-web-apps-with-blazor/>
- [8] Документация по ASP.NET Core Identity – [Электронный ресурс]. – Режим доступа: <https://metanit.com/sharp/aspnet5/16.1.php?ysclid=lx1sv845dg343582556>
- [9] Статья о Microsoft identity platform – [Электронный ресурс]. – Режим доступа: <https://learn.microsoft.com/en-us/entra/identity-platform/v2-overview>

ПРИЛОЖЕНИЕ А

Исходный текст кода для визуализации страницы Stays

Index.cshtml

```
@{  
    ViewData["Title"] = "TravelHaven.com | Official site | The best experience";  
}  
  
@using Travelling_Application.ViewModels  
@model SearchAccomodationViewModel  
  
<head>  
    <script src="https://code.jquery.com/jquery-3.6.0.min.js"></script>  
    <style>  
        .full-width-image {  
            width: 100%;  
            height: 10cm; /* Задаем высоту изображения */  
            background-image: url('https://kartinki.pics/pics/uploads/posts/2022-  
08/thumbs/1660697156_49-kartinkin-net-p-maldivi-oboi-krasivo-58.jpg'); /* Укажите путь к  
вашему изображению */  
            background-size: cover;  
            background-position: center;  
            position: relative;  
        }  
  
        .accommodation-card {  
            position: relative;  
            padding: 15px;  
            border: 1px solid #ddd;  
            border-radius: 8px;  
            margin-bottom: 20px;  
        }  
  
        .accommodation-image {  
            width: 100%;  
            height: auto;  
            display: block;  
            margin-top: 20px;  
            margin-left: 20px;  
            border-radius: 10px 20px 30px 40px;  
        }  
  
        .accommodation-details {  
            padding: 15px;  
        }  
  
        .accommodation-title {  
            margin-top: 0;  
        }  
    </style>
```

```
.accomodation-price {  
    font-weight: bold;  
}  
  
.mini-info {  
    margin-top: 5px;  
}  
  
.mini-info-item {  
    display: inline-block;  
    margin-right: 10px;  
    padding: 5px 10px;  
    background-color: #f0f0f0;  
    color: #333;  
    border-radius: 5px;  
    margin-bottom: 10px;  
}  
  
.invalid-field {  
    border-color: red !important;  
}  
  
.content {  
    margin-top: -0.6cm; /* Устанавливаем отступ сверху, чтобы контент начинался  
после изображения */  
}  
  
.no-results-message {  
    text-align: center;  
    padding: 20px;  
    margin-top: 20px;  
    border: 1px solid #ddd;  
    border-radius: 5px;  
    background-color: #f8f9fa;  
    color: #333;  
    font-size: 1.2rem;  
    box-shadow: 0 2px 4px rgba(0, 0, 0, 0.1);  
}  
  
.text-danger {  
    background-color: #f8d7da;  
    color: #721c24;  
    border-color: #f5c6cb;  
}  
  
.notification {  
    display: none;  
    padding: 10px;  
    margin-top: 10px;  
    border-radius: 5px;
```

```

        }

.favorite-icon {
    position: absolute;
    top: 10px;
    right: 10px;
}

.favorite-icon i {
    cursor: pointer;
    font-size: 2rem;
    color: red;
}

.favorite-icon .fa-heart-o {
    color: transparent;
    -webkit-text-stroke: 2px red;
}

.favorite-icon .fa-heart {
    color: red;
}

</style>

<script>
function validateButtons(accomId) {
    const radios = document.getElementsByName('room');
    let isChecked = false;
    for (const radio of radios) {
        if (radio.checked) {
            isChecked = true;
            break;
        }
    }
    if (!isChecked) {
        $('#notification_' + accomId).text('Please choose room for booking.')
            .addClass('text-danger')
            .addClass('notification')
            .show();
        return false; // Prevent form submission
    }

    $('#notification_' + accomId).text('Please choose room for booking.')
        .removeClass('text-danger')
        .hide();

    return true; // Allow form submission
}

function validateSearch() {
    var city = $('#citySelect').val();
    var dateIn = $('#checkInDate').val();
    var dateOut = $('#checkOutDate').val();
    var adults = $('#adults').val();
}

```

```

var rooms = $('#rooms').val();

var isValid = true;

if (!city) {
    $('#citySelect').addClass('invalid-field');
    isValid = false;
} else {
    $('#citySelect').removeClass('invalid-field');
}

if (!dateIn) {
    $('#checkInDate').addClass('invalid-field');
    isValid = false;
} else {
    $('#checkInDate').removeClass('invalid-field');
}

if (!dateOut) {
    $('#checkOutDate').addClass('invalid-field');
    isValid = false;
} else {
    $('#checkOutDate').removeClass('invalid-field');
}

if (!adults) {
    $('#adults').addClass('invalid-field');
    isValid = false;
} else {
    $('#adults').removeClass('invalid-field');
}

if (!rooms) {
    $('#rooms').addClass('invalid-field');
    isValid = false;
} else {
    $('#rooms').removeClass('invalid-field');
}

if (dateIn && dateOut) {
    var checkInDate = new Date(dateIn);
    var checkOutDate = new Date(dateOut);

    if (checkInDate >= checkOutDate) {
        $('#checkInDate').addClass('invalid-field');
        $('#checkOutDate').addClass('invalid-field');
        isValid = false;
    } else {
        $('#checkInDate').removeClass('invalid-field');
        $('#checkOutDate').removeClass('invalid-field');
    }
}

```

```

        return isValid;
    }

</script>
</head>

<div class="full-width-image"></div>

<main class="form-signin w-100 m-auto text-center" style="padding-bottom: 30px;">
    <body>
        <div class="container content">
            <form id="searchAccomodationForm" method="get"
                action="/Home/AccomodationResults" onsubmit="return validateSearch()">
                <div class="row justify-content-center">
                    <div class="col-md-3">
                        @{
                            var selectedCity = ViewBag.SelectedCity as string;
                        }
                        <div class="input-group mb-3" style="height: 45px;">
                            <span class="input-group-text"><i class="fa-solid fa-bed"></i></span>
                            <select class="form-select" id="citySelect" name="city">
                                <option value="" selected hidden>Where are you going?</option>
                                @foreach (var city in Model.Cities)
                                {
                                    <option value="@city">@city</option>
                                }
                            </select>
                        </div>
                        <script>
                            document.addEventListener("DOMContentLoaded", function () {
                                var selectedCity =
                                    '@Html.Raw(System.Web.HttpUtility.JavaScriptStringEncode(selectedCity))';
                                var selectElement = document.getElementById('citySelect');
                                selectElement.value = selectedCity;
                            });
                        </script>
                    </div>
                    <div class="col-md-6">
                        <div class="input-group mb-4" style="height: 45px;">
                            <span class="input-group-text">Check-In Date</span>
                            <input type="date" class="form-control" id="checkInDate"
                                name="checkInDate" value="@ViewBag.CheckInDate?.ToString("yyyy-MM-dd")"
                                min="@DateTime.Now.ToString("yyyy-MM-dd")">
                            <span class="input-group-text">Check-Out Date</span>
                            <input type="date" class="form-control" id="checkOutDate"
                                name="checkOutDate" value="@ViewBag.CheckOutDate?.ToString("yyyy-MM-dd")"
                                min="@DateTime.Now.ToString("yyyy-MM-dd")">
                        </div>
                    </div>
                    <div class="col-md-3">

```

```

<div class="input-group mb-3">
    <span class="input-group-text"><i class="fas fa-users"></i></span>
    <input type="number" class="form-control" id="adults" name="adults"
placeholder="Adult" min="1" max="50" value="@ViewBag.Adults">
    <input type="number" class="form-control" id="rooms" name="rooms"
placeholder="Room" min="1" max="30" value="@ViewBag.Rooms">
    <button type="submit" class="btn btn-primary">Search</button>
</div>
</div>
</div>
</form>

<div class="container-fluid" style="margin-left: 200px;">
<div class="row">
<div class="col-md-9">
    <div class="row">
        <div class="col">
            @if (Model?.Results != null && Model.Results.Any())
            {
                @foreach (var accomodation in Model.Results)
                {
                    <form id="bookAccomodation_@accomodation.Id" method="post"
action="/Home/BookAccomodation" onsubmit="return validateButtons(@accomodation.Id)""
class="bookAccomodationForm">
                        <input type="hidden" name="accomodationId"
value="@accomodation.Id" />
                        <input type="hidden" name="dateIn"
value="@ViewBag.CheckInDate?.ToString("yyyy-MM-dd")" />
                        <input type="hidden" name="dateOut"
value="@ViewBag.CheckOutDate?.ToString("yyyy-MM-dd")" />
                        <input type="hidden" name="rooms" value="@ViewBag.Rooms" />
                        <input type="hidden" name="adults" value="@ViewBag.Adults" />
                    <div class="accomodation-card">
                        <div class="row">
                            <div class="col-md-4">
                                
                            </div>
                            <div class="col-md-8">
                                <div class="accomodation-details">
                                    @{
                                        if (User.IsInRole("traveller") ||
User.IsInRole("Traveller"))
                                    {
                                        <div class="favorite-icon">
                                            <i class="fa @("fa-heart-o")" data-
accomodation-id="@accomodation.Id" style="cursor: pointer;"></i>

```

```

                </div>
            }
        }
    <h3 class="accomodation-title" style="margin-top: 0;
font-size: 1.5rem; text-align: left;">
    <a href="/Account>ShowAccomodationInformation?accomodationId=@accomodation.Id">@acco
modation.Name</a>
    </h3>
    <p class="accomodation-features" style="text-align:
left;">@accomodation.City, @accomodation.Country</p>
    <p class="mini-info">
        @{
            if (accomodation.SwimmingPool > 0)
            {
                <span class="mini-info-item"><i class="fa-solid fa-person-swimming"></i> @accomodation.SwimmingPool swimming pool</span>
            }
            if (accomodation.Restaurants > 0)
            {
                <span class="mini-info-item"><i class="fa-solid fa-utensils"></i> @accomodation.Restaurants restaurant</span>
            }
            if (accomodation.Parking)
            {
                <span class="mini-info-item"><i class="fa-solid fa-square-parking"></i> Parking</span>
            }
            if (accomodation.PrivateBeach)
            {
                <span class="mini-info-item"><i class="fa-solid fa-umbrella-beach"></i> Private Beach</span>
            }
            if (accomodation.SPA)
            {
                <span class="mini-info-item"><i class="fa-solid fa-spa"></i> SPA</span>
            }
            if (accomodation.FitnessCentre)
            {
                <span class="mini-info-item"><i class="fa-solid fa-dumbbell"></i> Fitness Centre</span>
            }
            if (accomodation.PetsAllowed)
            {
                <span class="mini-info-item"><i class="fa-solid fa-paw"></i> Pets Allowed</span>
            }
        }
    </p>
    <p class="mini-info">

```

```

        @{
            foreach (var nutrition in
accomodation.TypesOfNutrition)
{
    <span class="mini-info-item"><i class="fa-solid fa-bowl-food"></i> @nutrition</span>
}
}
</p>
<p class="accomodation-price">from
$@accomodation.MinCost / 1 night</p>

        @{
            if (User.IsInRole("traveller") ||
User.IsInRole("Traveller"))
{
    <h4>Choose Room For Booking:</h4>
    <input type="hidden" id="selectedRoom"
name="selectedRoom" value="" />

    <div class="available-rooms" style="margin-
bottom: 10px;">
        @foreach (var roomName in
accomodation.AvailableRoomsNames)
{
    <label class="radio-inline" style="margin-
right: 10px;">
        <input type="radio" name="room"
value="@roomName" onclick="updateSelectedRoom('@roomName')"/> @roomName
        </label>
}
</div>

    <div id="notification_@accomodation.Id" class="text-
danger" style="display:none;"></div>

    <div id="favoriteNotification_@accomodation.Id"
class="text-danger" style="display:none;"></div>

        <button id="bookButton_@accomodation.Id"
type="submit" class="btn btn-primary" style="margin-right: 20px; margin-top:
10px;">Book</button>
    }
}
</div>
</div>
</div>
</form>
}
}

```

```

        else
        {
            <div class="no-results-message">
                <p>No accommodation found for your search criteria. Please try
again with different options.</p>
            </div>
        }
    </div>
    </div>
</div>
</div>
</div>
</div>
</div>
</div>
</div>
</div>
</div>
</div>
@section Scripts {
<script>
$(document).ready(function () {
    $('#bookAccommodationForm').submit(function (event) {
        event.preventDefault();

        if (!validateButtons()) {
            return false;
        }

        var form = $(this);
        var formData = form.serialize();

        $.ajax({
            type: 'POST',
            url: $(this).attr('action'),
            data: formData,
            success: function (response) {
                var buttonId = form.find('button[type=submit]').attr('id');
                var button = $('#' + buttonId);
                button.css('background-color', 'green');
                button.text('Booked');
                button.prop('disabled', true);
            },
            error: function (xhr, status, error) {
                console.error('Ошибка: ' + error);
            }
        });
    });
});

$('.favorite-icon i').click(function () {
    var $icon = $(this);
    var accommodationId = $icon.data('accommodation-id');
    var dateIn = $('#checkInDate').val();
    var dateOut = $('#checkOutDate').val();
    var rooms = $('#rooms').val();
    var selectedRoom = $('input[name="room"]:checked').val(); // Get the selected
room value
});
```

```

        // Check if room type is selected
        if (!selectedRoom) {
            $('#favoriteNotification_' + accommodationId).text('Please choose a room type
before adding to favorites.').addClass('text-danger').addClass('notification').show();
            return; // Stop execution if no room is selected
        }

        $('#favoriteNotification_' + accommodationId).text('Please choose a room type
before adding to favorites.').removeClass('text-danger').removeClass('notification').hide(); //
Hide error message if room is selected

$.ajax({
    url: '/Home/AddToFavoriteAccomodation',
    type: 'POST',
    data: {
        accomodationId: accommodationId,
        dateIn: dateIn,
        dateOut: dateOut,
        rooms: rooms,
        selectedRoom: selectedRoom
    },
    success: function (response) {
        if (response.isFavorite) {
            $icon.removeClass('fa-heart-o').addClass('fa-heart');
        } else {
            $icon.removeClass('fa-heart').addClass('fa-heart-o');
        }
    },
    error: function () {
        alert('There was an error updating your favorites. Please try again.');
    }
});
});

function changeButton(button) {
    if (validateButtons()) {
        // Change button color
        button.style.backgroundColor = 'green';
        // Change button text
        button.innerText = 'Booked';
        button.disabled = true;
        return true; // Allow form submission
    }
    return false; // Prevent form submission
}
</script>

<script>

```

```
function updateSelectedRoom(roomName) {  
    document.getElementById('selectedRoom').value = roomName;  
}  
</script>  
}  
</body>  
</main>
```

ПРИЛОЖЕНИЕ Б

Исходный код

AccountController.cs

```
[HttpPost]
public async Task<IActionResult> SaveAccomodationForm(Accomodation model, string
photoDataAccomodation)
{
    var photos = JsonConvert.DeserializeObject<List<string>>(photoDataAccomodation);
    var photoBytesList = photos.Select(photoBase64 =>
Convert.FromBase64String(photoBase64.Split(',')[1])).ToList();
    var currentUser = await _context.Users.FirstOrDefaultAsync(u => u.UserName ==
User.Identity.Name);

    var entities = new List<EntityModel>
    {
        new EntityModel { Name = "Accommodation", Description = "Description for
Accommodation" },
        new EntityModel { Name = "Flight", Description = "Description for Flight" },
        new EntityModel { Name = "Car", Description = "Description for Car" },
        new EntityModel { Name = "Attraction", Description = "Description for Attraction" }
    };

    var accommodation = new Accomodation
    {
        Name = model.Name,
        City = model.City,
        Country = model.Country,
        TypesOfNutrition = model.TypesOfNutrition,
        Address = model.Address,
        TypeOfAccomodation = model.TypeOfAccomodation,
        Description = model.Description,
        Parking = model.Parking,
        SwimmingPool = model.SwimmingPool,
        FreeWIFI = model.FreeWIFI,
        PrivateBeach = model.PrivateBeach,
        LineOfBeach = model.LineOfBeach,
        Restaurants = model.Restaurants,
        SPA = model.SPA,
        Bar = model.Bar,
        Garden = model.Garden,
        TransferToAirport = model.TransferToAirport,
        TactileSigns = model.TactileSigns,
        SmookingRooms = model.SmookingRooms,
        FamilyRooms = model.FamilyRooms,
        CarChargingStation = model.CarChargingStation,
        WheelchairAccessible = model.WheelchairAccessible,
        FitnessCentre = model.FitnessCentre,
        PetsAllowed = model.PetsAllowed,
```

```

DeliveryFoodToTheRoom = model.DeliveryFoodToTheRoom,
EveryHourFrontDesk = model.EveryHourFrontDesk,
MainPhoto = photoBytesList[0],
VerifiedByAdmin = false,
RejectedByAdmin = false,
PublisherId = currentUser.Id,
AvailableRoomsNames = new List<string>(),
MinCost = 0,
RejectedMessage = string.Empty
};

string filePath = "C:\\Users\\Angelina\\Pictures\\Camera
Roll\\bbb846030d108b92a3fefbfca1f7bbe6.jpg";
byte[] fileBytes = System.IO.File.ReadAllBytes(filePath);
ViewBag.CarPhoto = fileBytes;
// добавить сохранение dat в другую БД
_context.Accomodation.Add(accommodation);
await _context.SaveChangesAsync();

foreach (var photo in photoBytesList)
{
    var photoss = new AccomodationPhotos
    {
        ObjectId = accommodation.Id,
        PhotoArray = photo,
    };
    _context.AccomodationPhotos.Add(photoss);
    await _context.SaveChangesAsync();
}

// Устанавливаем сообщение об успешном изменении в TempData
TempData["SuccessMessage"] = "The object Accommodation was successfully saved and
sent for moderation.';

return View("NewObject", entities);
}

public async Task<IActionResult> VerifiedBookings()
{
    var currentUser = await _context.Users.FirstOrDefaultAsync(u => u.UserName ==
User.Identity.Name);

    var unverifiedCars = await _context.BookingCars
.Where(c => c.VerifiedBooking && !c.CanceledBooking)
.ToListAsync();

    var unverifiedAirTickets = await _context.BookingAirTickets
.Where(c => c.VerifiedBooking && !c.CanceledBooking)
.ToListAsync();
}

```

```

var unverifiedAttractions = await _context.BookingAttractions
    .Where(c => c.VerifiedBooking && !c.CanceledBooking)
    .ToListAsync();

var unverifiedAccomodation = await _context.BookingAccomodations
    .Where(c => c.VerifiedBooking && !c.CanceledBooking)
    .ToListAsync();

var myVerifiedCars = new List<BookingCar>();
var myVerifiedAirTickets = new List<BookingAirTicket>();
var myVerifiedAttractions = new List<BookingAttraction>();
var myVerifiedAccomodation = new List<BookingAccomodation>();

foreach (var car in unverifiedCars)
{
    var exists = _context.Car.Any(c => c.Id == car.CarId && c.PublisherId == currentUser.Id);

    if (exists)
    {
        // Add to the verifiedCars list
        myVerifiedCars.Add(car);
    }
}

foreach (var airticket in unverifiedAirTickets)
{
    var exists = _context.AirTicket.Any(c => c.Id == airticket.AirTicketId && c.PublisherId == currentUser.Id);

    if (exists)
    {
        // Add to the verifiedCars list
        myVerifiedAirTickets.Add(airticket);
    }
}

foreach (var attraction in unverifiedAttractions)
{
    var exists = _context.Entertainment.Any(c => c.Id == attraction.AttractionId && c.PublisherId == currentUser.Id);

    if (exists)
    {
        // Add to the verifiedCars list
        myVerifiedAttractions.Add(attraction);
    }
}

foreach (var accomodation in unverifiedAccomodation)
{
}

```

```

    var exists = _context.Accomodation.Any(c => c.Id == accomodation.AccomodationId &&
c.PublisherId == currentUser.Id);

    if (exists)
    {
        // Add to the verifiedCars list
        myVerifiedAccomodation.Add(accomodation);
    }
}

var cars = new List<Car>();
var usersDataCars = new List<UserData>();

foreach (var car in myVerifiedCars)
{
    var currentCar = _context.Car.Find(car.CarId);
    cars.Add(currentCar);
    var user = _context.Users.Find(car.UserId);
    var userData = new UserData();
    userData.Name = user.Name;
    userData.Email = user.Email;
    userData.PhoneNumber = user.CountryCode + user.PhoneNumber;
    usersDataCars.Add(userData);
}

var airtickets = new List<AirTicket>();
var usersDataAirTickets = new List<UserData>();

foreach (var airticket in myVerifiedAirTickets)
{
    var currentAirticket = _context.AirTicket.Find(airticket.AirTicketId);
    airtickets.Add(currentAirticket);
    var user = _context.Users.Find(airticket.UserId);
    var userData = new UserData();
    userData.Name = user.Name;
    userData.Email = user.Email;
    userData.PhoneNumber = user.CountryCode + user.PhoneNumber;
    usersDataAirTickets.Add(userData);
}

var attractions = new List<Entertainment>();
var usersDataAttractions = new List<UserData>();

foreach (var attraction in myVerifiedAttractions)
{
    var currentAttraction = _context.Entertainment.Find(attraction.AttractionId);
    attractions.Add(currentAttraction);
    var user = _context.Users.Find(attraction.UserId);
    var userData = new UserData();
    userData.Name = user.Name;
    userData.Email = user.Email;
}

```

```

        userData.PhoneNumber = user.CountryCode + user.PhoneNumber;
        usersDataAttractions.Add(userData);
    }

    var accomodations = new List<Accomodation>();
    var usersDataAccomodation = new List<UserData>();

    foreach (var accomodation in myVerifiedAccomodation)
    {
        var currentAccomodation = _context.Accomodation.Find(accomodation.AccomodationId);
        accomodations.Add(currentAccomodation);
        var user = _context.Users.Find(accomodation.UserId);
        var userData = new UserData();
        userData.Name = user.Name;
        userData.Email = user.Email;
        userData.PhoneNumber = user.CountryCode + user.PhoneNumber;
        usersDataAccomodation.Add(userData);
    }

    var model = new ForVerificationObjectsViewModel
    {
        Cars = cars,
        AirTickets = airtickets,
        Attractions = attractions,
        Accomodation = accomodations,
        BookingCars = myVerifiedCars,
        BookingAirTickets = myVerifiedAirTickets,
        BookingAttractions = myVerifiedAttractions,
        BookingAccomodation = myVerifiedAccomodation,
        UsersDataAttractions = usersDataAttractions,
        UsersDataAirTickets = usersDataAirTickets,
        UsersDataAccomodation = usersDataAccomodation,
        UsersDataCars = usersDataCars
    };

    return View("VerifiedBookingsOwner", model);
}

[HttpPost]
public async Task<IActionResult> UpdateEntertainmentForm(int editAttractionId,
Entertainment model, string photoData)
{
    var currentUser = await _context.Users.FirstOrDefaultAsync(u => u.UserName ==
User.Identity.Name);

    var attraction = _context.Entertainment.Find(editAttractionId);

    string filePath = "C:\\Users\\Angelina\\Pictures\\Camera
Roll\\bbb846030d108b92a3fefbfca1f7bbe6.jpg";
    byte[] fileBytes = System.IO.File.ReadAllBytes(filePath);
    ViewBag.CarPhoto = fileBytes;
}

```

```

var photos = JsonConvert.DeserializeObject<List<string>>(photoData);
var photoBytesList = photos.Select(photoBase64 =>
Convert.FromBase64String(photoBase64.Split(',')[1])).ToList();

attraction.Title = model.Title;
attraction.City = model.City;
attraction.Country = model.Country;
attraction.Address = model.Address;
attraction.Languages = model.Languages;
attraction.TimeOfDay = model.TimeOfDay;
attraction.Cost = model.Cost;
attraction.FreeCancellation = model.FreeCancellation;
attraction.Category = model.Category;
attraction.MainPhoto = photoBytesList[0];
attraction.Description = model.Description;
attraction.AvailableDates = model.AvailableDates;
attraction.AmountOfTickets = model.AmountOfTickets;
attraction.VerifiedByAdmin = false;
attraction.PublisherId = currentUser.Id;
attraction.RejectedMessage = string.Empty;
attraction.RejectedByAdmin = false;

var photosToDelete = _context.ObjectPhotos.Where(photo => photo.ObjectId == editAttractionId).ToList();

if (photosToDelete.Any())
{
    // Удаляем найденные фотографии
    _context.ObjectPhotos.RemoveRange(photosToDelete);

    // Сохраняем изменения в базе данных
    _context.SaveChanges();
}

foreach (var photo in photoBytesList)
{
    var photoss = new Photos
    {
        ObjectId = editAttractionId,
        PhotoArray = photo,
    };
    _context.ObjectPhotos.Add(photoss);
    await _context.SaveChangesAsync();
}

await _context.SaveChangesAsync(); // Сохраняем изменения в базе данных
return await UnverifiedObjects();
}

public async Task<IActionResult> ShowAccomodationInformation(int accomodationId)

```

```

{
    var accomodation = _context.Accomodation.Find(accomodationId);

    string filePath = "C:\\Users\\Angelina\\Pictures\\Camera
Roll\\bbb846030d108b92a3fefbfca1f7bbe6.jpg";
    byte[] fileBytes = System.IO.File.ReadAllBytes(filePath);
    ViewBag.CarPhoto = fileBytes;

    var photos = _context.AccomodationPhotos
        .Where(photo => photo.ObjectId == accomodationId)
        .Select(photo => photo.PhotoArray)
        .ToList();

    var roomphotos = _context.RoomPhotos
        .Where(photo => photo.ObjectId == accomodationId)
        .Select(photo => photo.PhotoArray)
        .ToList();

    var rooms = _context.Rooms
        .Where(room => room.AccomodationId == accomodationId)
        .ToList();

    foreach(var room in rooms)
    {
        room.Photos = _context.RoomPhotos
            .Where(photo => photo.ObjectId == accomodationId && photo.RoomId == room.ID)
            .ToList();
    }

    var model = new ShowAccomodationInformationViewModel
    {
        Name = accomodation.Name,
        City = accomodation.City,
        Country = accomodation.Country,
        Address = accomodation.Address,
        TypeOfAccomodation = accomodation.TypeOfAccomodation,
        TypesOfNutrition = accomodation.TypesOfNutrition,
        Parking = accomodation.Parking,
        SwimmingPool = accomodation.SwimmingPool,
        FreeWIFI = accomodation.FreeWIFI,
        Photos = photos,
        PrivateBeach = accomodation.PrivateBeach,
        LineOfBeach = accomodation.LineOfBeach,
        Restaurants = accomodation.Restaurants,
        SPA = accomodation.SPA,
        Bar = accomodation.Bar,
        Garden = accomodation.Garden,
        AllRooms = rooms,
        TransferToAirport = accomodation.TransferToAirport,
        SmookingRooms = accomodation.SmookingRooms,
        FamilyRooms = accomodation.FamilyRooms,
        CarChargingStation = accomodation.CarChargingStation,
        WheelchairAccessible = accomodation.WheelchairAccessible,
    }
}

```

```

FitnessCentre = accomodation.FitnessCentre,
PetsAllowed = accomodation.PetsAllowed,
DeliveryFoodToTheRoom = accomodation.DeliveryFoodToTheRoom,
EveryHourFrontDesk = accomodation.EveryHourFrontDesk,
Description = accomodation.Description
};

return View("AccomodationInformation", model);
}

[HttpPost]
public async Task<IActionResult> CancelCarBooking(int cancelCarId)
{
    var car = _context.BookingCars.Find(cancelCarId);
    DateTime startDate = car.DateOfDeparture;
    DateTime endDate = car.ReturnDate;

    int carId = car.CarId;
    var originalCar = _context.Car.Find(carId);
    originalCar.StartDates.Add(startDate);
    originalCar.EndDates.Add(endDate);
    _context.Car.Update(originalCar);

    car.CanceledBooking = true;
    await _context.SaveChangesAsync();

    return await ShowUnverifiedBookingItems();
}

```

ПРИЛОЖЕНИЕ В

Итоговая диаграмма классов

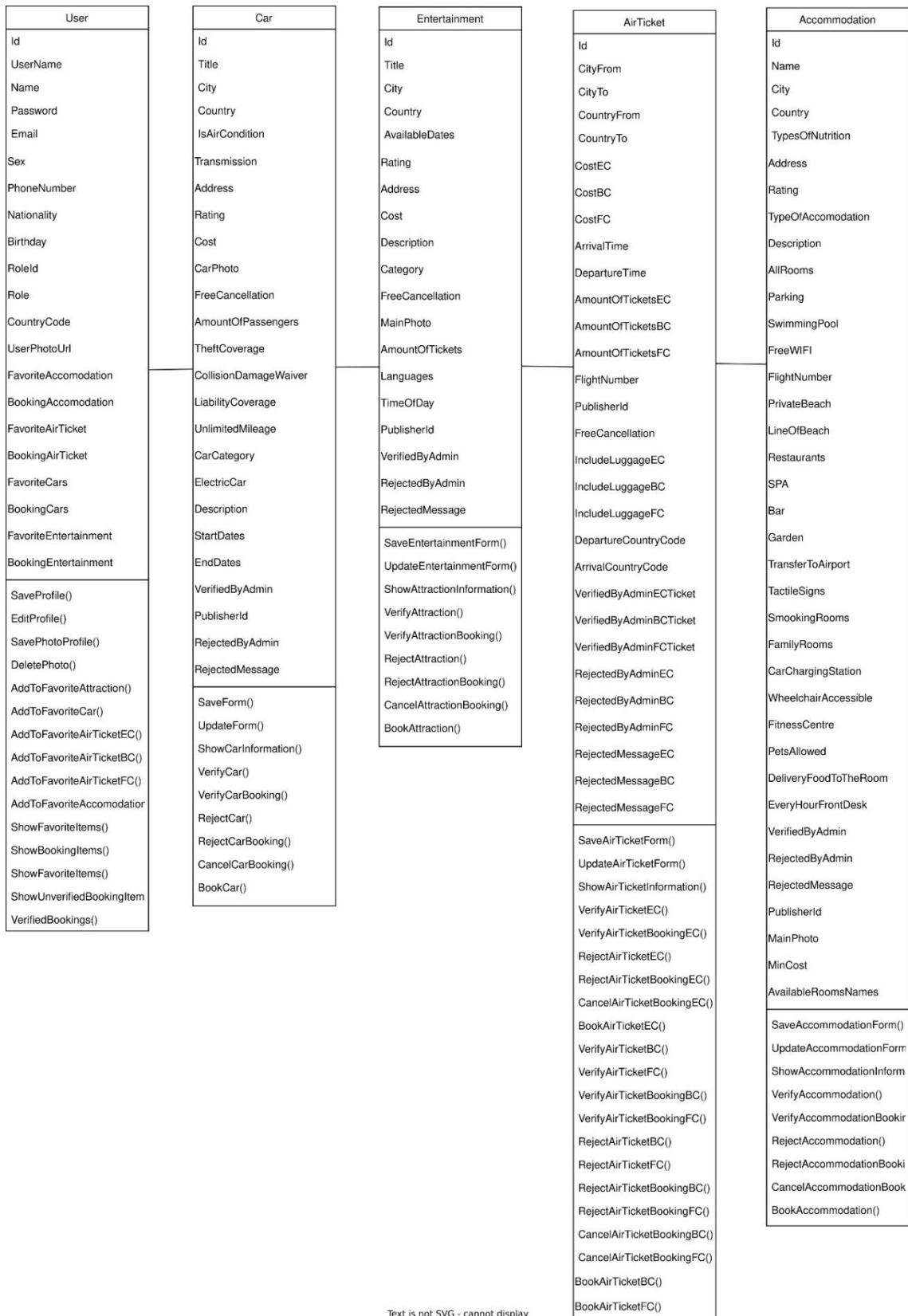


Рисунок В.1 – Итоговая диаграмма классов