МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждения образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ

ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет информационных технологий

Кафедра программной инженерии

Специальность 1-40 01 01 Программное обеспечение информационных технологий

Направление специальности 1-40 01 01 10 Программное обеспечение информационных технологий (программирование интернет приложений)

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**КУРСОВОГО ПРОЕКТ:**

по дисциплине «Объектно-ориентированные технологии программирования и стандарты проектирования»

Тема Программное средство для поиска и бронирования авиабилетов

Исполнитель

студент (ка) 2 курса группы 5 Коршун Никита Игоревич

(Ф.И.О.)

Руководитель работы преп.-стажер Север А.С.

(учен. степень, звание, должность, подпись, Ф.И.О.)

Курсовой проект защищен с оценкой

Председатель Пацей Н.В

(подпись)

Минск 2023

МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования   
«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет информационных технологий   
Кафедра программной инженерии

Утверждаю

Заведующий кафедрой ПИ

\_\_\_\_\_\_\_\_\_\_\_\_ \_\_ Н.В. Пацей

подпись инициалы и фамилия

“\_\_\_”\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_2023г.

**ЗАДАНИЕ**

**к курсовому проектированию**

**по дисциплине** "Объектно-ориентированные технологии программирования и стандарты проектирования"

|  |  |
| --- | --- |
| Специальность: 1-40 01 01 Программное обеспечение информационных технологий  Студент: Коршун Никита Игоревич | Группа:\_\_5\_\_ |
| **Тема: Программное средство для поиска и бронирования авиабилетов** | |

**1. Срок сдачи студентом законченной работы**: "20 мая 2023 г."

**2. Исходные данные к проекту:**

**2.1**. Функционально ПС поддерживает:

* Функции администратора:
  + Поддерживать работу c базой данных;
  + Просмотр данных всех пользователей;
  + Изменение привилегий пользователей;
* Функции клиента:
  + Выполнять регистрацию и авторизацию;
  + Редактирование профиля;
  + Просмотр информации о рейсах;
  + Бронирование билетов на рейсы;
  + Выполнять поисковые запросы.

**2.2.** При выполнении курсового проекта необходимо использовать принципы проектирования ООП. Приложение разрабатывается под ОС Windows и представляет собой настольное приложение (desktop). Отображение, бизнес логика должны быть максимально независимы друг от друга для возможности расширения. Диаграммы вариантов использования, классов реализации задачи, взаимодействия разработать на основе UML. Язык разработки проекта – C#. Управление программой должно быть интуитивно понятным и удобным. Использовать архитектурные и шаблоны проектирования.

**3. Содержание расчетно-пояснительной записки**

(перечень вопросов подлежащих разработке)

* Введение
* Постановка задачи и обзор литературы (алгоритмы решения, обзор прототипов, актуальность задачи)
* Проектирование архитектуры проекта (структура модулей, классов).
* Разработка функциональной модели и модели данных ПС (выполняемые функции)
* Руководство пользователя
* Тестирование
* Заключение
* Список используемых источников
* Приложения

**4. Форма представления выполненной курсовой работы:**

* + Теоретическая часть курсового проекта должны быть представлены в формате docx. Оформление записки должно быть согласно выданным правилам.
  + Листинги программы представляются в приложении.
  + Пояснительную записку, листинги, проект (инсталляцию проекта) необходимо загрузить диск, указанный преподавателем.

***Календарный план***

|  |  |  |  |
| --- | --- | --- | --- |
| №  п/п | Наименование этапов курсового проекта | Срок выполнения этапов проекта | Примечание |
| 1 | Введение | 19.02.2023 |  |
| 2 | Аналитический обзор литературы по теме проекта. Изучение требований, определение вариантов использования | 12.03.2023 |  |
| 3 | Анализ и проектирование архитектуры приложения (построение диаграмм, проектирование бизнес-слоя, представления и данных) | 26.03.2023 |  |
| 4 | Проектирование структуры базы данных. Разработка дизайна пользовательского интерфейса | 02.04.2023 |  |
| 5 | Кодирование программного средства | 23.04.2023 |  |
| 6 | Тестирования и отладка программного средства | 30.04.2023 |  |
| 7 | Оформление пояснительной записки | 07.05.2023 |  |
| 8 | Защита проекта | 20.05.2023 |  |

**5. Дата выдачи задания** 12.02.2023

Руководитель\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ *Север А.С.*

(подпись)

Задание принял к исполнению 12.02.2023\_\_\_\_\_\_\_\_\_\_\_\_\_\_ *Коршун Н. И.*

(дата и подпись студента)

Содержание

[ВВЕДЕНИЕ 6](#_Toc135335583)

[1 Аналитический обзор литературы и формирование требований 7](#_Toc135335584)

[1.1 Анализ прототипов 7](#_Toc135335585)

[1.1.1 Aviasales 7](#_Toc135335586)

[1.1.2 Kayak 8](#_Toc135335587)

[1.1.3 Skyscanner 8](#_Toc135335588)

[1.2 Требования к проекту 9](#_Toc135335589)

[2 Анализ требований к программному средству и разработка функциональных требований 11](#_Toc135335590)

[2.1 Описание средств разработки 11](#_Toc135335591)

[2.2 Технологии и инструменты разработки 11](#_Toc135335592)

[2.3 Спецификация функциональных требований к программному средству 12](#_Toc135335593)

[2.4 Спецификация функциональных требований 13](#_Toc135335594)

[3 Проектирование программного средства 15](#_Toc135335595)

[3.1 Общая структура 15](#_Toc135335596)

[3.2 Разработка общей схемы приложения 19](#_Toc135335597)

[3.3 Взаимоотношение между классами 22](#_Toc135335598)

[3.4 Модель базы данных 22](#_Toc135335599)

[3.5 Проектирование архитектуры приложения 24](#_Toc135335600)

[3.6 Проектирование последовательностей проекта 24](#_Toc135335601)

[4 Реализация программного средства 26](#_Toc135335602)

[4.1 Основные классы программного средства 26](#_Toc135335603)

[4.2 Выполнение входа 26](#_Toc135335604)

[4.3 Регистрация 26](#_Toc135335605)

[4.4 Просмотр рейсов 27](#_Toc135335606)

[4.5 Бронирование билетов на рейсы 27](#_Toc135335607)

[4.6 Добавление отзывов 28](#_Toc135335608)

[4.7 Функционирование панели администратора 28](#_Toc135335609)

[5 Тестирование, проверка работоспособности и анализ полученных результатов 29](#_Toc135335610)

[5.1 Тестирование авторизации и регистрации 29](#_Toc135335611)

[5.2 Тестирование поиска билетов на рейсы 30](#_Toc135335612)

[5.3 Тестирование добавления и просмотра отзывов 31](#_Toc135335613)

[6 Руководство по использованию 33](#_Toc135335614)

[ЗАКЛЮЧЕНИЕ 37](#_Toc135335615)

[Список использованных источников 38](#_Toc135335616)

[ПРИЛОЖЕНИЕ А 39](#_Toc135335617)

[ПРИЛОЖЕНИЕ Б 39](#_Toc135335618)

[ПРИЛОЖЕНИЕ В 40](#_Toc135335619)

[ПРИЛОЖЕНИЕ Г 41](#_Toc135335620)

[ПРИЛОЖЕНИЕ Д 43](#_Toc135335621)

# ВВЕДЕНИЕ

В современном мире все больше людей стремятся расширить свои горизонты и открыть для себя новые места. Путешествия становятся неотъемлемой частью жизни многих людей, и при этом они ищут удобные и надежные способы планирования и бронирования своих поездок.

В этой ситуации разработка программного средства, которое предоставляет возможность легко и удобно искать и бронировать авиабилеты, становится актуальной и востребованной.

Целью данного курсового проекта является разработка программного средства, которое облегчит процесс поиска, сравнения и бронирования авиабилетов для путешественников. Мы стремимся создать интуитивно понятный и функциональный интерфейс, который позволит пользователям быстро и эффективно находить подходящие рейсы, сравнивать цены и осуществлять бронирование с минимальными усилиями.

Приложение имеет 2 типа пользователей: администратор и пользователь. Они имеют разные полномочия. Пользователи могут выполнять сортировку для поиска подходящих для них билетов на рейс и бронировать их, просматривать отзывы о компаниях и оставлять свои собственные. Администраторы могут изменять роль пользователей, могут добавлять и удалять рейсы, просматривать список пассажиров на рейсы.

Для реализации данного проекта будет использован язык программирования WPF (Windows Presentation Foundation), который обеспечивает гибкость и мощные возможности в создании пользовательских интерфейсов. В качестве базы данных будет применена MS SQL, обеспечивающая надежное хранение и доступ к информации о рейсах, бронированиях и пользователях.

Таким образом, разрабатываемое программное средство представляет собой инновационное решение, которое совмещает простоту использования, надежность и широкий функционал для удовлетворения потребностей путешественников в поиске и бронировании авиабилетов.

# Аналитический обзор литературы и формирование требований

## Анализ прототипов

Были проанализированы цели и задачи, поставленные в данном курсовом проекте, а также рассмотрены аналогичные примеры их решений. На основании анализа всех достоинств и недостатков данных альтернативных решений были сформулированы требования к данному программному средству.

### Aviasales

Самое популярное на территории СНГ аналогичное решение — интернет-ресурс «Aviasales» [1], крупнейшая в площадка для поиска билетов на самолет, на которой пользователи могут найти подходящий для них рейс и забронировать его. Основными преимуществами данного ресурса является широкий выбор авиакомпаний, простой и интуитивно понятный интерфейс, а также функции поиска, позволяющие находить наилучшие предложения. Пользователи могут выбрать нужные параметры поиска, такие как даты вылета и прилета, количество пассажиров и класс обслуживания, чтобы получить наиболее подходящие варианты. Кроме того, сервис предоставляет информацию о дополнительных услугах, таких как багаж и страхование, а также может предлагать специальные предложения и акции.

Однако, некоторые недостатки Aviasales включают ограниченные возможности фильтрации и сортировки результатов поиска. Иногда пользователи могут столкнуться с несоответствиями цен и наличия билетов, особенно в случае позднего поиска. Кроме того, в редких случаях могут возникать технические проблемы, связанные с обновлением данных и актуальностью информации о билетах.

Интерфейс интернет-ресурса представлен на рисунке 1.1.

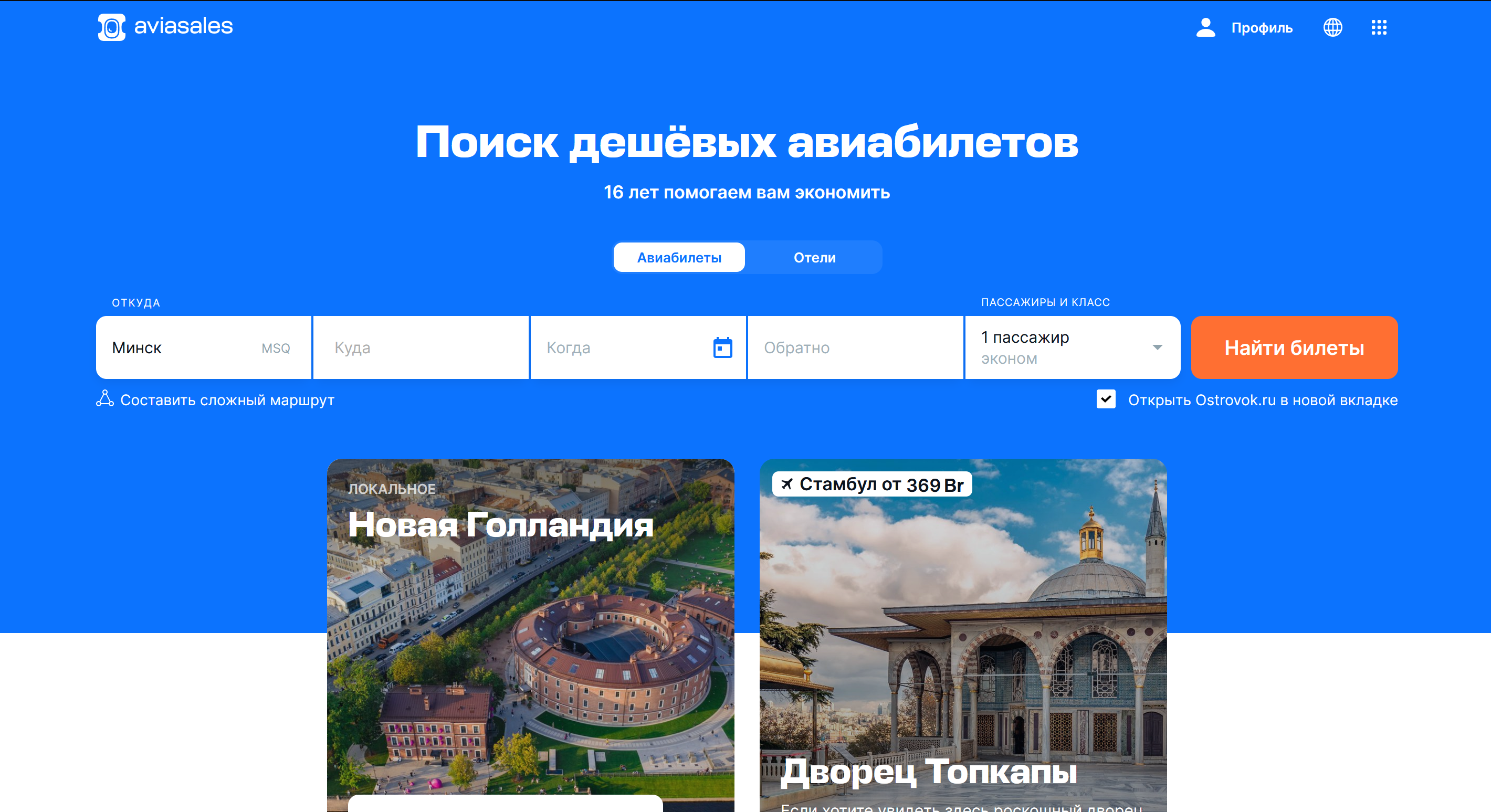


Рисунок 1.1 – Интернет-ресурс «Aviasales»

В целом, интернет-ресурс «Aviasales» представляет собой удобную и популярную площадку для поиска и бронирования авиабилетов на территории СНГ и в других странах.

### Kayak

Еще одним альтернативным решением, которое может использоваться для поиска и бронирования авиабилетов, является интернет-ресурс «Kayak» [2]. Он позволяет пользователям найти подходящий билет на рейс, также он предоставляет пользователям возможность аренды гостиниц, машин.

Одним из главных преимуществ Kayak является его простой и интуитивно понятный интерфейс, а также возможность аренды гостиницы и транспорта. Также на сайте представлены различные категории товаров, что упрощает навигацию и поиск. Кроме того, пользователи могут получить информацию о авиакомпании и посмотреть её отзывы.

Однако, как и у других площадок для поиска и бронирования авиабилетов, есть некоторые недостатки. Например, ограниченная доступность для некоторых регионов и маршрутов. Также иногда могут возникать задержки в обновлении цен и наличии билетов, что может привести к несоответствиям в результате поиска.

Интерфейс интернет-ресурса представлен на рисунке 1.2.

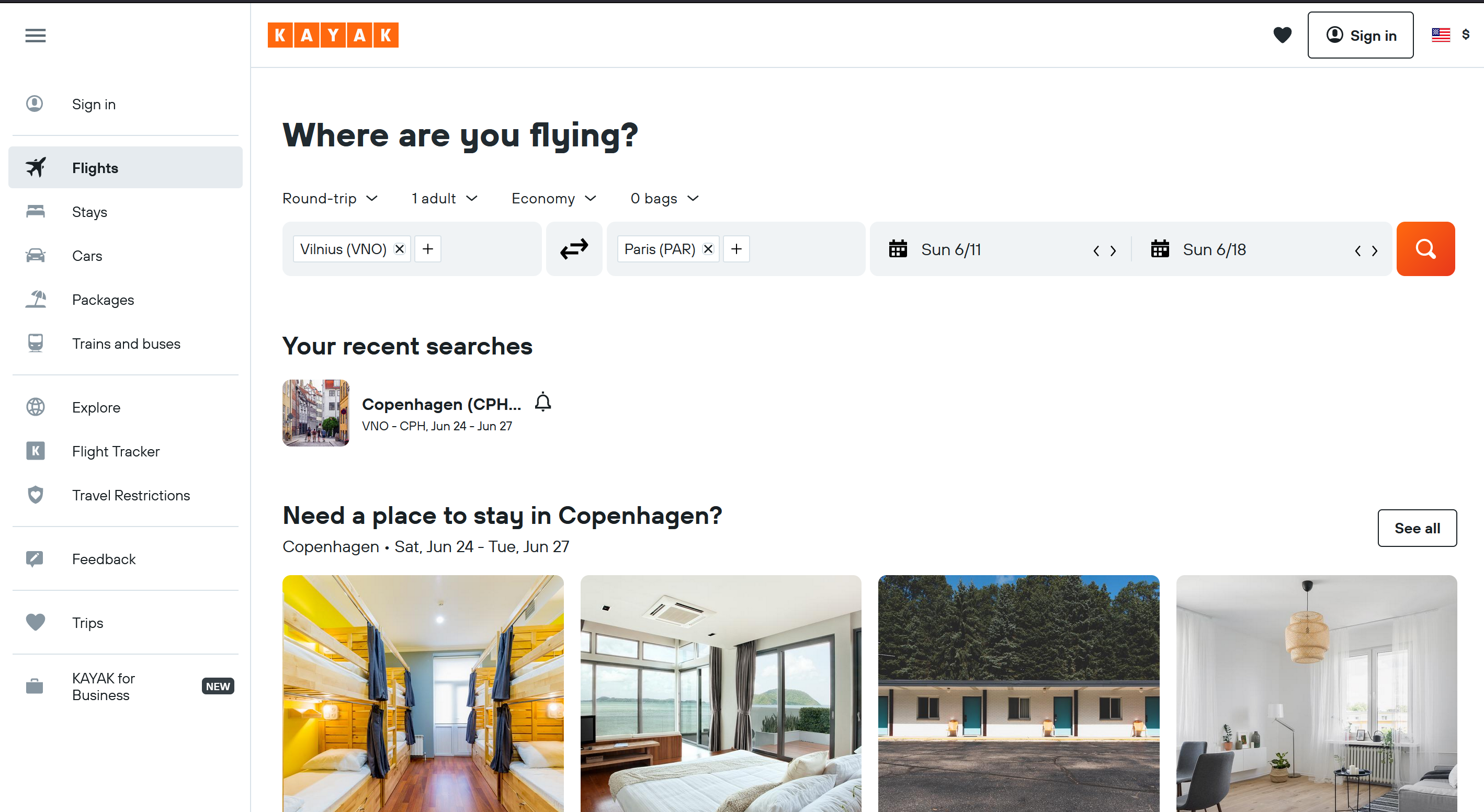


Рисунок 1.2 – Интернет-ресурс «Kufar»

Kayak является удобным и популярным ресурсом для поиска и бронирования авиабилетов и услуг на ряду с Aviasales, но наиболее популярен в странах ЕС.

### Skyscanner

Другой интернет-ресурс, который также предлагает услуги по поиску и бронированию авиабилетов, - это «Skyscanner» [3].

Skyscanner - это популярный онлайн-сервис, предоставляющий возможность пользователям искать и сравнивать цены на авиабилеты от различных авиакомпаний и агентств.

Одним из преимуществ Skyscanner является его широкий охват глобальных маршрутов и множество доступных предложений. Пользователи могут легко находить билеты на международные рейсы и сравнивать цены на разных платформах. Skyscanner также отличается своей удобной навигацией и простым процессом бронирования. Он предоставляет пользователю информацию о доступных вариантах перелетов, включая время вылета и прилета, длительность перелета и количество остановок. Также ресурс предоставляет возможность проката автомобилей и поиска отелей.

Однако, некоторые недостатки Skyscanner включают возможность несоответствия цен и наличия билетов, особенно в случае быстро изменяющихся тарифов и акций. Иногда пользователи могут столкнуться с ограничениями фильтрации результатов поиска, что может усложнить процесс выбора оптимального варианта. Кроме того, в редких случаях могут возникать технические проблемы или задержки в обновлении информации о билетах, что может потребовать дополнительного времени для поиска и бронирования.

Интерфейс интернет-ресурса представлен на рисунке 1.3.

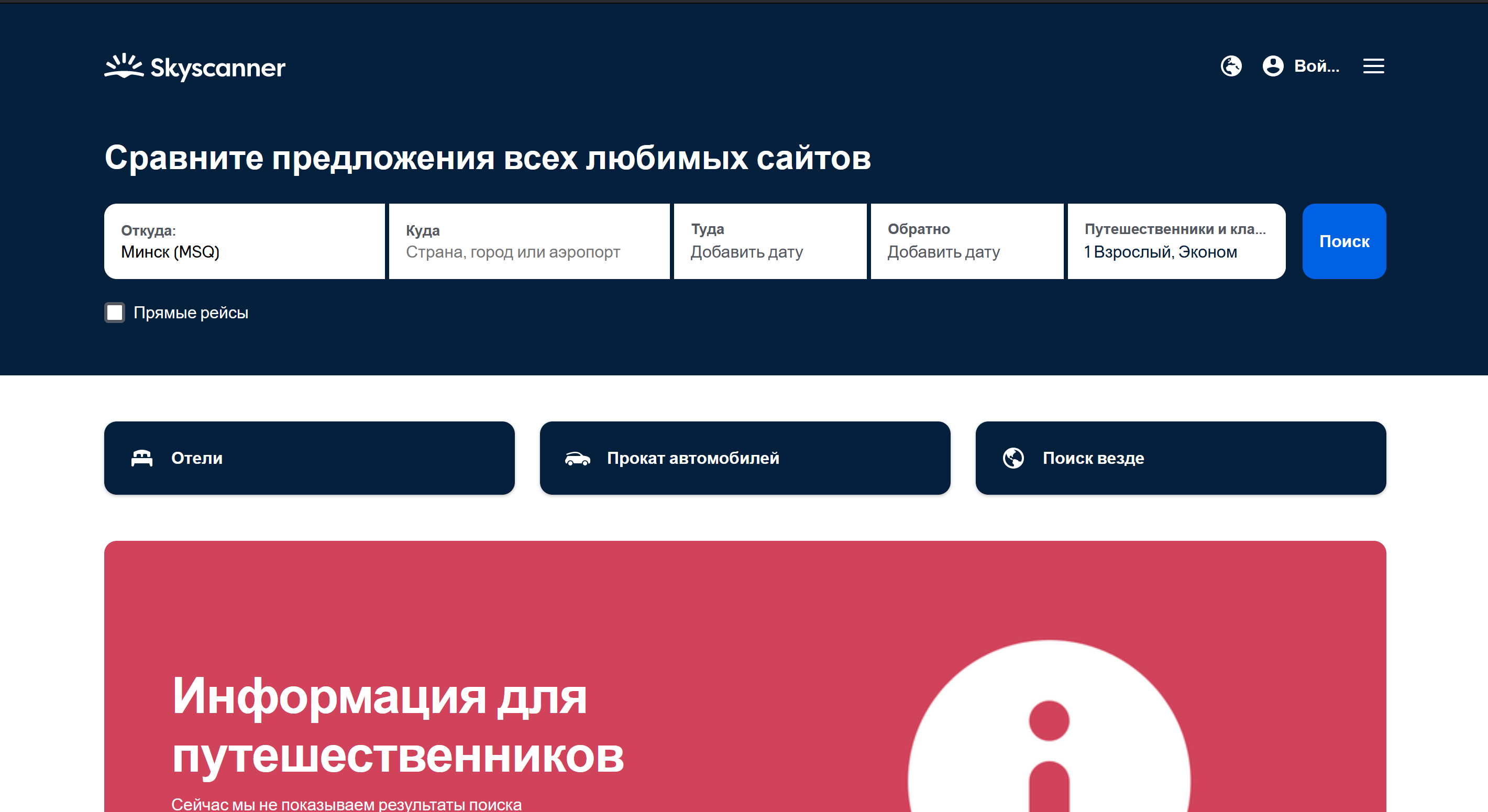


Рисунок 1.3 – Интернет-ресурс «Skyscanner»

По итогу, «Skyscanner» является полезным и удобным ресурсом для поиска и бронирования авиабилетов, который может быть полезен при бронировании путешествия.

## Требования к проекту

Обзор вышеперечисленных известных аналогов позволяет проанализировать все преимущества и недостатки альтернативных возможностей и позволяет сформулировать список требований, предъявляемых к программному средству, разрабатываемому в данном курсовом проекте. Программное средство должно обеспечивать возможность выполнения перечисленных ниже функций:

* управление администратором базой данных;
* возможность пользователю зарегистрироваться или войти в существующую учетную запись;
* возможность авторизированным пользователям бронирования билетов на рейсы;
* возможность просмотра личной страницы пользователя с отображением всех рейсов пользователя;
* просмотр и добавление отзывов на авиакомпании.

Таким образом, можно сказать, что требования к разрабатываемому программному средству основаны на анализе прототипов и аналогов, и представляют собой список функций, необходимых для управления базой данных и обеспечения комфортного пользовательского опыта.

Кроме того, важным критерием является безопасность пользователей при совершении бронировании билетов, поэтому необходимо уделять внимание проверке введенной информации. Разрабатываемое программное средство должно быть удобным и простым в использовании.

# Анализ требований к программному средству и разработка функциональных требований

Для начала этапа разработки необходимо сформулировать функциональные требования к программному средству.

## Описание средств разработки

При разработке приложения были использованы:

* интегрированная среда разработки Microsoft Visual Studio 2022;
* программная платформа .NET 6.0;
* язык программирования C#;
* технология WPF;
* расширяемый язык разметки XAML;
* технология Entity Framework Core;
* MS SQL Server.

Данный набор средств разработки и технологий был выбран с учетом их широкой функциональности, эффективности и поддержки, чтобы обеспечить разработку качественного и масштабируемого программного средства. Использование интегрированной среды разработки Microsoft Visual Studio 2022, программной платформы .NET 6.0, языка программирования C# и других технологий позволило нам эффективно реализовать требования и достичь желаемого функционального результата.

## Технологии и инструменты разработки

Microsoft Visual Studio 2022 - это программный продукт, который объединяет инструменты для написания, тестирования и сборки кода, а также для публикации разработанных приложений. Он позволяет создавать различные типы приложений, включая консольные и десктопные, с использованием технологий, таких как WinForms или WPF [4].

Платформа .NET Framework - это технология, разработанная компанией Microsoft в 2002 году, которая обеспечивает возможность создания и выполнения веб-служб и приложений Windows. Она предоставляет среду программирования для локального сохранения и выполнения объектного кода, а также его выполнения в Интернете или удаленно. Основой платформы является общеязыковая среда Common Language Runtime, которая может выполнять код на разных языках программирования [5].

Для разработки в платформе .NET применяется язык программирования C#, который является основным в данной технологии. Это объектно-ориентированный язык, который характеризуется строгой статической типизацией, поддержкой перегрузки операторов, указателей на функции-члены классов, атрибутов, событий и свойств, а также исключений. Кроме того, C# используется в качестве основного языка разработки приложений на технологии WPF [6].

Microsoft WPF - это технология, используемая для создания клиентских приложений Windows с интерактивным пользовательским интерфейсом и разделением дизайна и бизнес-логики. Она является альтернативой WinForms и включена в состав .NET Framework. WPF использует язык разметки XAML для описания элементов пользовательского интерфейса и обеспечивает более высокую гибкость и масштабируемость в сравнении с WinForms [7].

WPF позволяет разрабатывать визуальный интерфейс, используя язык разметки XAML (eXtensible Application Markup Language). Он предоставляет различные элементы управления, возможности привязки данных, макеты, 2D и 3D графику, анимацию, стили, шаблоны, документы, текст, мультимедиа и оформление. XAML - это декларативный язык описания интерфейса на основе XML, который позволяет отделить дизайн интерфейса от его реализации и бизнес-логики [8].

Entity Framework Core является расширяемой технологией, предоставляющей высокий уровень абстракции для взаимодействия с базами данных в приложении WPF. В отличие от традиционных средств ADO.NET, которые требуют создания подключений и объектов для взаимодействия с базами данных, Entity Framework Core позволяет абстрагироваться от деталей реализации базы данных и работать с данными на уровне объектов. Entity Framework Core также является кроссплатформенной и поддерживает .NET Core.

Существуют три возможных способа использования Entity Framework Core в приложении WPF: Database First, Model First и Code First.

При использовании Database First, Entity Framework Core создает классы, отражающие модель базы данных.

При использовании Model First, сначала создается модель базы данных, а затем на ее основе создается реальная база данных на сервере.

При использовании Code First, разработчик создает классы модели данных, которые будут храниться в базе данных, а затем Entity Framework Core генерирует базу данных и ее таблицы на основе этих классов [9].

MS SQL Server использует реляционную модель баз данных, где данные хранятся в таблицах, состоящих из строк и столбцов. Каждая строка содержит отдельный объект, а атрибуты объектов хранятся в столбцах. Для взаимодействия с базой данных используется язык SQL (Structured Query Language). Клиент отправляет запрос на языке SQL, который база данных интерпретирует и выполняет, а затем возвращает результат выполнения запроса клиенту. Основной используемый язык запросов в MS SQL Server - Transact-SQL, который основан на SQL с дополнительными расширениями [11].

## Спецификация функциональных требований к программному средству

Программное средство должно предоставлять следующие функциональные возможности:

Для пользователя:

* регистрация;
* авторизация;
* изменение данных своего профиля;
* бронирование билетов на рейсы;
* просмотр доступных рейсов по разным критериям;
* возврат забронированных билетов;
* добавление отзывов авиакомпаниям;
* просмотр отзывов авиакомпаний;

Для администратора:

* авторизация;
* изменение данных своего профиля;
* просмотр информации о пользователях;
* изменение привелегий у пользователей;
* добавление новых рейсов;
* просмотр доступных рейсов;
* удаление рейсов;
* просмотр пассажиров на определенный рейс;

Реализация этих требований позволит создать удобное и функциональное программное средство, которое будет полезным для пользователей и обеспечит эффективный поиск и бронирование авиабилетов.

По итогу работы над продуктом, его необходимо протестировать и отладить, проанализировать его возможности и, по необходимости, внести дополнения или улучшения проекта, с возможностью дальнейшего расширения проекта.

## Спецификация функциональных требований

Для функциональности ПС необходимо создание базы данных для хранения информации приложения. Подробно база данных описано в следующем разделе.

В программном средстве необходимо реализовать регистрацию и авторизацию пользователей для доступа ко всем возможностям приложения. Для авторизации входными параметрами являются логин и пароль пользователя, которые содержатся в базе данных. Для регистрации входными данными являются логин, имя, фамилия, e-mail и пароль. Введенные данные, успешно прошедшие валидацию, заносятся в базу данных.

Пользователь может просматривать историю билетов, производит бронирование билетов на рейс и оставлять отзывы авиакомпаниям. Для каждого пользователя создается его профиль, в котором он может просмотреть информацию о своем профиле. Все сведения пользователе должны автоматически обновляться в базе данных.

Администратор обладает теми же возможностями, что и пользователь, а также обладает возможностями изменения привилегий пользователя, добавления и удаления рейсов, а также обладает возможностью просматривать пассажиров на рейсы.

Описание функциональности программного средства представлено на UML-схеме, изображённой на рисунке 2.1.



Рисунок 2.1 – UML-диаграмма возможностей

Таким образом, программа должна обеспечивать удобный интерфейс для пользователя и администратора, а также функционал для работы с базой данных и другими необходимыми технологиями.

# Проектирование программного средства

## Общая структура

Программное средство для поиска и бронирования авиабилетов имеет следующею структуру, представленную на рисунке 3.1.

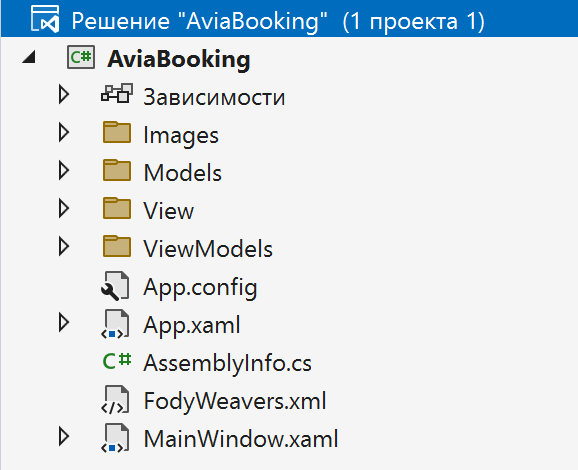


Рисунок 3.1 – Структура проекта

Описание структуры основных папок и файлов проекта представлено в таблице 3.1.

Таблица 3.1 – Описание структуры папок и файлов проекта

|  |  |
| --- | --- |
| Имя пакета | Содержание |
| Папка Images | Здесь содержатся все изображения которые используются в приложении |
| Папка Models | Здесь описаны модели, с которыми происходит вся работа в приложении:  Авиакомпания, Пользователь, Рейc, Платеж, Отзыв, Пассажир |

Продолжение таблицы 3.1

|  |  |
| --- | --- |
| Имя пакета | Содержание |
| Папка View | Содержит все представления, которые позволяют пользователю работать с приложением. Они описывают графическую составляющую приложения. |
| Папка ViewModels | Содержит логику, которая позволяет получить данные при помощи View, обработать их, использую при этом Model, после чего передать в базу данных. |
| App.config | Файл конфигурации приложения |
| App.xaml | Файл, который определяет ресурсы которые являются общими для приложения |
| AssemblyInfo.cs | Содержит метаданные, атрибуты и другую информацию которая связана с сборкой |
| FodyWeavers.xaml | Файл конфигурации для плагина Fody, содержит данные предустановки для корректной работы плагина |
| MainWindow.xaml | Главная страница приложения, содержит разметку всего приложения |

Таким образом, сформированная таблица помогает понять общую структуру проекта проектируемого программного средства.

Более подробная структура содержимого папок «Models» и «ViewModels» программного средства показана на рисунке 3.2.

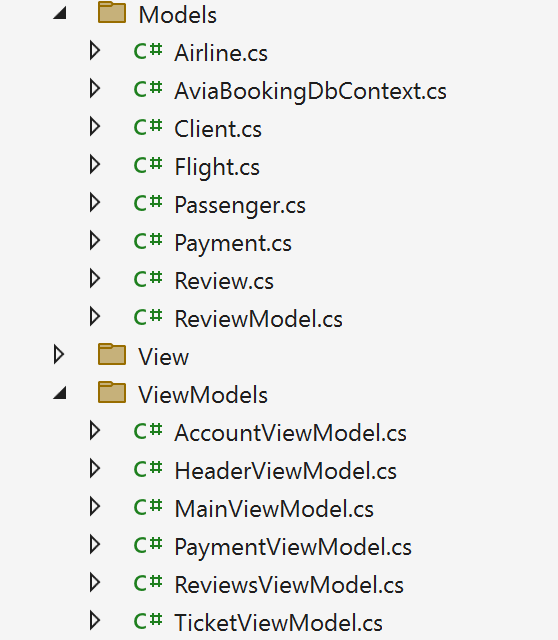


Рисунок 3.2 — Подробная структура папок «Models» и «ViewModels»

Представленное на рисунке 3.2 содержимое папок «Models» и «ViewModel» демонстрирует организацию и структуру данных в проекте.

Описание файлов папок представлено в таблице 3.2.

Таблица 3.2 – Описание файлов папок «Model» и «Repositories»

|  |  |
| --- | --- |
| Имя файла | Содержание |
| Airline.cs | Файл, содержащий описание сущности Airline базы данных, который определяет структуру и свойства сущности которая будет отображена в базе данных. |
| AviaBookingDbContext.cs | Файл контекста базы данных, который представляет основу для взаимодействия с базой данных в приложении. |
| Client.cs | Файл, содержащий описание сущности Client базы данных, который определяет структуру и свойства сущности которая будет отображена в базе данных. |
| Flight.cs | Файл, содержащий описание сущности Flight базы данных, который определяет структуру и свойства сущности которая будет отображена в базе данных. |
| Passenger.cs | Файл, содержащий описание сущности Passenger базы данных, который определяет структуру и свойства сущности которая будет отображена в базе данных. |
| Payment.cs | Файл, содержащий описание сущности Payment базы данных, который определяет структуру и свойства сущности которая будет отображена в базе данных. |
| Review.cs | Файл, содержащий описание сущности Review базы данных, который определяет структуру и свойства сущности которая будет отображена в базе данных. |
| ReviewModel.cs | Файл, содержащий класс, обьединяющий две сущности Review и Client. |
| AccountViewModel.cs | Файл, который является связующим звеном между представлением и сущностью. |
| HeaderViewModel.cs | Файл, который является связующим звеном между представлением и сущностью. |
| MainViewModel.cs | Файл, который является связующим звеном между представлением и сущностью. |

Продолжение таблицы 3.2

|  |  |
| --- | --- |
| PaymentViewModel.cs | Файл, который является связующим звеном между представлением и сущностью. |
| ReviewsViewModel.cs | Файл, который является связующим звеном между представлением и сущностью. |
| TicketViewModel.cs | Файл, который является связующим звеном между представлением и сущностью. |

Рассмотрим папку «View». Подробная структура показана на рисунке 3.3.

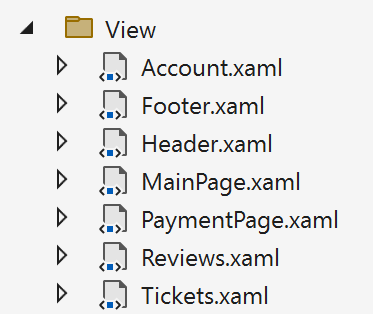


Рисунок 3.3 — Подробная структура папки «View»

Описание файлов папок представлено в таблице 3.3.

Таблица 3.3 – Описание файлов папки «View»

|  |  |
| --- | --- |
| Имя файла | Содержание |
| Account.xaml | Файл содержащий разметку интерфейса личного кабинета. |
| Footer.xaml | Файл содержащий разметку интерфейса подвала приложения. |
| Header.xaml | Файл содержащий разметку интерфейса шапки приложения. |
| MainPage.xaml | Файл содержащий разметку интерфейса главной страницы приложения. |
| PaymentPage.xaml | Файл содержащий разметку интерфейса страницы оплаты. |

Продолжение таблицы 3.3

|  |  |
| --- | --- |
| Reviews.xaml | Файл содержащий разметку интерфейса страницы отзывов. |
| Tickets.xaml | Файл содержащий разметку интерфейса страницы с рейсами. |

В целом, описание структуры проекта позволяет лучше понимать, как устроено программное средство и какие компоненты в нем присутствуют.

## Разработка общей схемы приложения

Рассмотрим общую схему программного средства «Площадка объявлений». На рисунке 3.4 представлена общая схема работы приложения.

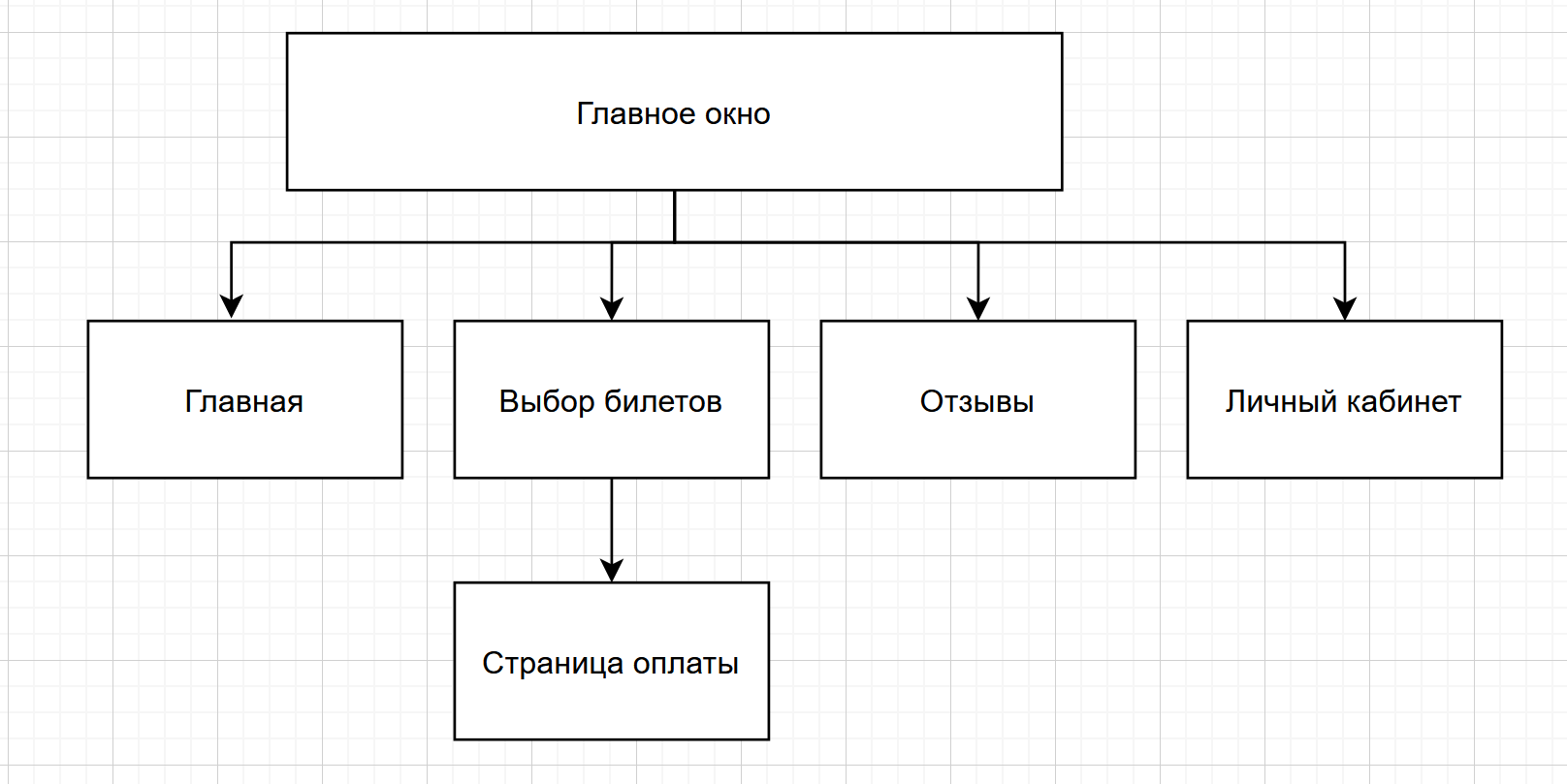


Рисунок 3.4 — Общая схема приложения

Из нее видно, что при запуске приложения у нас появляется главное окно, в котором мы можем выбрать на какую страницу перейти.

В главном окне работа идёт в страничном режиме.

Рассмотрим каждую страницу в отдельности.

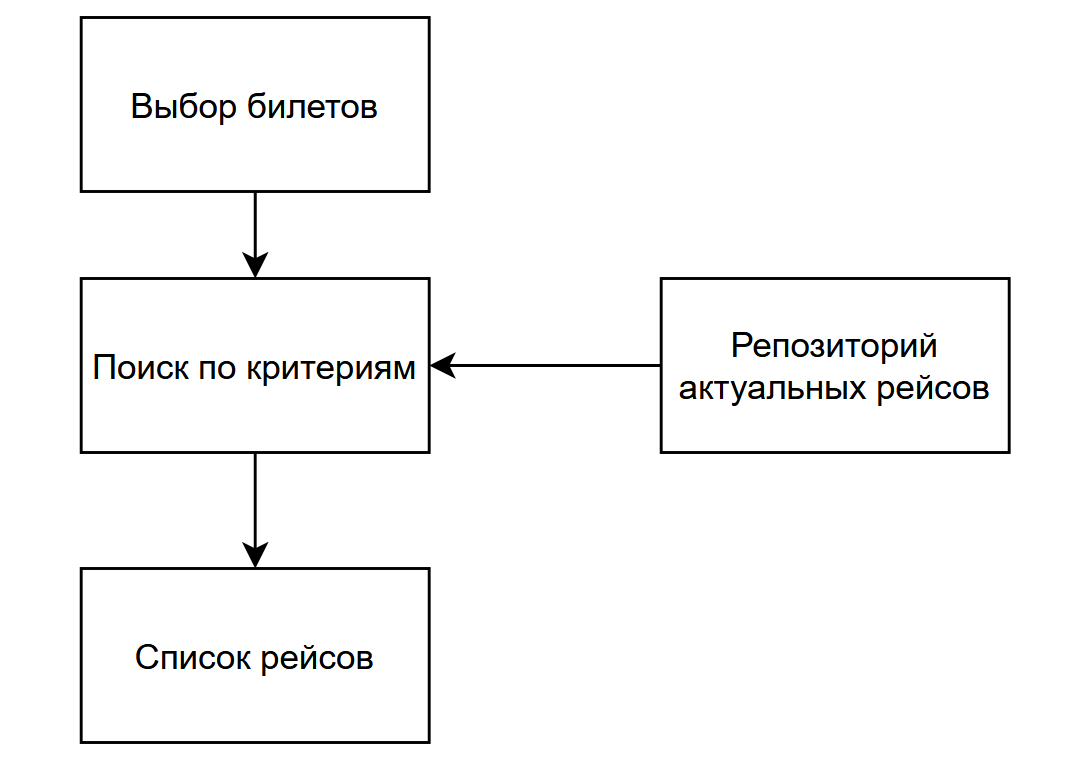


Рисунок 3.5 — Страница «Выбор билетов»

Эта страница содержит список всех актуальных рейсов.

Также она содержит механизм поиска рейса по определённым критериям.

Осуществить поиск можно по таким критериям:

* место отправления;
* место прибытия;
* дата отправления;
* дата прибытия;
* количество пассажиров;
* класс рейса.

На основе критериев поиска, формируется результирующий набор. При нажатии мы видим более подробную информацию о рейсе.

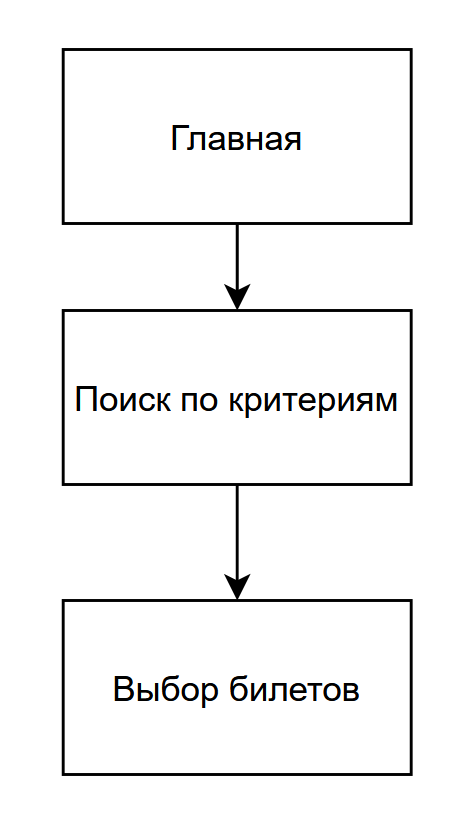


Рисунок 3.6 — Страница «Главная»

На рисунке 3.6 показана схема работы страницы «Мои объявления».

На этой странице ведётся ввод данных в поисковое меню и переход на страницу «Выбор билетов» для вывода результатов поиска.

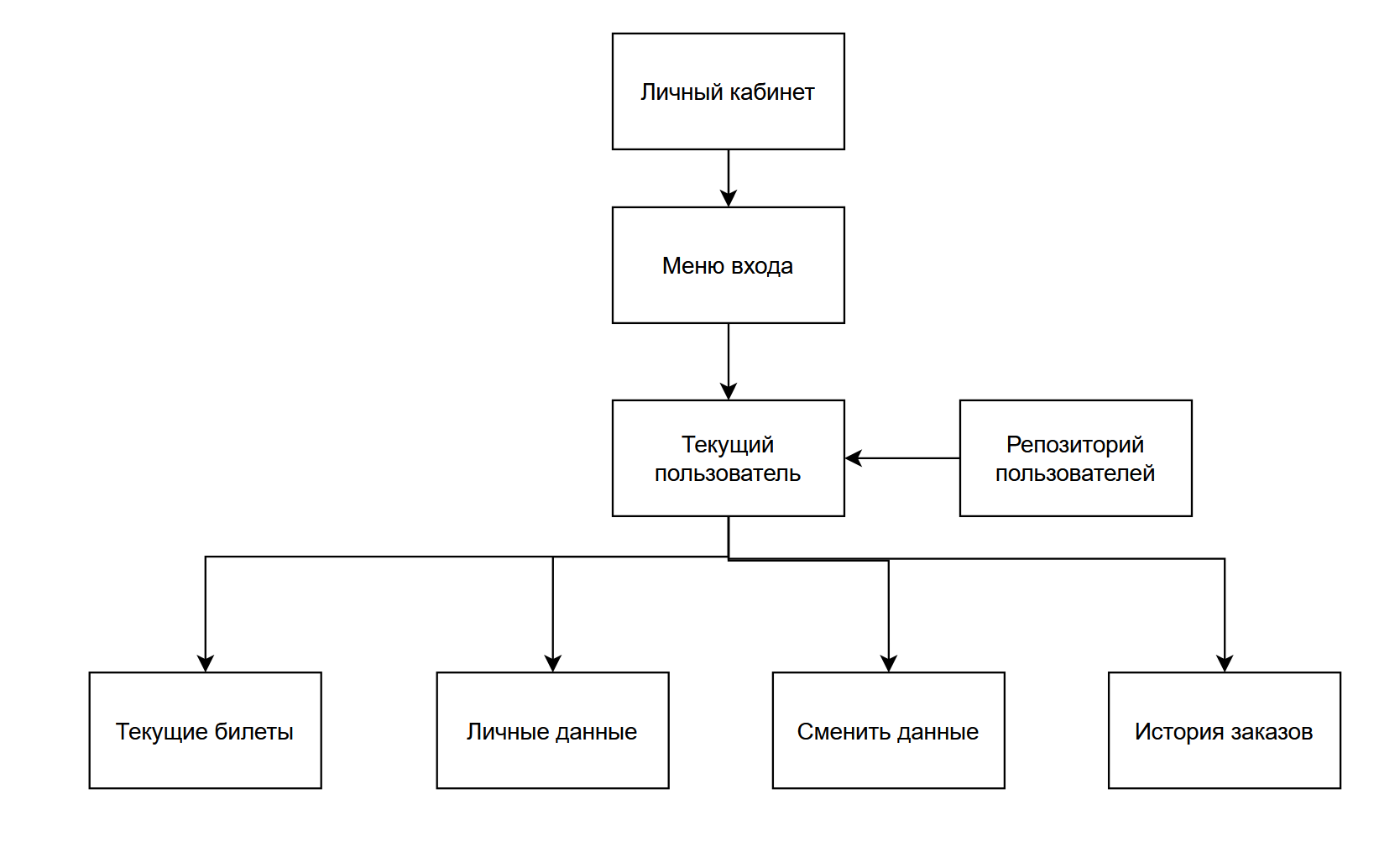


Рисунок 3.7 — Страница «Личный кабинет»

На рисунке 3.7 показана схема страницы «Личный кабинет».

При первом входе на эту страницу, появляется меню входа/регистрации. При вводе корректных данных в окно авторизации, появляется личный кабинет пользователя, в ином случае выдает сообщение «Совпадение не найдено». В окне регистрации имеются поля ввода псевдонима, имени, фамилии, электронной почты и пароля. При корректности данных, нас пересылает на окно авторизации где мы можем авторизоваться в созданный аккаунт.

В личном кабинете можно редактировать свои данные, такие как:

* псевдоним;
* имя;
* фамилия;
* email;
* пароль.

На этой странице есть 4 кнопки: «Текущие билеты», «Личные данные», «Сменить данные», «История заказов».

При нажатии на 1-ю кнопку происходит вывод всех билетов на рейсы, дата отправления которых позже чем текущий день.

При нажатии на 2-ю кнопку – выводит все данные пользователя, такие как псевдоним, имя, фамилия, почта, пароль, имеется кнопка «Показать пароль», которая нужна для отображения пароля.

При нажатии на 3-ю кнопку – выводит все данные пользователя с возможность изменить их, внизу кнопка «Сохранить изменения», по нажатии на которую выводится сообщение об успешном изменении данных.

При нажатии на 4-ю кнопку – выводит все билеты на рейсы дата отправления которых была раньше чем текущий день.

Имеется также 5-я кнопка, при нажатии на которую происходит выход из личного кабинета.

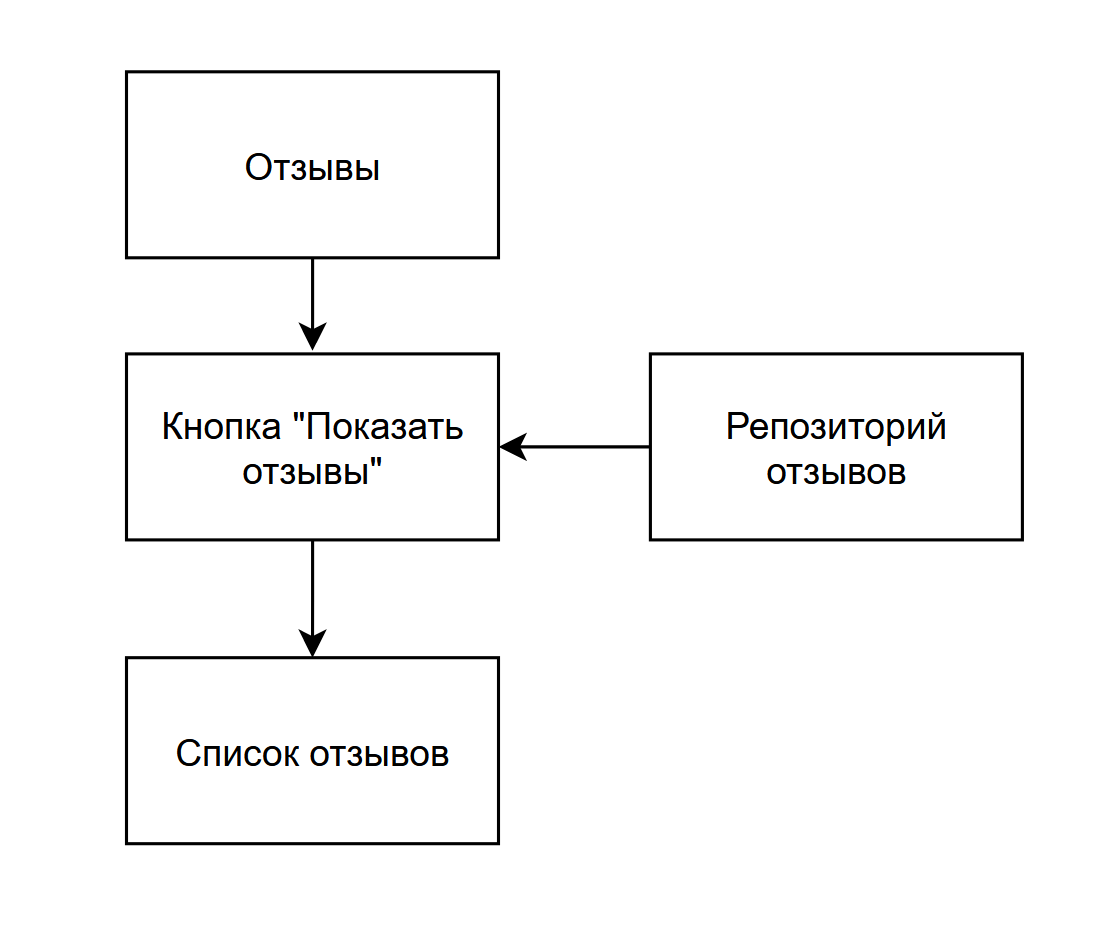


Рисунок 3.8 — Страница «Отзывы»

В этом окне у нас производится вывод отзывов компании.

При нажатии на кнопку «Показать отзывы», если выбрана авиакомпания в выпающем списке, то выведутся отзывы этой компании, иначе выведется сообщение «Выберите авиакомпанию».

## Взаимоотношение между классами

Для визуализации взаимосвязей между классами используется диаграмма UML – графическое представление набора элементов, изображаемое чаще всего в виде связанного графа с вершинами (сущностями) и ребрами (отношениями).

Для представления внутренней структуры программы в виде классов и связей между ними используется диаграмма классов. Приложение спроектировано таким образом, что каждый класс выполняет свои функции и практически не зависит от других. Диаграмма классов представлена в [приложении А](#_ПРИЛОЖЕНИЕ_А).

## Модель базы данных

Для реализации поставленной задачи была создана база данных AviaBooking. Для ее создания использовалась система управления реляционными базами данных MS SQL Server. База данных состоит из таблиц, представленных на рисунке 3.9. Скрипт для создания базы данных представлен в [приложении Г](#_ПРИЛОЖЕНИЕ_Г).

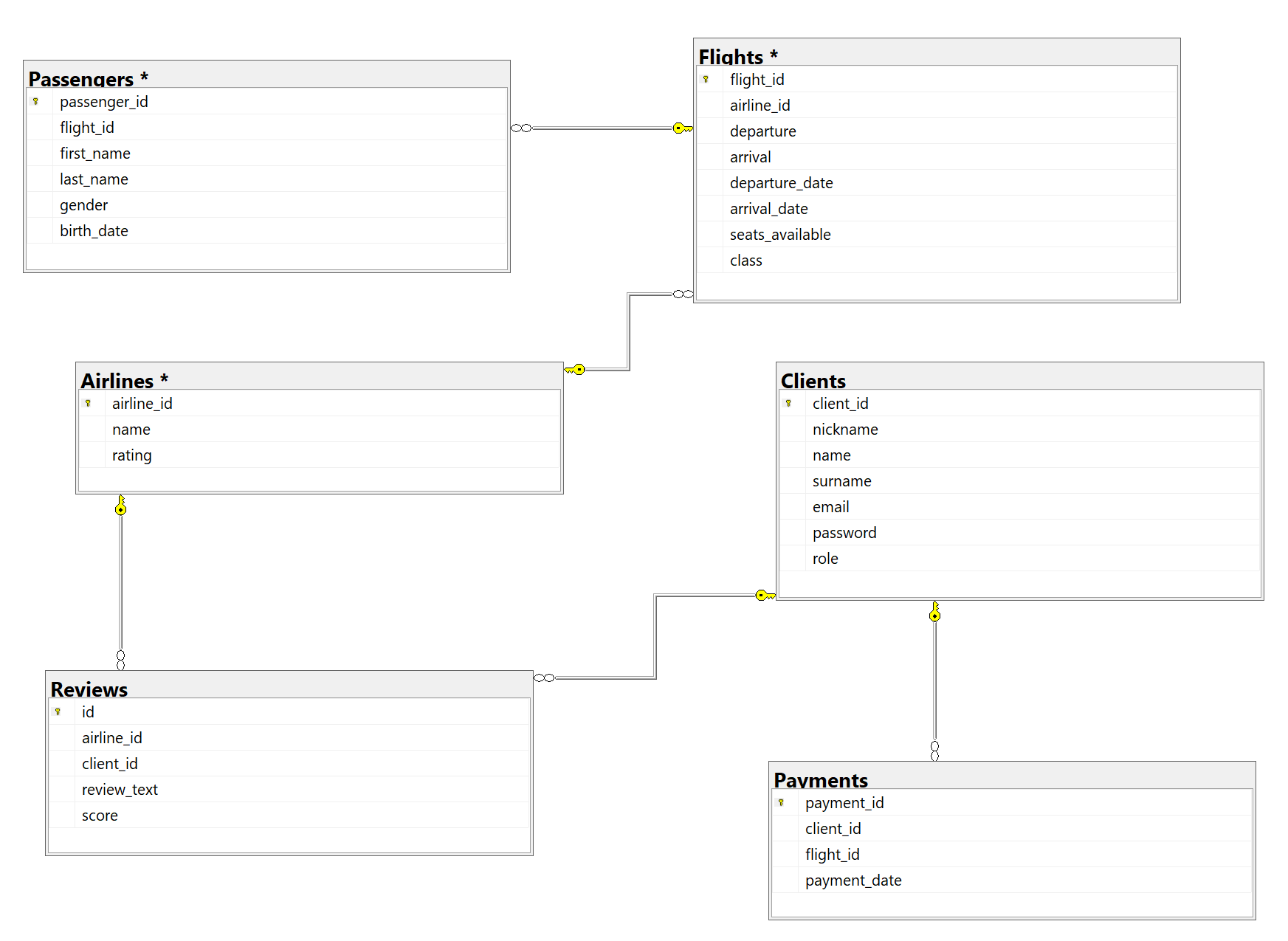


Рисунок 3.9 — База данных AviaBooking

На рисунке 3.10 проиллюстрирована структура таблицы «Clients», которая содержит информацию о пользователях. В данной таблице поле «client\_id», представленное целочисленным значением, является первичным ключом. Поле nickname хранит псевдоним пользователя, name — имя пользователя, surname — фамилия пользователя, email хранит электронную почту пользователя, password – пароль, role — роль пользователя (admin для администратора, user для пользователя).



Рисунок 3.10 — Структура таблицы «Users»

На рисунке 3.11 изображена структура таблицы «Airlines», содержащая информацию о авиакомпаниях. Первичный ключ «airline\_id» хранит уникальный идентификатор авиакомпании, name — название компании, rating — оценка по отзывам.

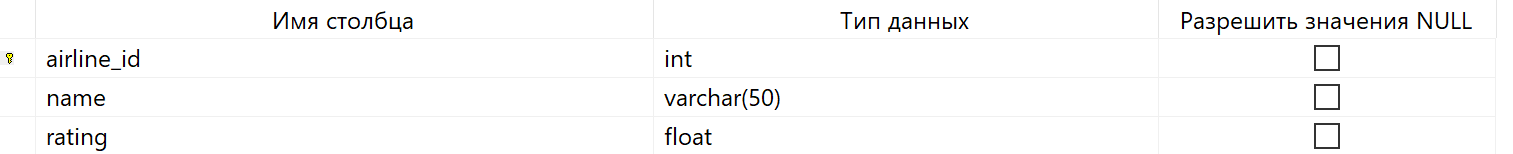


Рисунок 3.11 — Структура таблицы «Airlines»

На рисунке 3.12 изображена структура таблицы «Flights», содержащая информацию о рейсах. Первичный ключ «flight\_id» хранит уникальный идентификатор рейса, airline\_id – id авиакомпании, которая осуществляет авиаперевозку, departure – место отправления, arrival – место прибытия, departure\_date – дата отправления, arrival\_date – дата прибытия, seats\_available – количество свободных мест, class – класс рейса.

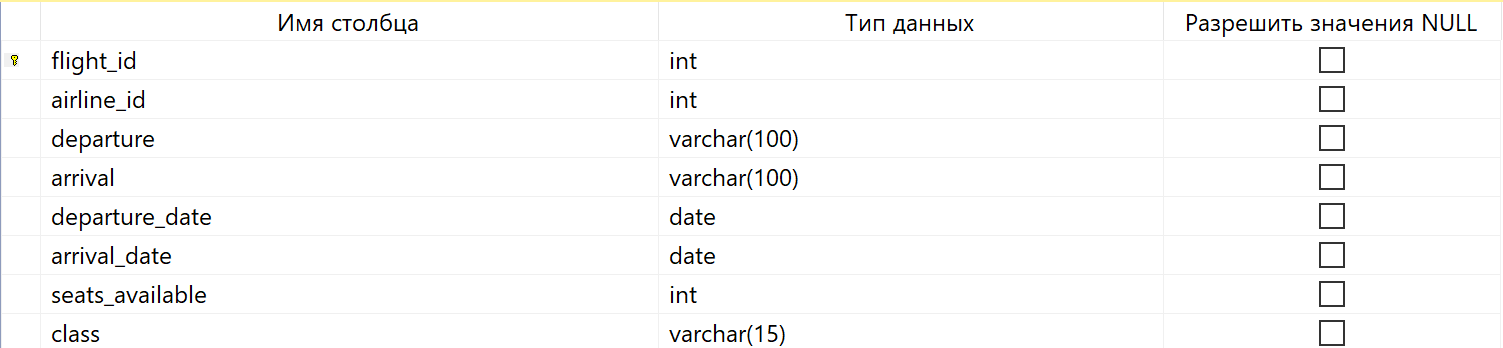


Рисунок 3.12 — Структура таблицы «Flights»

На рисунке 3.13 изображена структура таблицы «Passengers», содержащая информацию о пассажирах. Первичный ключ «passenger\_id» хранит уникальный идентификатор пассажира, flight\_id – идентификатор рейса, first\_name – имя пассажира, last\_name – фамилия пассажира, gender – пол пассажира, birth\_date – дата рождения пассажира.

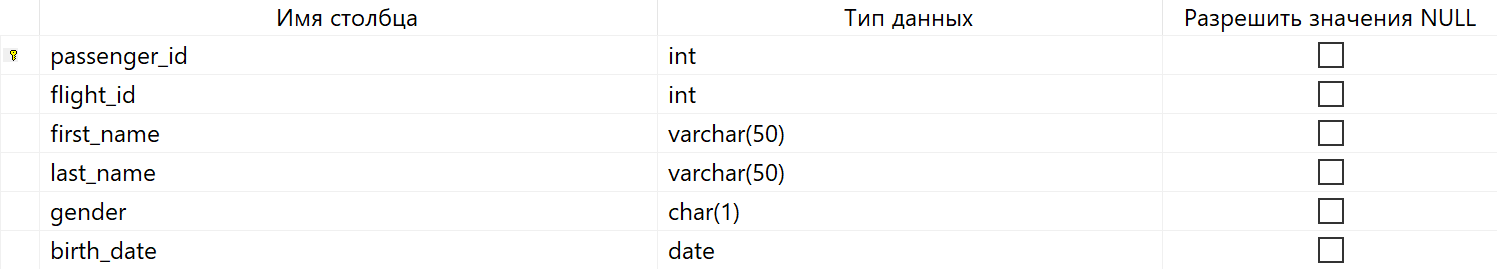


Рисунок 3.13 — Структура таблицы «Passengers»

На рисунке 3.14 изображена структура таблицы «Payments», содержащая информацию о платежах. Первичный ключ «payment\_id» хранит уникальный идентификатор платежа, client\_id – идентификатор пользователя, flight\_id– идентификатор рейса, payment\_date – дата платежа.

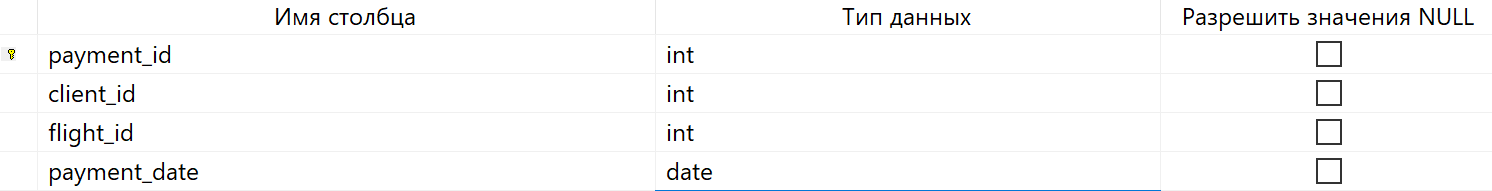


Рисунок 3.14 — Структура таблицы «Payments»

На рисунке 3.15 изображена структура таблицы «Reviews», содержащая информацию о отзывах. Первичный ключ «id» хранит уникальный идентификатор отзыва, airline\_id – идентификатор авиакомпании, client\_id – идентификатор пользователя, review\_text – текст отзыва, score – оценка авиакомпании.

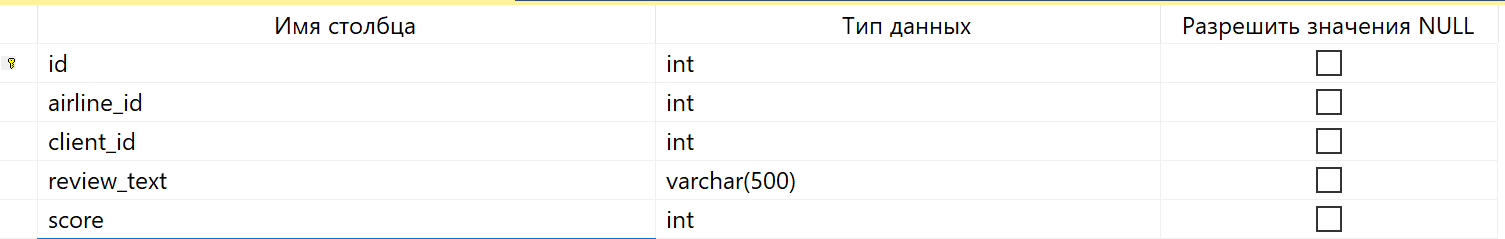


Рисунок 3.15 — Структура таблицы «Reviews»

Создание базы данных позволит эффективно хранить и управлять информацией о пользователях, рейсах и авиакомпаниях, необходимой для функционирования интернет-ресурса.

## Проектирование архитектуры приложения

Для общего представления функционального назначения системы используется диаграмма использования, описывающая, какой функционал разрабатываемой программной системы доступен каждой группе пользователей. На диаграмме использования применяются два типа основных сущностей: варианты использования и группы пользователей.

Разные группы пользователей в диаграмме называются актёрами, и обозначают любые сущности, использующие систему. Любая функция системы называется вариантом использования. Каждый вариант использования обозначает набор действий, который может быть использован актёром для взаимодействия с системой, и определяет набор действий, выполняемых этой системой.

Диаграмма использования представлена в [приложении Б](#_ПРИЛОЖЕНИЕ_Б).

## Проектирование последовательностей проекта

В целях визуализации взаимодействия объектов системы между собой во времени в едином сценарии использования используется ещё одна UML-диаграмма – диаграмма последовательностей. Данная диаграмма иллюстрирует, как различные части системы взаимодействуют друг с другом для выполнения функции, а также порядок, в котором происходит взаимодействие при выполнении конкретного случая использования.

Для отображения течения времени используется линия жизни объекта, которая изображается с помощью штриховой линии, которая проводится вертикально вниз. С помощью линии жизни показывается период, в течение которого объект существует в системе. Сами объекты изображаются в виде прямоугольников, а сообщения, которыми они обмениваются – в виде линий со стрелками.

Диаграмма последовательностей представлена в [приложении В](#_ПРИЛОЖЕНИЕ_В).

Таким образом, благодаря проектированию программного средства удалось создать эффективное и удобное приложение, соответствующее всем требованиям и спецификациям, описанным в предыдущих разделах.

# Реализация программного средства

## Основные классы программного средства

Для выполнения технических задач программного средства для поиска и бронирования авиабилетов должны быть реализованы следующие функции и соответствующие им классы и методы:

* выполнение входа;
* регистрация;
* просмотр рейсов;
* бронирование билетов на рейсы;
* добавление отзывов;
* функционирование панели администратора.

Далее подробно рассмотрены каждые из необходимых для выполнения технических задач функции, а также созданные для их выполнения классы и методы и их функционал и реализация.

## Выполнение входа

Графическая часть формы авторизации описана в классе Account. После нажатия кнопки «Войти» сработает команда LogButton(), преобразующая введённый пользователем пароль в строковое значение для последующей проверки на соответствие с сохранённым в базе данных паролем. Данный метод описан в классе AccountViewModel – логика модели представления для личного кабинета.

При успешной валидации введённых пользователем входных данных выполняется проверка на существование введённого логина в базе данных, и при нахождении значения выполняется проверка на соответствие введённого пароля и хранящегося в базе данных пароля. При успешном прохождении всех проверок пользователь перенаправляется на страницу личного кабинета, а противном случае – получает сообщение о соответствующей ошибке.

Код команды LogButton представлен в [приложении Д.](#_ПРИЛОЖЕНИЕ_Д)

Таким образом, в классе AccountViewModel определены свойства, методы и события, которые обеспечивают взаимодействие с базой данных и обработку действий пользователя при авторизации.

## Регистрация

Для совершения пользователем регистрации нового аккаунта реализована форма в представлении Account.

При переходе к форме регистрации к форме авторизации добавляется 3 поля, для ввода имени, фамилии, почты.

При нажатии пользователя на кнопку «Регистрация» вызывается команда RegButton(). Данный метод находится в классе AccountViewModel – логика модели представления для личного кабинета.

В начале все введённые пользователем данные проходят проверку: для валидации e-mail адреса вызывается метод IsNullOrEmpty(), а также идёт проверка по регулярному выражению, имя, фамилия и пароль также проверяются на корректность с использованием метода IsNullOrEmpty().

После успешного прохождения вышеописанных проверок происходит попытка добавить пользователя в базу данных и перенаправление пользователя на главную страницу. Если логин или адрес электронной почты уже зарегистрирован в базе данных, то пользователь получает соответствующее уведомление об ошибке.

Код команды RegButton() представлен в [приложении Д.](#_ПРИЛОЖЕНИЕ_Д)

Таким образом, класс AccountViewModel содержит логику и данные, связанные с формой регистрации, и предоставляет методы для выполнения регистрации и уведомлений об изменении значений свойств.

## Просмотр рейсов

При запуске приложения мы попадаем на главную страницу, где мы можем используя форму поиска ввести данные для фильтрации рейсов и перейти к результатам нажав кнопку «Найти билеты», либо мы можем перейти на страницу «Выбор билетов» нажав одноименную кнопку в шапке или в подвале приложения. При переходе на страницу «Выбор билетов» загружается графическая часть, которая определена в классе Tickets.

В конструкторе класса происходит загрузка всех рейсов. Также создаются объекты окон, которые будут использоваться для отображения информации о выбранном объявлении и быстрого просмотра.

В классе TicketViewModel реализованы методы для фильтрации и перехода на страницу оплаты билета. После нажатия на кнопку «Поиск», вызывается метод FindResults() – метод находится в классе TicketsViewModel – логика модели представления для страницы со всеми рейсами. Все данные хранятся в виде коллекций.

Код метода FindResults() представлен в [приложении Д.](#_ПРИЛОЖЕНИЕ_Д)

## Бронирование билетов на рейсы

При нажатии на кнопку «Выбрать» определенного рейса на странице «Выбрать билеты», нас перенаправляет на страницу оплаты, графическая часть которой определена в классе PaymentPage.

В конструкторе класса происходит установка значения count, которое определяет количество форм ввода для ввода данных пассажиров.

В классе PaymentViewModel реализованы команды для проверки введеных значений пользователей и добавления данных пассажиров в базу данных. При проверке проверяется каждый обьект коллекции «Passengers», созданной при создании обьекта класса, если хоть одно поле обьекта «Passenger» равно null, то выводится соответствующее сообщение об ошибке. Если все поля заполнены, то каждый обьект коллекции «Passengers» добавляется в базу данных.

Листинг реализации класса PaymentViewModel представлен в [приложении Д.](#_ПРИЛОЖЕНИЕ_Д)

В итоге можно сказать, что класс PaymentViewModel обеспечивает валидацию и обработку данных перед их сохранением в базу данных.

## Добавление отзывов

Графическая часть страницы с отзывами определена в классе Reviews.

Класс ReviewsViewModel является моделью представления для страницы отзывов. Внутри класса определены:

* коллекции для добавления и вывода отзывов;
* команды для отображение меню добавления отзыва, отправки отзыва, а также вывода всех отзывов определенной компании;
* поле SelectedAirline для хранения выбранной пользователем авиакомпании в списке;
* конструктор класса, который получает все элементы таблицы Airlines, для заполнения списка авиакомпаний.

Листинг реализации класса ReviewsViewModel представлен в [приложении Д.](#_ПРИЛОЖЕНИЕ_Д)

В общем, класс предоставляет функционал для просмотра и добавления отзывов для выбранной авиакомпании.

## Функционирование панели администратора

Все необходимые администратору команды находятся в AccountViewModel.

При входе в личный кабинет пользователя проверяется роль пользователя, если она равна «admin», то в личном кабинете отображается кнопка «Панель управления» при нажатии на которую отображается панель управления администратора. Она состоит из 4 кнопок.

Первая кнопка «Управление рейсами», при нажатии на которую появляется список всех рейсов, которые есть в базе данных, при нажатии на кнопку «Удалить рейс», выбранный рейс удаляется из списка и из базы данных.

Вторая кнопка «Управление пользователями», при нажатии на кнопку выводится список всех пользователей, у каждого пользователя имеется кнопка «изменить роль», при нажатии на которую роль пользователя изменяется на роль администратора, если роль была «user», то после смены будет «admin».

Третья кнопка «Просмотр пассажиров на рейс», при нажатии открывается меню в котором можно ввести номер рейса и при нажатии на кнопку «Найти пассажиров» найти всех пассажиров для этого рейса. При нажатии на кнопку выводится список найденных пассажиров для этого рейса.

Четвертая кнопка «Заполнить базу данных» отвечает за заполнение базы данных данными.

Листинг реализации класса AccountViewModel представлен в [приложении Д.](#_ПРИЛОЖЕНИЕ_Д)

Таким образом, в разделе подробно описаны основные классы и команды, используемые для реализации функциональности программного средства для поиска и бронирования авиабилетов.

# Тестирование, проверка работоспособности и анализ полученных результатов

## Тестирование авторизации и регистрации

При тестировании данного приложения, были применены сценарии, которые могли бы привести к ошибке. В этой главе мы рассмотрим некоторые такие сценарии и посмотрим на их обработку.

В момент авторизации, возможна такая ситуация, в которой пользователь ничего не ввёл. Обработка данного сценария приведена на рисунке 5.1.

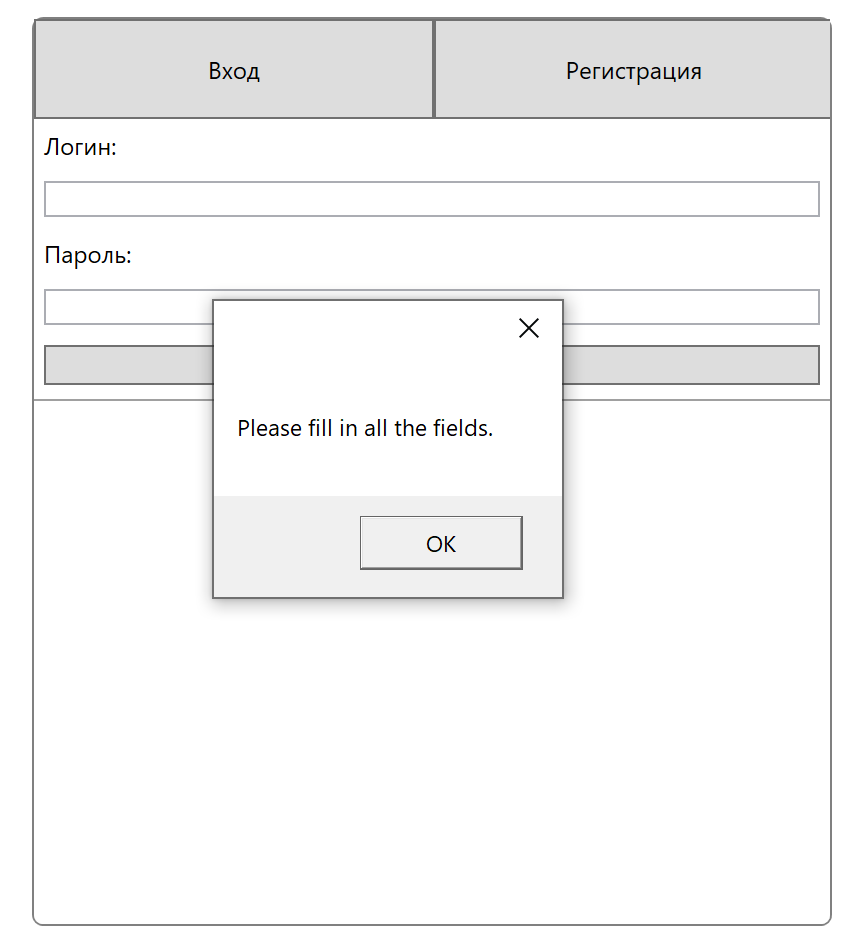


Рисунок 5.1 — Обработка пустых полей при авторизации

Как видно с рисунка, было выведено окно с сообщением «Please fill in all the fields».

При вводе некорректных данных, возникает ошибка, представленная на рисунке 5.2.

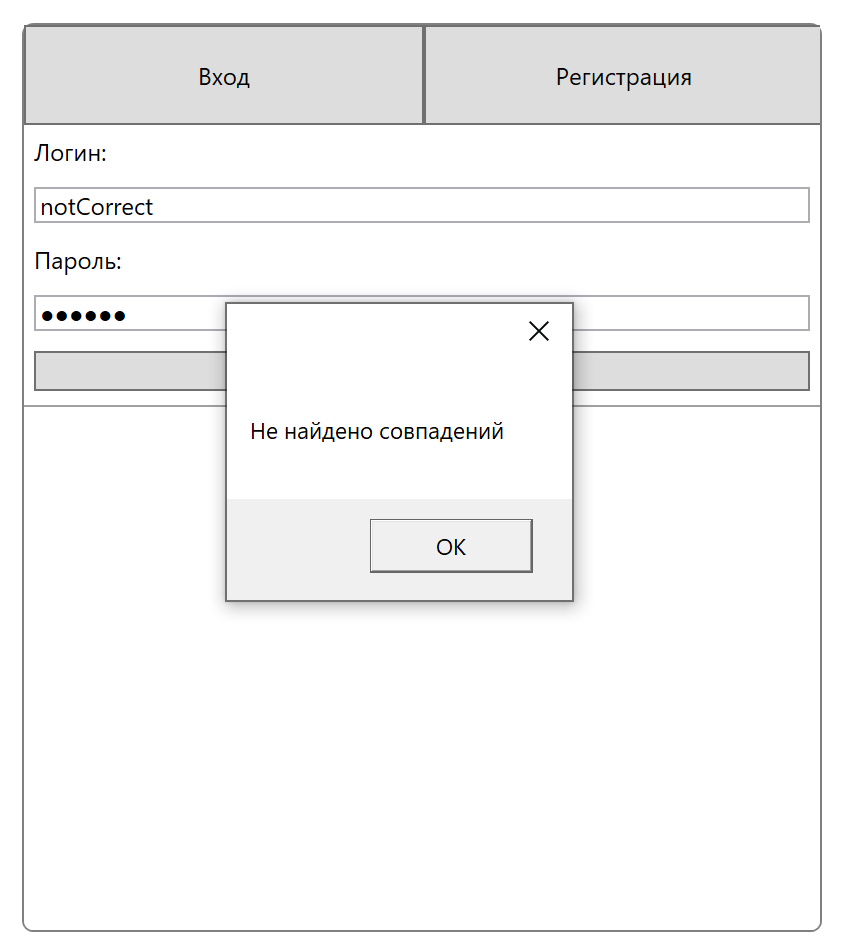


Рисунок 5.2 — Обработка некорректных данных

Как видно с рисунка, было выведено окно с сообщением «Не найдено совпадений».

Также возможен такие случаи:

Пользователь хочет зарегистрироваться, при этом не указав никаких данных (рисунок 5.3).

Пользователь указывает электронную почту, которая уже привязана к учётной записи (рисунок 5.4)

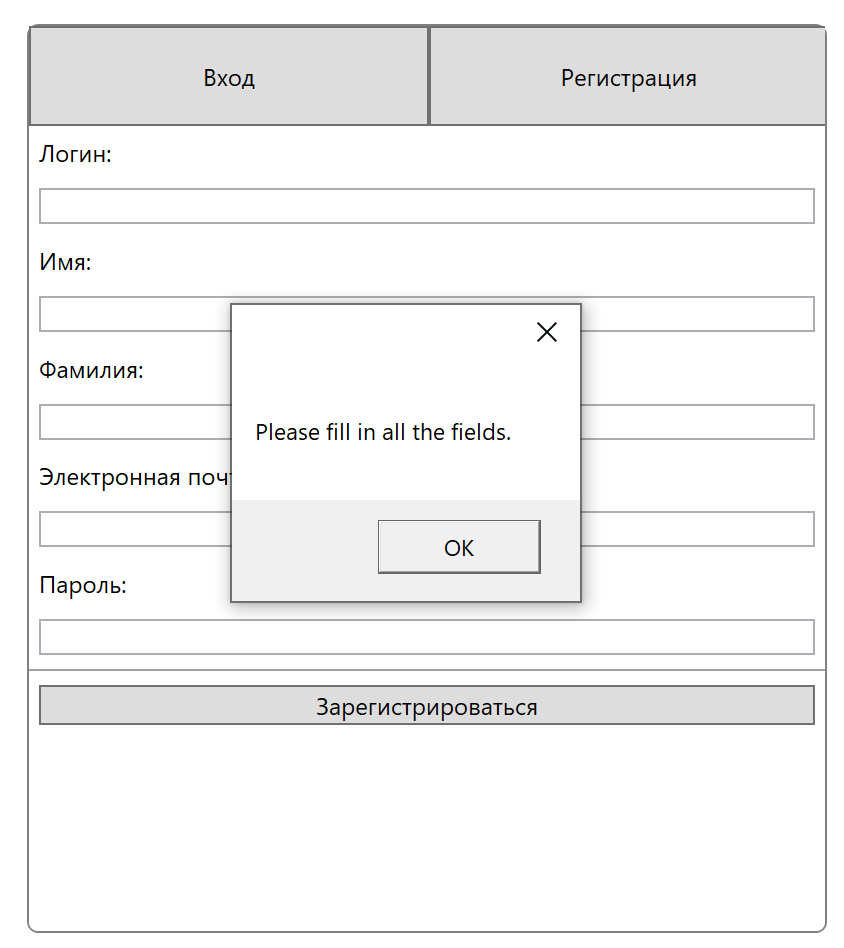


Рисунок 5.3 — Обработка пустых полей при регистрации

Как видно с рисунка, выводится окно с просьбой заполнить все поля.

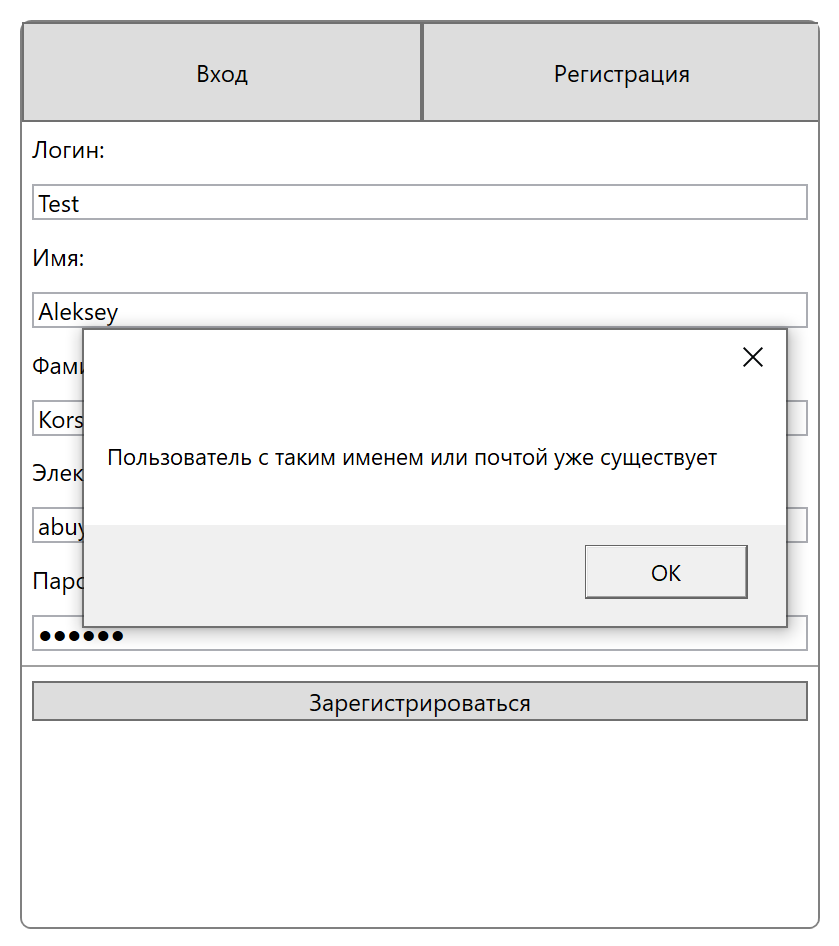


Рисунок 5.4 — Обработка ввода уже зарегистрированного Mail

Как видно с рисунка, было выведено окно с сообщением «Пользователь с таким именем или почтой уже есть».

## Тестирование поиска билетов на рейсы

При заполнении формы для поиска билетов па заданным параметрам, если все поля заполнены, то нас перенаправит на страницу «Выбор билетов», где в случае если есть рейсы которые подходят по параметрам, то будут выведены результаты, в ином случае будет выведено сообщение что рейсы не найдены. Форма для поиска билетов представляет собой 5 полей и 1 кнопку. Первые два поля – текстовые поля для ввода места отправления и места прибытия. Следующие два поля – поля для ввода даты, которые имеют кнопку, при нажатии на которую открывается календарь где можно выбрать нужную дату. Проверка введенных дат осуществляется путем сравнения чтобы дата отправления была не раньше чем текущий день. Последнее поле представляет собой кнопку, при нажатии на которую открывается выпадающее меню, где можно выбрать количество пассажиров и класс рейса. Примеры обработки исключительных ситуаций представлены на рисунках 5.5 – 5.6.



Рисунок 5.5 — Обработка пустого поискового запроса

Как видно с рисунка, было выведено окно с сообщением «Заполните все поля корректно».

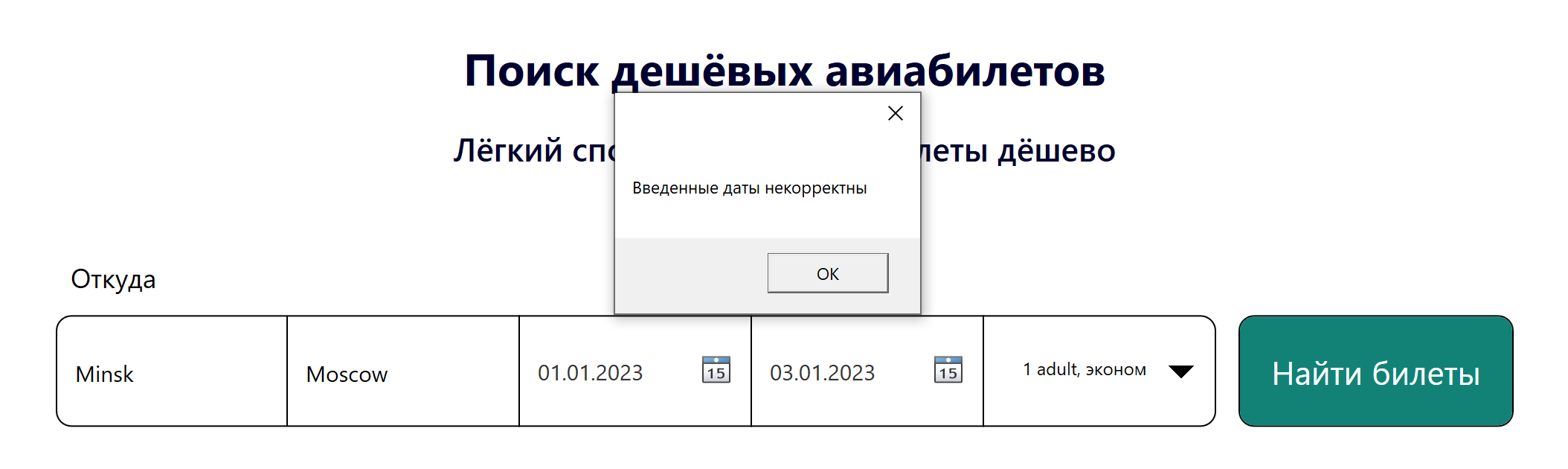


Рисунок 5.6 — Обработка поискового запроса с неверными датами

Как видно с рисунка, было выведено окно с сообщением «Введенные даты некорректны».

## Тестирование добавления и просмотра отзывов

При просмотре и добавлении отзывов могут возникать исключительные ситуации. Отзывы авиакомпании показываются только если выбрана какая-нибудь авиакомпания в выпадающем списке. Меню с добавлением отзыва открывается только если пользователь авторизован в личном кабинете. Добавление отзыва происходит только если выбрана авиакомпания. Примеры обработки исключительных ситуаций представлены на рисунках 5.7 – 5.13.

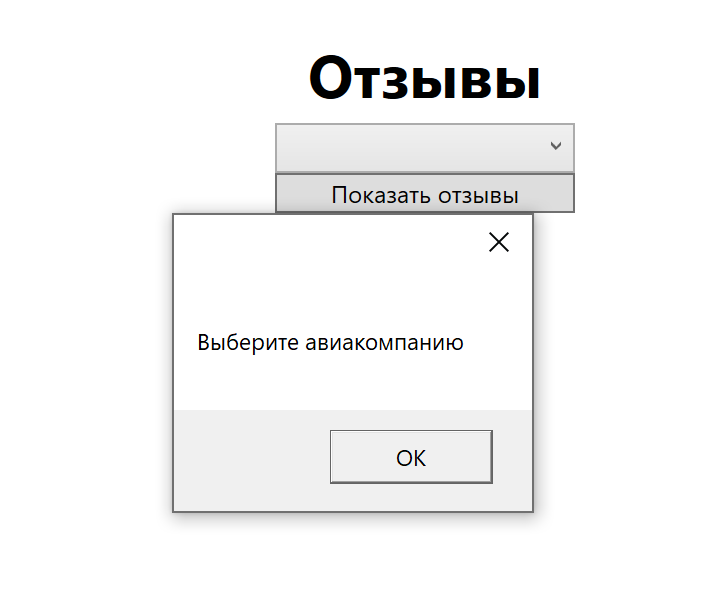


Рисунок 5.7 — Попытка отображения отзывов при не выбранной авиакомпании

Как видно с рисунка, было выведено окно с сообщением «Выберите авиакомпанию».

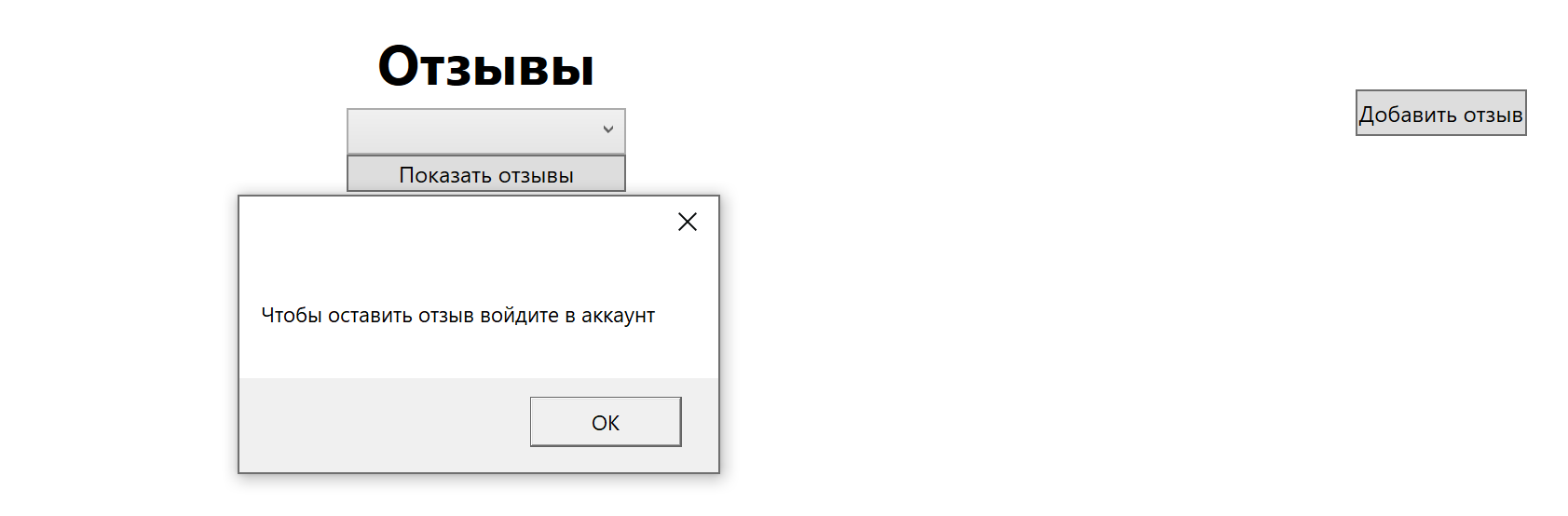


Рисунок 5.8 — Попытка открыть меню добавления отзыва не авторизовавшись в личном кабинете

Как видно с рисунка, было выведено окно с сообщением «Чтобы оставить отзыв войдите в аккаунт».

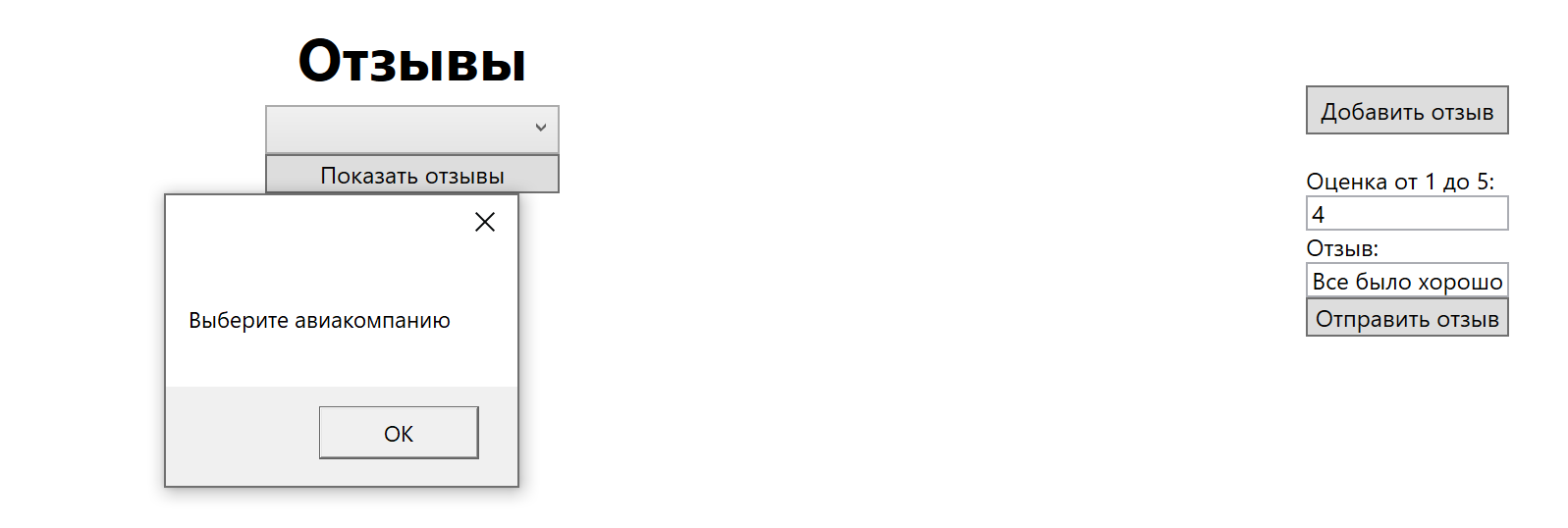


Рисунок 5.9 — Попытка добавить отзыв не выбрав авиакомпанию в выпадающем списке

Таким образом, в данном разделе были рассмотрены обработки всех исключений, которые могут быть вызваны пользователем при использовании программного средства.

# Руководство по использованию

При запуске приложения мы попадаем на главную страницу (рисунок 6.1).

Чтобы перейти на другие страницы можно воспользоваться кнопками в верхней или нижней части приложения. Для того чтобы найти рейсы, нужно заполнить все поля формы поиска и нажать на кнопку «Найти билеты».

Нажав на кнопку «Найти билеты» нас перенаправляет на страницу с рейсами, где в случае если рейсы найдены будет выведен список рейсов, в ином случае будет выведено сообщение что рейсы не были найдены.

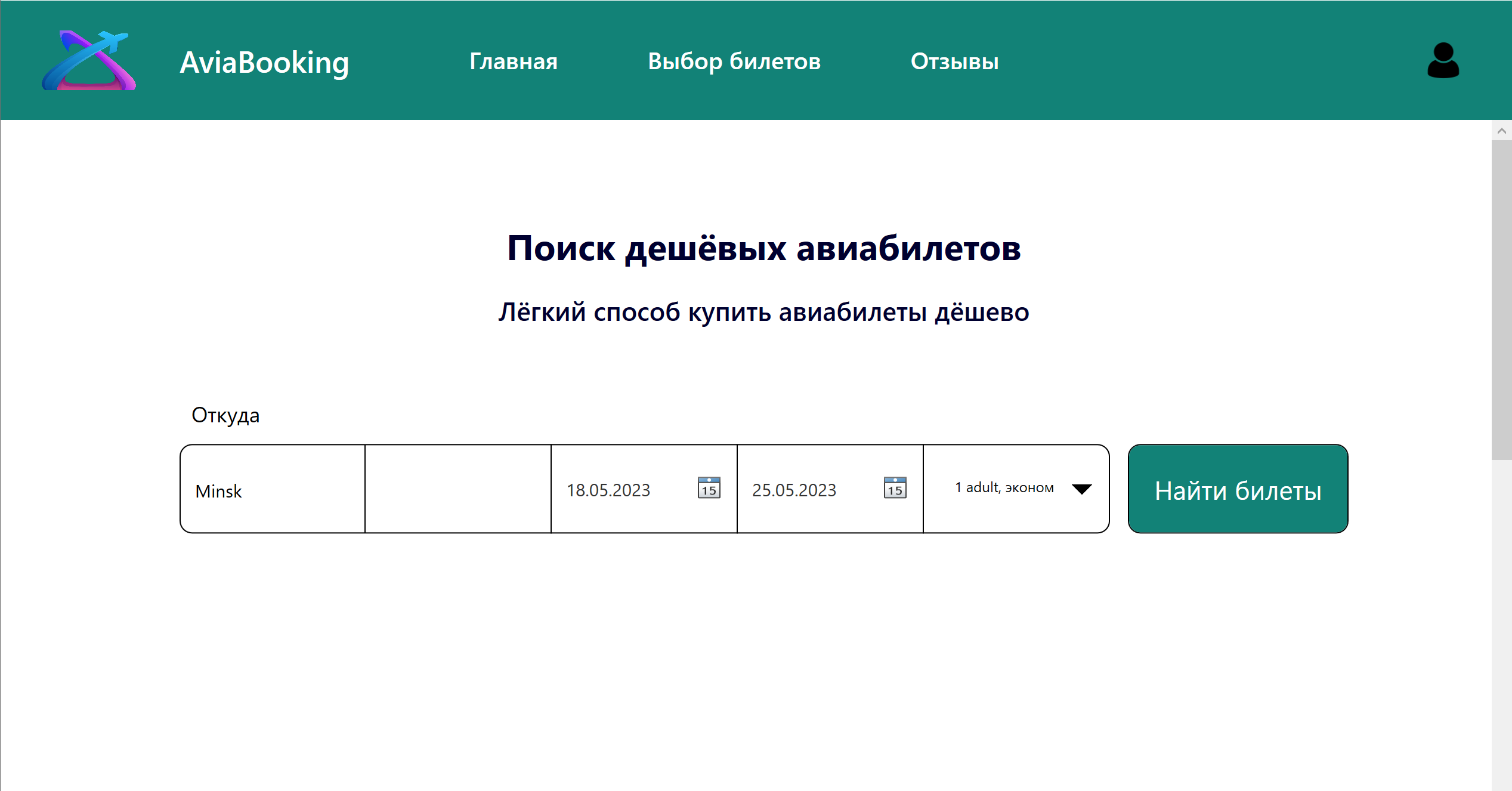


Рисунок 6.1 – Главная страница

После поиска билетов мы попадаем на страницу «Выбор билетов», где мы можем просмотреть найденные рейсы и изменить фильтры поиска.

В левой части страницы имеется меню с полями для фильтрации рейсов.

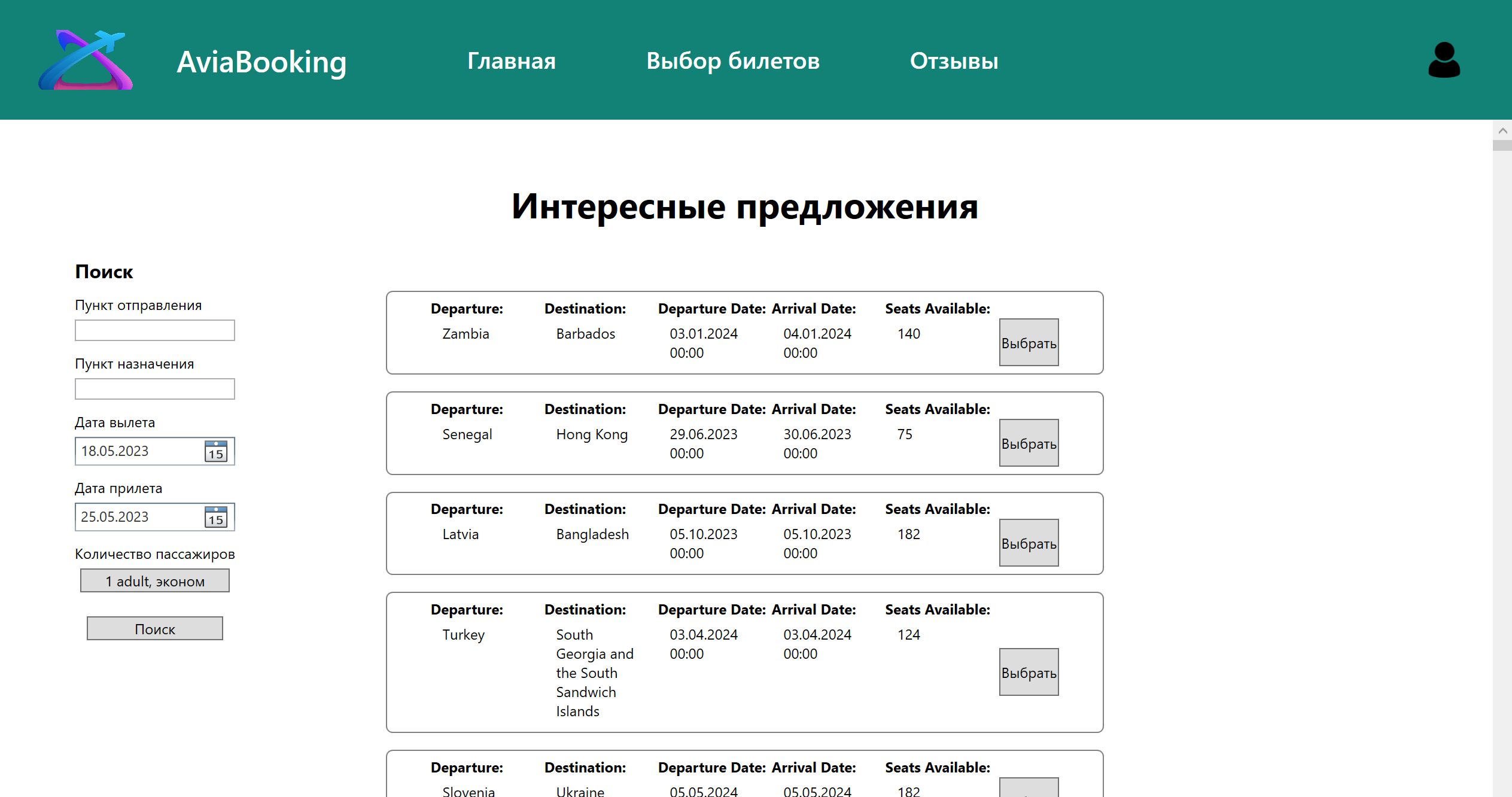


Рисунок 6.2 – Выбор билетов

Для поиска нужных билетов нам нужно данные, такие как, место отправления, место прибытия, дата вылета, дата прилета в меню фильтрации.

После этого, найденные рейсы будут отображены справа от меню фильтрации.

Ниже показан рисунок 6.3, на котором показана страница «Отзывы».

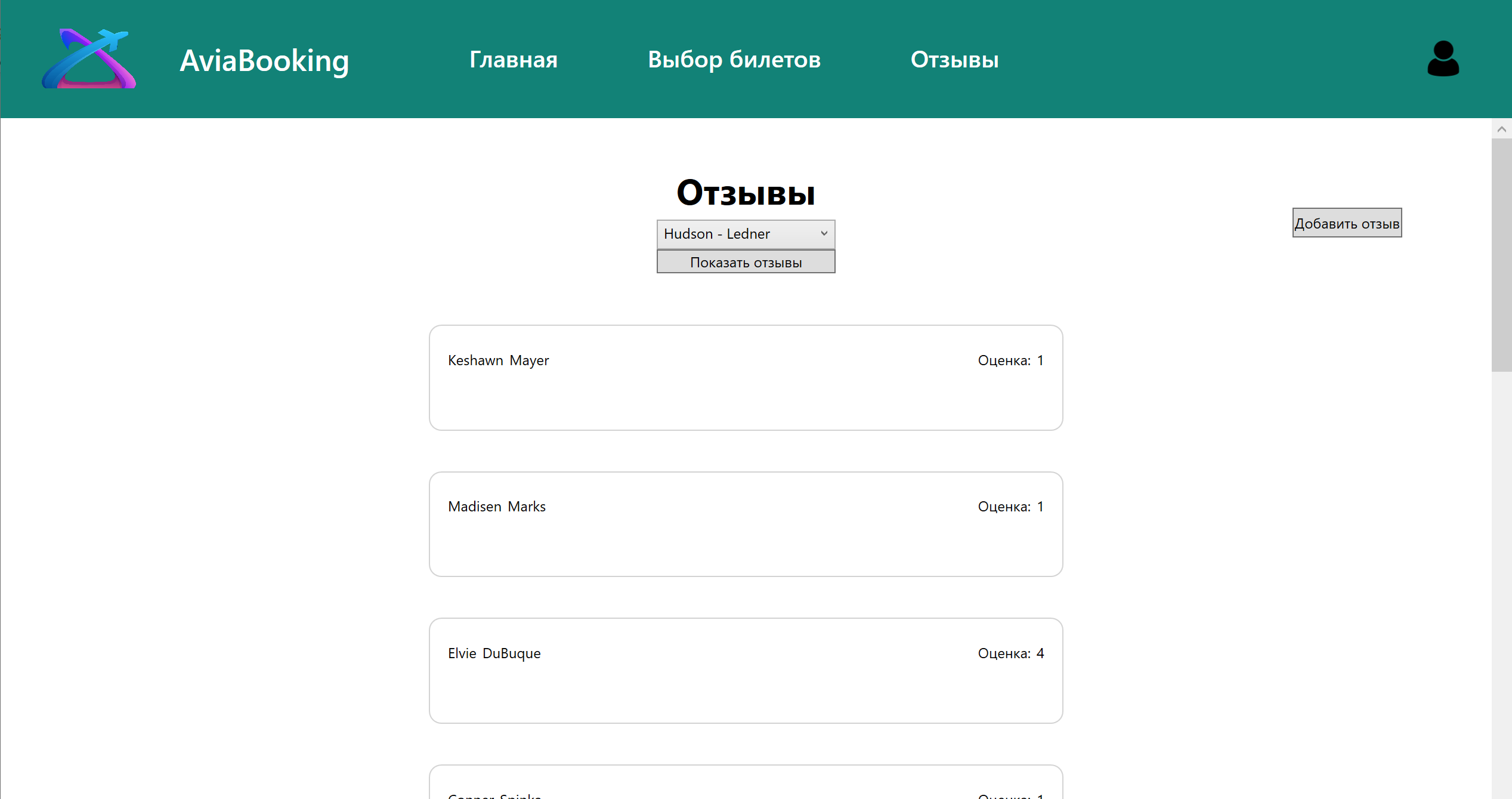


Рисунок 6.3 – Страница «Отзывы»

Ниже показана страница бронирования рейса (рисунок 6.4).

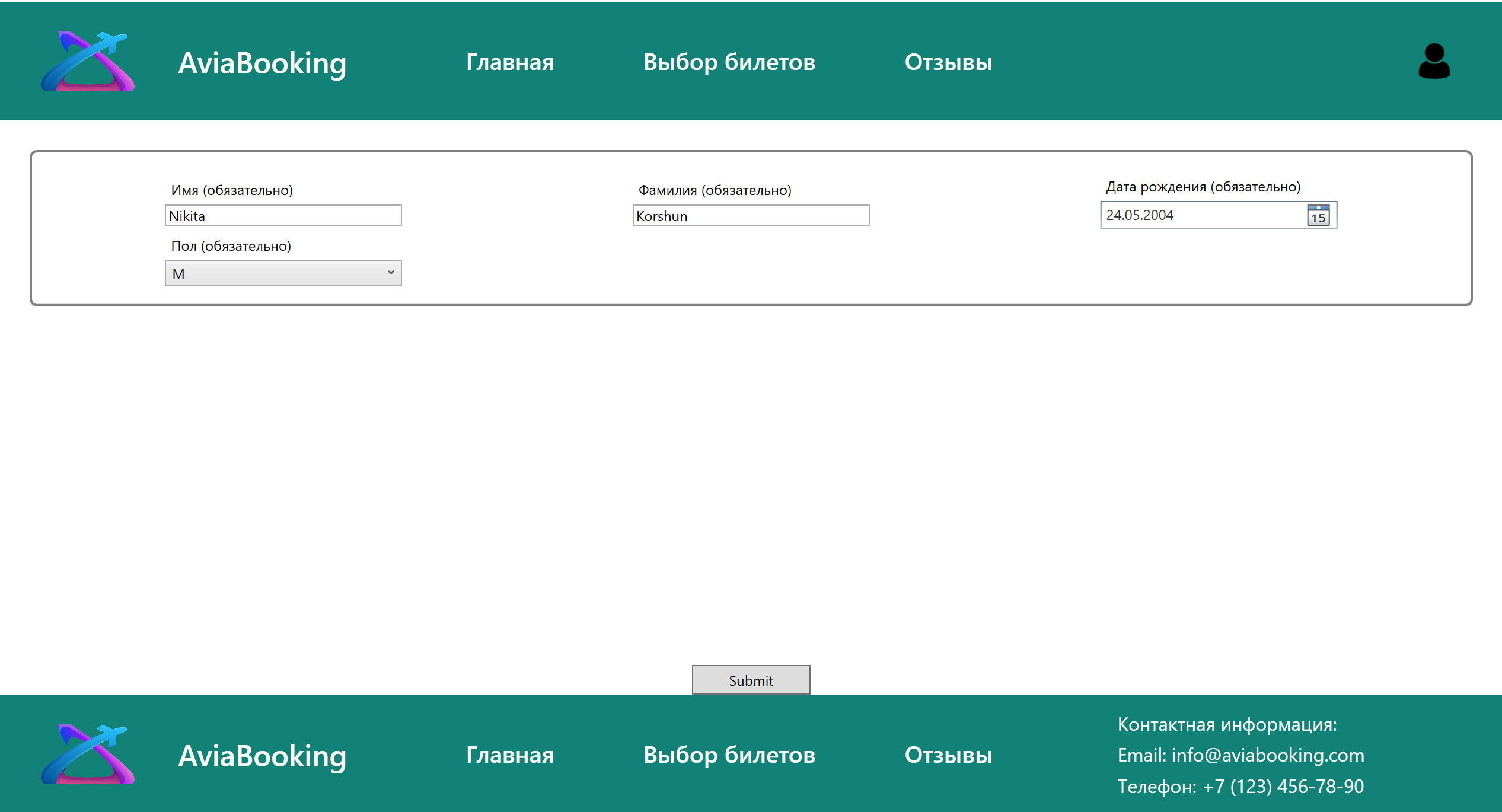


Рисунок 6.4 – Страница бронирования рейса

Ниже показана страница авторизации (рисунок 6.5).

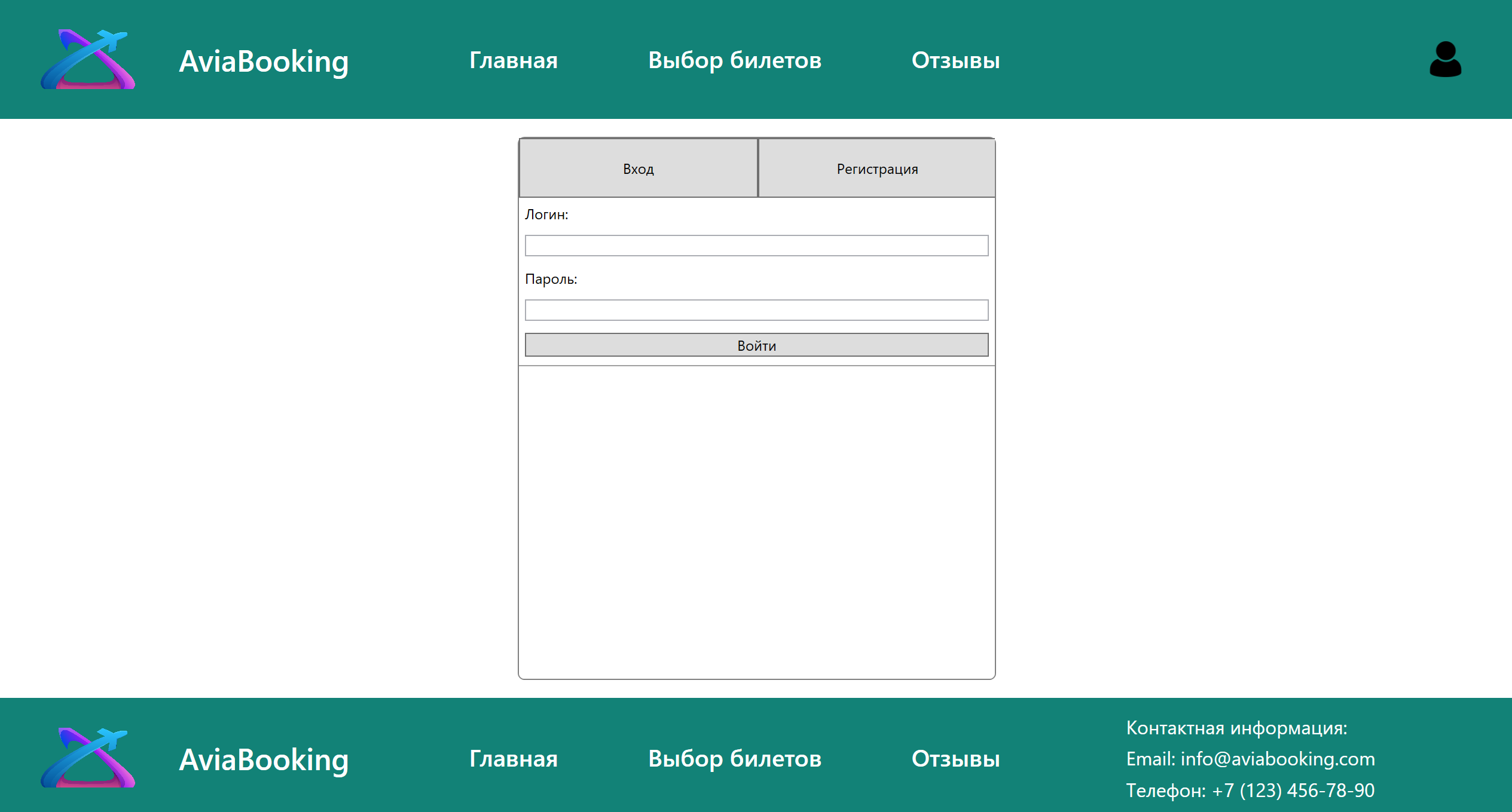


Рисунок 6.5 – Страница авторизации

Ниже показана страница регистрации (рисунок 6.6).

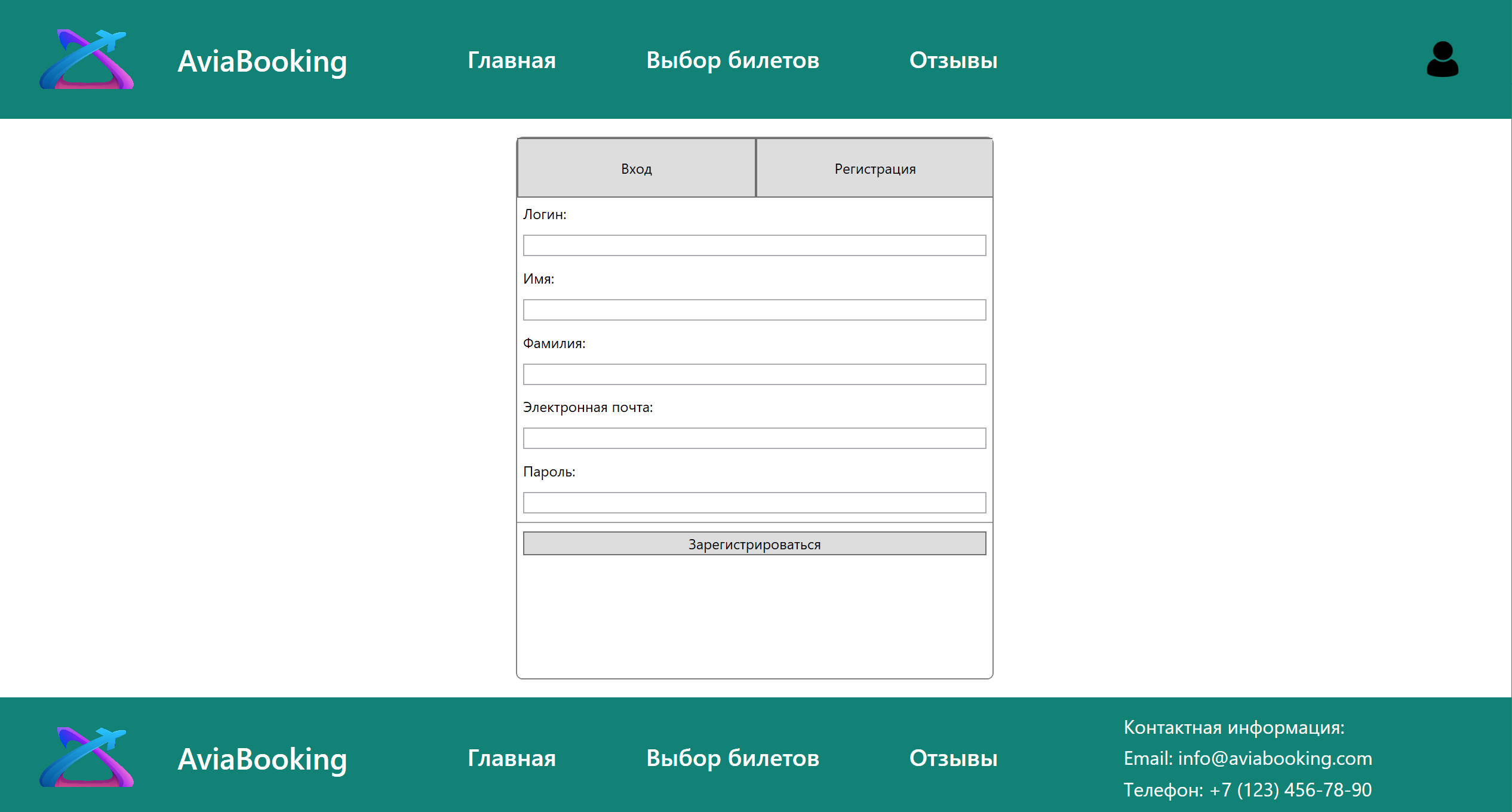


Рисунок 6.6 – Страница регистрации

Ниже показана страница личного кабинета пользователя (рисунок 6.7).

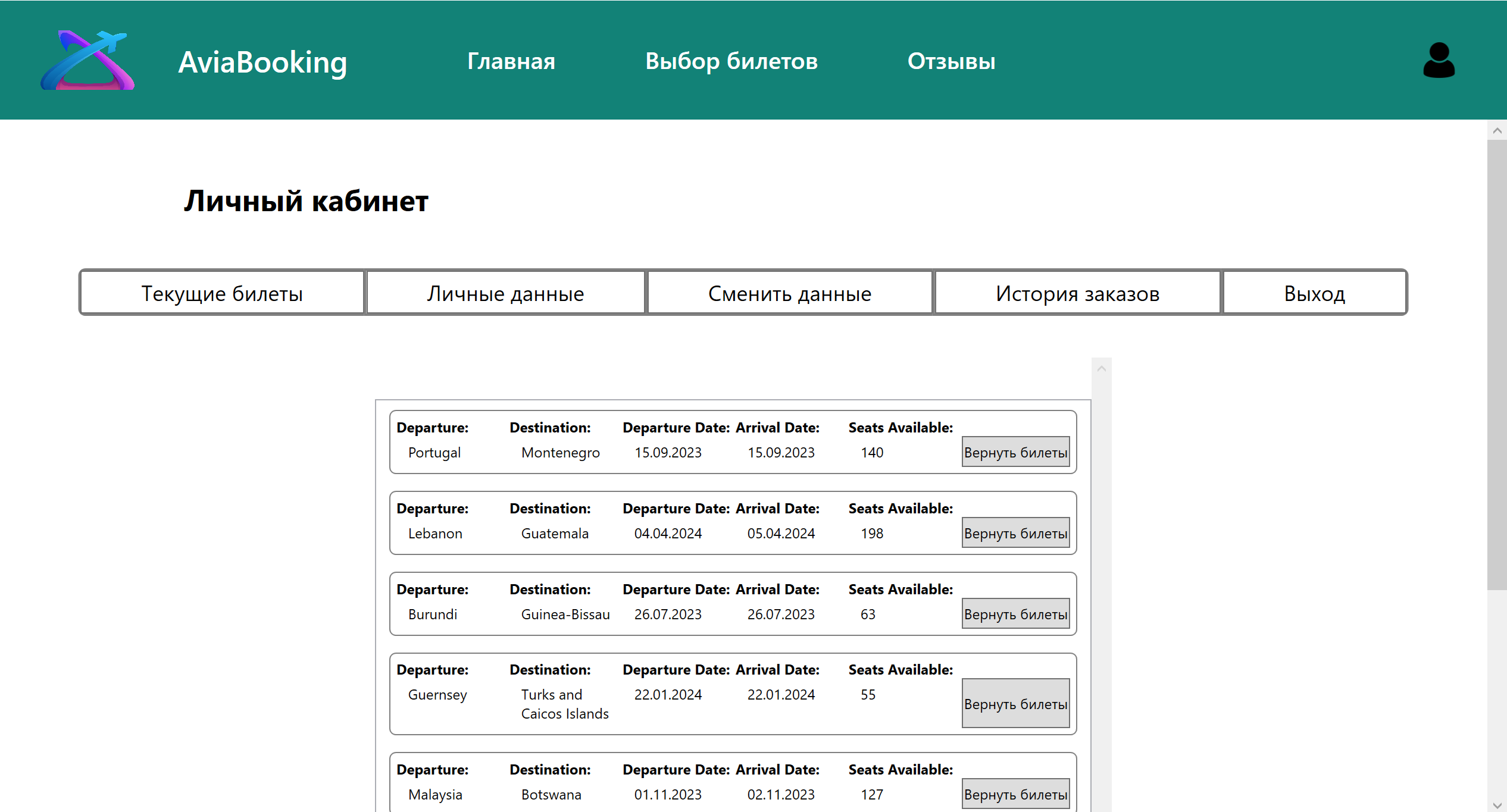


Рисунок 6.7 – Страница личного кабинета пользователя

Также на странице «Личный кабинет» есть кнопки, нажав на которые будут открыты соответствующие названиям окна:

* текущие билеты;
* личные данные;
* сменить данные
* история заказов.

При переходе на страницу «Личный кабинет» мы попадаем на страницу, в которой отображены наши текущие рейсы, а также личные данные которые мы можем просмотреть нажав на кнопку «Личные данные», а также изменить, нажав на кнопку «Сменить данные».

На рисунке 6.8 показана панель управления администратора администратора.

Есть 3 вкладки: Управление рейсами, Управление пользователями, Просмотр пассажиров на рейс.

На этой странице можно совершать действия над рейсами, изменять привилегии пользователей и просматривать список пассажиров на рейсы.

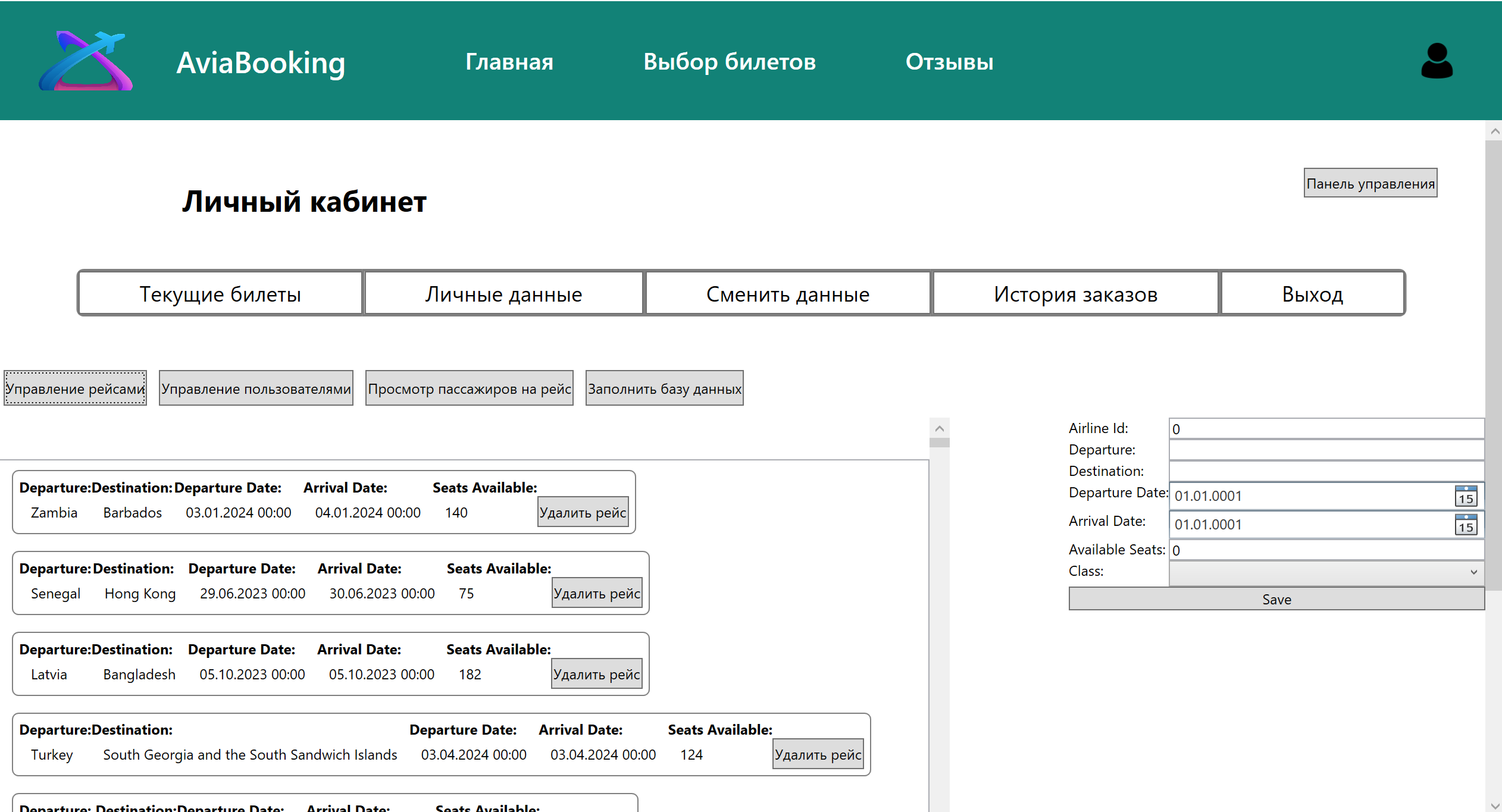


Рисунок 6.8 – Панель администратора

В целом, данный раздел предоставляет всю необходимую информацию, чтобы пользователи могли успешно использовать приложение, искать и бронировать билеты на рейсы, просматривать и добавлять отзывы авиакомпаниям, а администраторы имели контроль над рейсами и пользователями.

# ЗАКЛЮЧЕНИЕ

В данном курсовом проекте было разработано программное средство для поиска и бронирования авиабилетов при помощи языка программирования C#, API-интерфейса Windows Presentation Foundation, технологии Entity Framework и базы данных Microsoft SQL Server. При выполнении курсового проекта использовались принципы и приемы ООП.

При разработке программного средства были выполнены все пункты из указанного списка предполагаемого основного функционала приложения, а именно:

Для пользователя:

* регистрация;
* авторизация;
* изменение данных своего профиля;
* бронирование билетов на рейсы;
* просмотр доступных рейсов по разным критериям;
* возврат забронированных билетов;
* добавление отзывов авиакомпаниям;
* просмотр отзывов авиакомпаний;

Для администратора:

* авторизация;
* изменение данных своего профиля;
* просмотр информации о пользователях;
* изменение привелегий у пользователей;
* добавление новых рейсов;
* просмотр доступных рейсов;
* удаление рейсов;
* просмотр пассажиров на определенный рейс;

Тестирование программы показало, что она работает корректно и выполняет все свои функции.

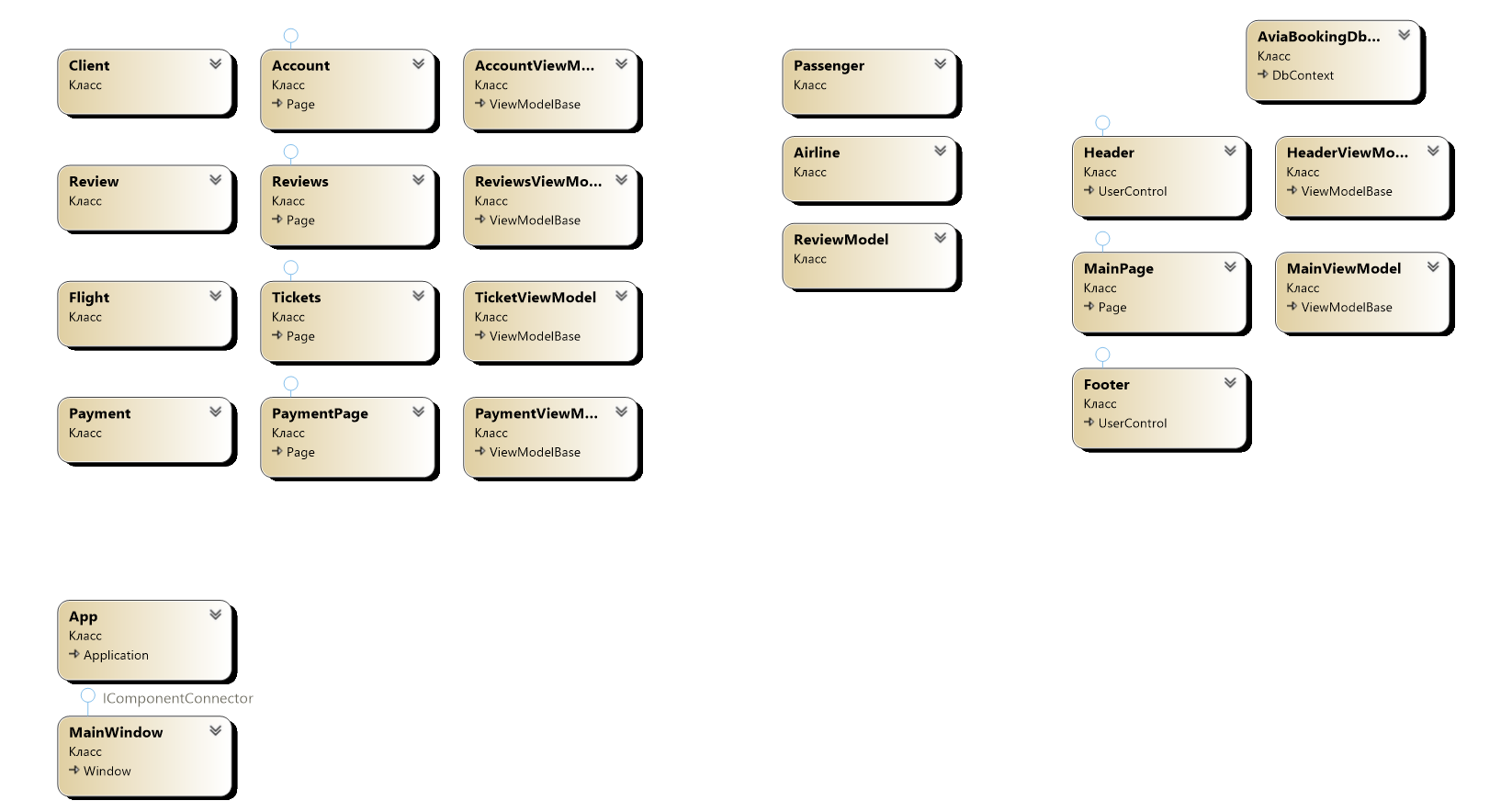
В соответствии с полученным результатом работы программы можно сделать вывод, что разработанная программа работает верно, а требования технического задания выполнены в полном объеме.

# Список использованных источников

1. Aviasales [Электронный ресурс] – https://www.aviasales.by/– Дата доступа 12.03.2023
2. Kayak [Электронный ресурс] – https://www.kayak.com/ – Дата доступа 13.03.2023
3. Skyscanner [Электронный ресурс] – https://ru.skyscanner.com/ – Дата доступа 15.03.2023
4. Microsoft Visual Studio [Электронный ресурс] – https://ru.wikipedia.org/wiki/Microsoft\_Visual\_Studio – Дата доступа 23.04.2023
5. Полное руководство по языку программирования C# 7.0 и платформе .NET 4.7. Режим доступа: https://metanit.com/sharp/tutorial/ – Дата доступа: 23.04.2023
6. Пацей, Н.В. Курс лекций по языку программирования C# / Н. В. Пацей. – Минск: БГТУ, 2018. – 175 с.
7. Руководство по WPF // [Электронный ресурс]. – Режим доступа: https://metanit.com/sharp/wpf/ – Дата доступа: 25.04.2023
8. Руководство по XAML // [Электронный ресурс]. – Режим доступа: https://www.tutorialspoint.com/xaml/index.htm– Дата доступа: 25.04.2023
9. Работа с Entity Framework Core [Электронный ресурс] – https://professorweb.ru/my/entity-framework/6/level1/ – Дата доступа 26.04.2023
10. Блинова, Е.А. Курс лекций по Базам данным / Е.А. Блинова. – Минск: БГТУ, 2019. – 175 с.

# ПРИЛОЖЕНИЕ А

Диаграмма классов



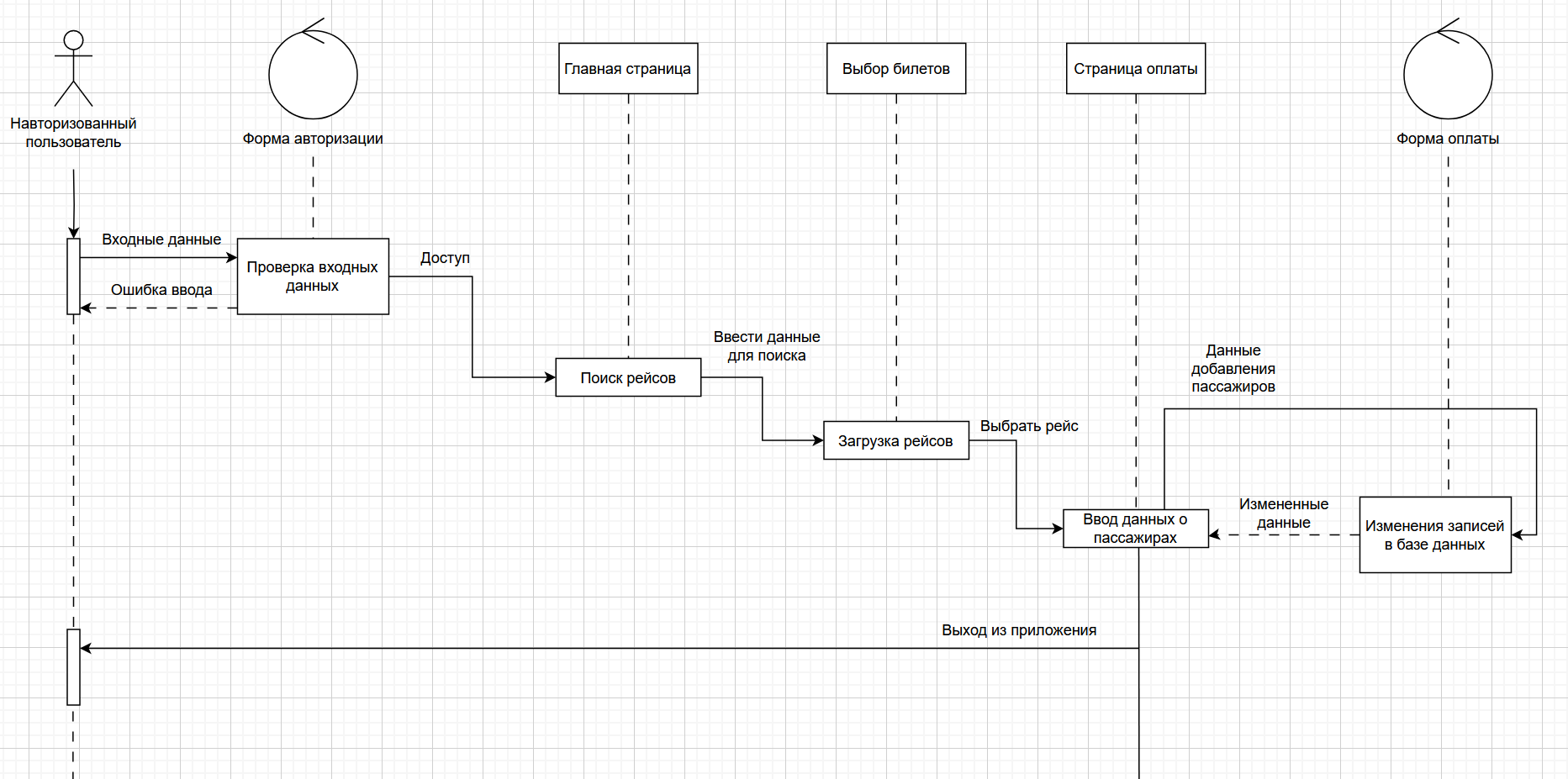
# ПРИЛОЖЕНИЕ Б

Диаграмма использований



# ПРИЛОЖЕНИЕ В

Диаграмма последовательностей



# ПРИЛОЖЕНИЕ Г

Листинг – База данных AviaBooking

|  |
| --- |
| CREATE DATABASE AviaBooking;  USE AviaBooking;  CREATE TABLE Airlines (  airline\_id INT IDENTITY(1,1) PRIMARY KEY,  name VARCHAR(50),  rating FLOAT  );  CREATE TABLE Clients (  client\_id INT IDENTITY(1,1) PRIMARY KEY,  nickname VARCHAR(30),  name VARCHAR(50),  surname VARCHAR(50),  email VARCHAR(50),  password VARCHAR(256),  role VARCHAR(10)  );  CREATE TABLE Flights (  flight\_id INT IDENTITY(1,1) PRIMARY KEY,  airline\_id INT,  departure VARCHAR(50),  arrival VARCHAR(50),  departure\_date DATE,  arrival\_date DATE,  seats\_available INT,  class VARCHAR(10),  FOREIGN KEY (airline\_id) REFERENCES Airlines (airline\_id)  );  CREATE TABLE Payments (  payment\_id INT IDENTITY(1,1) PRIMARY KEY,  client\_id INT,  flight\_id INT,  payment\_date DATE,  FOREIGN KEY (client\_id) REFERENCES Clients(client\_id),  FOREIGN KEY (flight\_id) REFERENCES Flights(flight\_id)  );  CREATE TABLE Passengers (  passenger\_id INT NOT NULL PRIMARY KEY IDENTITY,  first\_name VARCHAR(50) NOT NULL,  last\_name VARCHAR(50) NOT NULL,  gender CHAR(1) NOT NULL CHECK (gender IN ('М', 'Ж')),  birth\_date DATE NOT NULL,  flight\_id INT NOT NULL,  FOREIGN KEY (flight\_id) REFERENCES Flights(flight\_id)  );  CREATE TABLE Reviews  (  id int NOT NULL IDENTITY(1,1) PRIMARY KEY,  airline\_id int NOT NULL,  client\_id int NOT NULL,  review\_text varchar(500) NOT NULL,  score int NOT NULL,  FOREIGN KEY (airline\_id) REFERENCES Airlines(airline\_id),  FOREIGN KEY (client\_id) REFERENCES Clients(client\_id)  ); |

# ПРИЛОЖЕНИЕ Д

Листинг – Код команды LogButton

|  |
| --- |
| public ICommand LogButton => \_logButton ??= new DelegateCommand<object>(parameter =>  {  PasswordBox box = (PasswordBox)parameter;  LogPassword = box.Password;  if (string.IsNullOrEmpty(LogNickname) || string.IsNullOrEmpty(LogPassword))  {  MessageBox.Show("Please fill in all the fields.");  } else  {  using (var db = new AviaBookingDbContext())  {  var client = db.Clients.FirstOrDefault(f => f.Nickname == LogNickname && f.Password == box.Password);  if(client!= null)  {  LoginPageVisibility = Visibility.Collapsed;  AccountPageVisibility = Visibility.Visible;  CurrentTicketsVisibility = Visibility.Visible;  TicketsNotFoundVisibility = Visibility.Collapsed;  LogNickname = client.Nickname;  LogName = client.Name;  LogEmail = client.Email;  LogSurname = client.Surname;  ClientID = client.Id;  TempNickname = LogNickname;  TempName = LogName;  TempSurname = LogSurname;  TempEmail = LogEmail;  TempPassword = LogPassword;  for(int i = 0; i < LogPassword.Length; i++)  {  LogSecuredPassword += '\*';  }  // Получаем все записи из таблицы Payments, где клиент совпадает с искомым клиентом.  var payments = db.Payments.Where(p => p.ClientId == client.Id).ToList();  // Получаем все рейсы, которые соответствуют найденным записям в таблице Payments.  var flights = new List<Flight>();  foreach (var payment in payments)  {  var flight = db.Flights.FirstOrDefault(f => f.Id == payment.FlightId && f.DepartureDate > DateTime.Today);  if (flight != null)  {  flights.Add(flight);  }  }  if (flights.Count > 0)  Flights = new ObservableCollection<Flight>(flights);  else  {  TicketsNotFoundVisibility = Visibility.Visible;  CurrentTicketsVisibility = Visibility.Collapsed;  }  if(client.Role == "admin")  {  isAdmin = true;  AdminButtonVisibility = Visibility.Visible;  }  }  else  {  MessageBox.Show("Не найдено совпадений");  }  }  }    }, parameter => parameter is PasswordBox box); |

Листинг – Код команды RegButton

|  |
| --- |
| public ICommand RegButton => \_regButton ??= new DelegateCommand<object>(parameter =>  {  PasswordBox box = (PasswordBox)parameter;  string LogPassword = GetSecureHash(box.SecurePassword);  if (string.IsNullOrEmpty(LogNickname) || string.IsNullOrEmpty(LogName) ||  string.IsNullOrEmpty(LogSurname) || string.IsNullOrEmpty(LogEmail) ||  string.IsNullOrEmpty(LogPassword))  {  MessageBox.Show("Please fill in all the fields.");  }  if (!string.IsNullOrEmpty(LogEmail))  {  if (!LogEmail.Contains("@") || !LogEmail.Contains("."))  {  MessageBox.Show("Please enter a valid email address.");  }  else  {  using (var db = new AviaBookingDbContext())  {  var checkClient = db.Clients.FirstOrDefault(p => p.Nickname == LogNickname || p.Email == LogEmail);  if(checkClient == null)  {  var client = new Client();  client.Nickname = LogNickname;  client.Name = LogName;  client.Surname= LogSurname;  client.Email= LogEmail;  client.Password= box.Password;  client.Role = "user";  db.Clients.Add(client);  db.SaveChanges();  RegButtonVisibility = Visibility.Collapsed;  LogButtonVisibility = Visibility.Visible;  }  else  {  MessageBox.Show("Пользователь с таким именем или почтой уже существует");  }  }  }  }  }, parameter => parameter is PasswordBox box); |

Листинг – Код метода FindResults

|  |
| --- |
| public void FindResults()  {  using (var db = new AviaBookingDbContext())  {  int sumCount = AdultsCount + ChildrenCount;  var flights = db.Flights  .Where(f => f.Departure == Departure && f.Destination == Destination &&  f.DepartureDate == DepartureDate && f.ArrivalDate == ArrivalDate &&  f.AvailableSeats >= sumCount &&  f.Class == Class)  .ToList();  Flights = new ObservableCollection<Flight>(flights);  if (Flights.Count == 0)  {  FlightsNotFoundVisibility = Visibility.Visible;  FlightListVisibility = Visibility.Collapsed;  }  else  {  FlightsNotFoundVisibility = Visibility.Collapsed;  FlightListVisibility = Visibility.Visible;  }  }  } |

Листинг – Код класса PaymentViewModel

|  |
| --- |
| public class PaymentViewModel : ViewModelBase  {  public Flight flight { get; set; }  public Passenger passenger { get; set; }  public int count { get; set; }  public PaymentViewModel() { }  public ObservableCollection<Passenger> Passengers { get; set; }  public string Gender { get; set; }  public Visibility PassengersListVisibility { get; set; }  public Visibility SuccessPaymentVisibility { get; set; }  public PaymentViewModel(Flight flight, int count)  {  PassengersListVisibility = Visibility.Visible;  SuccessPaymentVisibility = Visibility.Collapsed;  this.flight = flight;  this.count = count;  Passengers = new ObservableCollection<Passenger>();  for(int i = 0; i < count; i++)  {  Passenger passenger = new Passenger();  Passengers.Add(passenger);  }  }  public ICommand SubmitPayment  {  get  {  return new DelegateCommand(() =>  {  bool isOk = true;  using (var db = new AviaBookingDbContext())  {  foreach (var pas in Passengers)  {  pas.FlightId = flight.Id;  char selectedGender = Convert.ToChar(pas.Gender);  if (!string.IsNullOrEmpty(pas.FirstName) &&  !string.IsNullOrEmpty(pas.LastName) &&  (pas.Gender == 'М' || pas.Gender == 'Ж') &&  pas.BirthDate != null &&  pas.FlightId > 0)  {  db.Passengers.Add(pas);  }  else  {  isOk = false;  }  }  if (isOk == false)  MessageBox.Show("Заполните все поля");  else  {  var payment = new Payment();  MainWindow mainWindow = (MainWindow)Application.Current.MainWindow;  Header mainFrame = (Header)mainWindow.FindName("Header");  var accountViewModel = mainFrame.DataContext as HeaderViewModel;  if (accountViewModel != null)  {  var account = accountViewModel.account.DataContext as AccountViewModel;  payment.ClientId = account.ClientID;  payment.FlightId = flight.Id;  payment.PaymentDate = DateTime.Today;  }  db.Payments.Add(payment);  db.SaveChanges();  PassengersListVisibility = Visibility.Collapsed;  SuccessPaymentVisibility = Visibility.Visible;  }  }  });  }  }  } |

Листинг – Код класса ReviewsViewModel

|  |
| --- |
| public class ReviewsViewModel : ViewModelBase  {  public ObservableCollection<Airline> Airlines { get; set; }  public ObservableCollection<Client> Clients { get; set; }  public ObservableCollection<Review> Reviews{ get; set; }  public Airline SelectedAirline { get; set; }  public ObservableCollection<ReviewModel> ReviewModels { get; set; }  public Review review { get; set; }  public Visibility ReviewMenuVisibility { get; set; }  public int ClientID { get; set; }  public ReviewsViewModel()  {  using (var db = new AviaBookingDbContext())  {  var airlines = db.Airlines.ToList();  Airlines = new ObservableCollection<Airline>(airlines);  }  ReviewMenuVisibility = Visibility.Collapsed;  review = new Review();  }    public ICommand SendReview  {  get  {  return new DelegateCommand(() =>  {  using (var db = new AviaBookingDbContext())  {  if(SelectedAirline == null)  {  MessageBox.Show("Выберите авиакомпанию");  }  if(SelectedAirline != null)  {  if (int.TryParse(review.Score.ToString(), out int score) && review.Text != null  && review.Score >= 1 && review.Score <= 5)  {  review.AirlineId = SelectedAirline.Id;  MessageBox.Show(review.Score + " " + review.Text);  review.ClientId = ClientID;  db.Reviews.Add(review);  db.SaveChanges();  var reviews = db.Reviews.Where(r => r.AirlineId == SelectedAirline.Id).ToList();  var clientIds = reviews.Select(r => r.ClientId).Distinct().ToList();  var clients = db.Clients.Where(c => clientIds.Contains(c.Id)).ToList();  ReviewModels = new ObservableCollection<ReviewModel>(  reviews.Join(clients, r => r.ClientId, c => c.Id, (r, c) => new ReviewModel  {  ClientName = c.Name,  ClientSurname = c.Surname,  ReviewText = r.Text,  Score = r.Score,  }).ToList());  }  else if(review.Score < 1 || review.Score > 5)  {  MessageBox.Show("Введите оценку от 1 до 5");  }  else  {  MessageBox.Show("Заполните отзыв");  }  }    }  });  }  }  public ICommand ShowReviewMenu  {  get  {  return new DelegateCommand(() =>  {  MainWindow mainWindow = (MainWindow)Application.Current.MainWindow;  Header mainFrame = (Header)mainWindow.FindName("Header");  var accountViewModel = mainFrame.DataContext as HeaderViewModel;  if (accountViewModel != null)  {  var account = accountViewModel.account.DataContext as AccountViewModel;  if(account.ClientID > 0)  {  if (ReviewMenuVisibility == Visibility.Collapsed)  {  ReviewMenuVisibility = Visibility.Visible;  }  else  {  ReviewMenuVisibility = Visibility.Collapsed;  }  ClientID = account.ClientID;  }  else  {  MessageBox.Show("Чтобы оставить отзыв войдите в аккаунт");  }  }    });  }  }  public ICommand ShowReviews  {  get  {  return new DelegateCommand(() =>  {  using (var db = new AviaBookingDbContext())  {  if(SelectedAirline != null)  {  var reviews = db.Reviews.Where(r => r.AirlineId == SelectedAirline.Id).ToList();  var clientIds = reviews.Select(r => r.ClientId).Distinct().ToList();  var clients = db.Clients.Where(c => clientIds.Contains(c.Id)).ToList();  ReviewModels = new ObservableCollection<ReviewModel>(  reviews.Join(clients, r => r.ClientId, c => c.Id, (r, c) => new ReviewModel  {  ClientName = c.Name,  ClientSurname = c.Surname,  ReviewText = r.Text,  Score = r.Score,  }).ToList());  } else  {  MessageBox.Show("Выберите авиакомпанию");  }  }  });  }  }  } |

Листинг – Код класса AccountViewModel

|  |
| --- |
| public class AccountViewModel : ViewModelBase  {  public Visibility LoginPageVisibility { get; set; }  public Visibility AccountPageVisibility { get; set; }  public Visibility RegButtonVisibility { get; set; }  public Visibility LogButtonVisibility { get; set; }  private ICommand \_logButton;  private ICommand \_regButton;  public string LogNickname { get; set; }  public string LogName { get; set; }  public string LogSurname { get; set; }  public string LogEmail { get; set; }  public string LogPassword { get; set; }  public string LogSecuredPassword { get; set; }  public string TempNickname { get; set; }  public string TempName { get; set; }  public string TempSurname { get; set; }  public string TempEmail { get; set; }  public string TempPassword { get; set; }  public Visibility CurrentTicketsVisibility { get; set; }  public Visibility PersonalDataVisibility { get; set; }  public Visibility ChangePersonalDataVisibility { get; set; }  public Visibility HistoryVisibility { get; set; }  public Visibility TicketsNotFoundVisibility { get; set; }  public bool isLogin { get; set; }  public bool isAdmin { get; set; }  public Visibility AdminPanelVisibility { get; set; }  public Visibility AdminButtonVisibility { get; set; }  public Visibility FlightsMenuVisibility { get; set; }  public Visibility ClientsMenuVisibility { get; set; }  public Visibility PassengersMenuVisibility { get; set; }  public ObservableCollection<Flight> Flights { get; set; }  public Flight SelectedDeleteFlight { get; set; }  public int AirlineId { get; set; }  public string Departure { get; set; }  public string Destination { get; set; }  public DateTime DepartureDate { get; set; }  public DateTime ArrivalDate { get; set; }  public int AvailableSeats { get; set; }  public ComboBoxItem Class { get; set; }  public Client Client { get; set; }  public ObservableCollection<Client> Clients { get; set; }  public int FlightNumber { get; set; }  public ObservableCollection<Passenger> Passengers { get; set; }  public int ClientID { get; set; }  public AccountViewModel()  {  LoginPageVisibility = Visibility.Visible;  AccountPageVisibility = Visibility.Collapsed;  RegButtonVisibility = Visibility.Collapsed;  LogButtonVisibility = Visibility.Visible;  CurrentTicketsVisibility = Visibility.Collapsed;  PersonalDataVisibility = Visibility.Collapsed;  ChangePersonalDataVisibility = Visibility.Collapsed;  HistoryVisibility = Visibility.Collapsed;  TicketsNotFoundVisibility = Visibility.Collapsed;  AdminPanelVisibility = Visibility.Collapsed;  AdminButtonVisibility = Visibility.Collapsed;  isLogin = true;  isAdmin = false;  }  public ICommand AdminPanelButton  {  get  {  return new DelegateCommand(() =>  {  CurrentTicketsVisibility = Visibility.Collapsed;  PersonalDataVisibility = Visibility.Collapsed;  ChangePersonalDataVisibility = Visibility.Collapsed;  HistoryVisibility = Visibility.Collapsed;  TicketsNotFoundVisibility = Visibility.Collapsed;  if (AdminPanelVisibility == Visibility.Collapsed)  {  AdminPanelVisibility = Visibility.Visible;  }  else  {  AdminPanelVisibility = Visibility.Collapsed;  }  FlightsMenuVisibility = Visibility.Collapsed;  ClientsMenuVisibility = Visibility.Collapsed;  PassengersMenuVisibility = Visibility.Collapsed;  });  }  }  public ICommand AddFlightButton  {  get  {  return new DelegateCommand(() =>  {  if (AirlineId == null)  {  MessageBox.Show("Please enter Airline Id.");  }  if (string.IsNullOrEmpty(Departure))  {  MessageBox.Show("Please enter Departure.");  }  if (string.IsNullOrEmpty(Destination))  {  MessageBox.Show("Please enter Destination.");  }  if(DepartureDate <= DateTime.Today)  {  MessageBox.Show("DepartureDate cannot be past");  }  if (DepartureDate > ArrivalDate)  {  MessageBox.Show("Departure Date cannot be later than Arrival Date.");  }  if (ArrivalDate.Subtract(DepartureDate).TotalDays > 1)  {  MessageBox.Show("The difference between Departure Date and Arrival Date cannot be more than 1 day.");  }  if (AvailableSeats <= 0)  {  MessageBox.Show("Please enter a valid number of Available Seats.");  }  else  {  Flight flight = new Flight();  flight.AirlineId = AirlineId;  flight.Departure = Departure;  flight.Destination = Destination;  flight.DepartureDate = DepartureDate;  flight.ArrivalDate = ArrivalDate;  flight.AvailableSeats = AvailableSeats;  ComboBoxItem selectedItem = Class as ComboBoxItem;  if (selectedItem != null)  {  flight.Class = selectedItem.Content.ToString();  }  MessageBox.Show(flight.Class);  using (var db = new AviaBookingDbContext())  {  db.Flights.Add(flight);  db.SaveChanges();  }  Flights.Add(flight);  }  });  }  }  public ICommand FlightsMenuButton  {  get  {  return new DelegateCommand(() =>  {  FlightsMenuVisibility = Visibility.Visible;  ClientsMenuVisibility = Visibility.Collapsed;  PassengersMenuVisibility = Visibility.Collapsed;  using (var db = new AviaBookingDbContext())  {  var flights = db.Flights.ToList();  Flights = new ObservableCollection<Flight>(flights);  }  });  }  }  public ICommand ClientsMenuButton  {  get  {  return new DelegateCommand(() =>  {  FlightsMenuVisibility = Visibility.Collapsed;  ClientsMenuVisibility = Visibility.Visible;  PassengersMenuVisibility = Visibility.Collapsed;  using (var db = new AviaBookingDbContext())  {  var clients = db.Clients.ToList();  Clients = new ObservableCollection<Client>(clients);  }  });  }  }  public ICommand PassengersMenuButton  {  get  {  return new DelegateCommand(() =>  {  FlightsMenuVisibility = Visibility.Collapsed;  ClientsMenuVisibility = Visibility.Collapsed;  PassengersMenuVisibility = Visibility.Visible;  });  }  }  public ICommand DeleteFlightCommand  {  get  {  return new DelegateCommand(() =>  {  using (var db = new AviaBookingDbContext())  {  db.Flights.Remove(SelectedDeleteFlight);  db.SaveChanges();  Flights.Remove(SelectedDeleteFlight);  }  });  }  }  public ICommand \_changeClientRole { get; set; }  public ICommand ChangeClientRole => \_changeClientRole ??= new DelegateCommand<object>(parameter =>  {  if(Client != null)  {  using (var db = new AviaBookingDbContext())  {  var client = db.Clients.FirstOrDefault(c => c.Nickname == Client.Nickname);  if (client.Role == "user")  {  client.Role = "admin";  db.SaveChanges();  var clients = db.Clients.ToList();  Clients = new ObservableCollection<Client>(clients);  }  }      }  });  public ICommand FindPassengers  {  get  {  return new DelegateCommand(() =>  {  if(FlightNumber != null)  {  using (var db = new AviaBookingDbContext())  {  var passengers = db.Passengers  .Where(p => p.FlightId == FlightNumber)  .ToList();  Passengers = new ObservableCollection<Passenger>(passengers);  }  }  else  {  MessageBox.Show("Введите номер рейса");  }  });  }  }  public ICommand FillDataBase  {  get  {  return new DelegateCommand(() =>  {  using (var db = new AviaBookingDbContext())  {  for (int i = 0; i < 10; i++)  {  var faker = new Faker<Airline>()  .RuleFor(x => x.Name, f => f.Company.CompanyName())  .RuleFor(x => x.Rating, f => Math.Round(f.Random.Double(1, 5), 2));  Airline airline = faker.Generate();  db.Airlines.Add(airline);  db.SaveChanges();  }  }  using (var db = new AviaBookingDbContext())  {  for (int i = 0; i < 25; i++)  {  var faker = new Faker<Client>()  .RuleFor(x => x.Nickname, f => f.Internet.UserName())  .RuleFor(x => x.Name, f => f.Name.FirstName())  .RuleFor(x => x.Surname, f => f.Name.LastName())  .RuleFor(x => x.Email, f => f.Internet.Email())  .RuleFor(x => x.Password, f => f.Internet.Password());  Client client = faker.Generate();  client.Role = "user";  db.Clients.Add(client);  db.SaveChanges();  }  }  using (var db = new AviaBookingDbContext())  {  var faker = new Faker<Flight>()  .RuleFor(x => x.AirlineId, f => f.Random.Number(1, 10))  .RuleFor(x => x.Departure, f => f.Address.Country())  .RuleFor(x => x.Destination, f => f.Address.Country())  .RuleFor(x => x.DepartureDate, f => f.Date.Future())  .RuleFor(x => x.ArrivalDate, (f, x) => f.Date.Between(x.DepartureDate, x.DepartureDate.AddDays(1)))  .RuleFor(x => x.AvailableSeats, f => f.Random.Number(50, 200))  .RuleFor(x => x.Class, f => f.PickRandom(new[] { "Economy", "Comfort", "Business", "First Class" }));  for (int i = 0; i < 400; i++)  {  Flight flight = faker.Generate();  db.Flights.Add(flight);  }  db.SaveChanges();  }  using (var db = new AviaBookingDbContext())  {  var faker = new Faker<Passenger>()  .RuleFor(x => x.FirstName, f => f.Person.FirstName)  .RuleFor(x => x.LastName, f => f.Person.LastName)  .RuleFor(x => x.BirthDate, f => f.Person.DateOfBirth)  .RuleFor(x => x.FlightId, f => f.Random.Number(1, 400));  for (int i = 0; i < 1000; i++)  {  Passenger passenger = faker.Generate();  Random rand = new Random();  int random = rand.Next(1, 100);  if (random <= 50)  {  passenger.Gender = 'М';  }  else  {  passenger.Gender = 'Ж';  }  db.Passengers.Add(passenger);  }  db.SaveChanges();  }  using (var db = new AviaBookingDbContext())  {  var faker = new Faker<Payment>()  .RuleFor(x => x.ClientId, f => f.Random.Number(1, 25))  .RuleFor(x => x.FlightId, f => f.Random.Number(1, 400))  .RuleFor(x => x.PaymentDate, f => f.Date.Past());  for (int i = 0; i < 100; i++)  {  Payment payment = faker.Generate();  db.Payments.Add(payment);  }  db.SaveChanges();  }  using (var db = new AviaBookingDbContext())  {  var faker = new Faker<Review>()  .RuleFor(x => x.AirlineId, f => f.Random.Number(1, 10))  .RuleFor(x => x.ClientId, f => f.Random.Number(1, 25))  .RuleFor(x => x.Score, f => f.Random.Number(1, 5));  for (int i = 0; i < 100; i++)  {  Review review = faker.Generate();  review.Text = "";  db.Reviews.Add(review);  }  db.SaveChanges();  }  MessageBox.Show("База данных заполнена!");  });  }  }  public ICommand \_returnFlightTickets { get; set; }  public ICommand ReturnFlightTickets => \_returnFlightTickets ??= new DelegateCommand<object>(parameter =>  {  var box = (ListView)parameter;  Flight selectedFlight = box.SelectedItem as Flight;  using (var db = new AviaBookingDbContext())  {  var paymentId = db.Payments.FirstOrDefault(p => p.FlightId == selectedFlight.Id);  if (paymentId != null)  {  db.Payments.Remove(paymentId);  db.SaveChanges();  }  var client = db.Clients.FirstOrDefault(f => f.Nickname == LogNickname && f.Password == LogPassword);  if (client != null)  {  // Получаем все записи из таблицы Payments, где клиент совпадает с искомым клиентом.  var payments = db.Payments.Where(p => p.ClientId == client.Id).ToList();  // Получаем все рейсы, которые соответствуют найденным записям в таблице Payments.  var flights = new List<Flight>();  foreach (var payment in payments)  {  var flight = db.Flights.FirstOrDefault(f => f.Id == payment.FlightId && f.DepartureDate > DateTime.Today);  if (flight != null)  {  flights.Add(flight);  }  }  if (flights.Count > 0)  Flights = new ObservableCollection<Flight>(flights);  else  {  TicketsNotFoundVisibility = Visibility.Visible;  CurrentTicketsVisibility = Visibility.Collapsed;  }  }  }  });  #region Смена Вход/Регистрация  public ICommand ChooseRegistrationButton  {  get  {  return new DelegateCommand(() =>  {  RegButtonVisibility = Visibility.Visible;  LogButtonVisibility = Visibility.Collapsed;  isLogin = false;  });  }  }  public ICommand ChooseLoginButton  {  get  {  return new DelegateCommand(() =>  {  RegButtonVisibility = Visibility.Collapsed;  LogButtonVisibility = Visibility.Visible;  isLogin = true;  });  }  }  #endregion  #region Кнопки сменты видимости меню в личном кабинете  public ICommand CurrentTicketsButton  {  get  {  return new DelegateCommand(() =>  {  CurrentTicketsVisibility = Visibility.Visible;  PersonalDataVisibility = Visibility.Collapsed;  ChangePersonalDataVisibility = Visibility.Collapsed;  HistoryVisibility = Visibility.Collapsed;  TicketsNotFoundVisibility = Visibility.Collapsed;  using (var db = new AviaBookingDbContext())  {  var client = db.Clients.FirstOrDefault(f => f.Nickname == LogNickname && f.Password == LogPassword);  if (client != null)  {  // Получаем все записи из таблицы Payments, где клиент совпадает с искомым клиентом.  var payments = db.Payments.Where(p => p.ClientId == client.Id).ToList();  // Получаем все рейсы, которые соответствуют найденным записям в таблице Payments.  var flights = new List<Flight>();  foreach (var payment in payments)  {  var flight = db.Flights.FirstOrDefault(f => f.Id == payment.FlightId && f.DepartureDate > DateTime.Today);  if (flight != null)  {  flights.Add(flight);  }  }  if (flights.Count > 0)  Flights = new ObservableCollection<Flight>(flights);  else  {  TicketsNotFoundVisibility = Visibility.Visible;  CurrentTicketsVisibility = Visibility.Collapsed;  }  }  }  });  }  }  public ICommand PersonalDataButton  {  get  {  return new DelegateCommand(() =>  {  CurrentTicketsVisibility = Visibility.Collapsed;  PersonalDataVisibility = Visibility.Visible;  ChangePersonalDataVisibility = Visibility.Collapsed;  HistoryVisibility = Visibility.Collapsed;  TicketsNotFoundVisibility = Visibility.Collapsed;  });  }  }  public ICommand ChangePersonalDataButton  {  get  {  return new DelegateCommand(() =>  {  CurrentTicketsVisibility = Visibility.Collapsed;  PersonalDataVisibility = Visibility.Collapsed;  ChangePersonalDataVisibility = Visibility.Visible;  HistoryVisibility = Visibility.Collapsed;  TicketsNotFoundVisibility = Visibility.Collapsed;  });  }  }  public ICommand HistoryButton  {  get  {  return new DelegateCommand(() =>  {  CurrentTicketsVisibility = Visibility.Collapsed;  PersonalDataVisibility = Visibility.Collapsed;  ChangePersonalDataVisibility = Visibility.Collapsed;  HistoryVisibility = Visibility.Visible;  TicketsNotFoundVisibility = Visibility.Collapsed;  using (var db = new AviaBookingDbContext())  {  var client = db.Clients.FirstOrDefault(f => f.Nickname == LogNickname && f.Password == LogPassword);  if (client != null)  {    // Получаем все записи из таблицы Payments, где клиент совпадает с искомым клиентом.  var payments = db.Payments.Where(p => p.ClientId == client.Id).ToList();  // Получаем все рейсы, которые соответствуют найденным записям в таблице Payments.  var flights = new List<Flight>();  foreach (var payment in payments)  {  var flight = db.Flights.FirstOrDefault(f => f.Id == payment.FlightId && f.DepartureDate <= DateTime.Today);  if (flight != null)  {  flights.Add(flight);  }  }  if (flights.Count > 0)  Flights = new ObservableCollection<Flight>(flights);  else  {  TicketsNotFoundVisibility = Visibility.Visible;  HistoryVisibility = Visibility.Collapsed;  }  }  }  });  }  }  public ICommand ShowPasswordButton  {  get  {  return new DelegateCommand(() =>  {  if(LogSecuredPassword.Contains('\*'))  {  LogSecuredPassword = LogPassword;  } else  {  LogSecuredPassword = "";  for(int i = 0; i < LogPassword.Length; i++)  {  LogSecuredPassword += '\*';  }  }  });  }  }  public ICommand ChangeAccountDataButton  {  get  {  return new DelegateCommand(() =>  {  using (var db = new AviaBookingDbContext())  {  var client = db.Clients.FirstOrDefault(f => f.Nickname == LogNickname);  if (client != null)  {  if (string.IsNullOrEmpty(LogNickname) || string.IsNullOrEmpty(LogName) ||  string.IsNullOrEmpty(LogSurname) || string.IsNullOrEmpty(LogEmail) ||  string.IsNullOrEmpty(LogPassword))  {  MessageBox.Show("Please fill in all the fields.");  }  if (!string.IsNullOrEmpty(LogEmail))  {  if (!LogEmail.Contains("@") || !LogEmail.Contains("."))  {  MessageBox.Show("Please enter a valid email address.");  }  else  {  client.Nickname = TempNickname;  client.Name = TempName;  client.Surname = TempSurname;  client.Email = TempEmail;  client.Password = TempPassword;  db.SaveChanges();  MessageBox.Show("Данные изменены");  LogNickname = TempNickname;  LogName = TempName;  LogSurname = TempSurname;  LogEmail = TempEmail;  LogPassword = TempPassword;  }  }  }  }  });  }  }  #endregion  #region Функции входа/выхода  // Функция входа  public ICommand LogButton => \_logButton ??= new DelegateCommand<object>(parameter =>  {  PasswordBox box = (PasswordBox)parameter;  LogPassword = box.Password;  if (string.IsNullOrEmpty(LogNickname) || string.IsNullOrEmpty(LogPassword))  {  MessageBox.Show("Please fill in all the fields.");  } else  {  using (var db = new AviaBookingDbContext())  {  var client = db.Clients.FirstOrDefault(f => f.Nickname == LogNickname && f.Password == box.Password);  if(client!= null)  {  LoginPageVisibility = Visibility.Collapsed;  AccountPageVisibility = Visibility.Visible;  CurrentTicketsVisibility = Visibility.Visible;  TicketsNotFoundVisibility = Visibility.Collapsed;  LogNickname = client.Nickname;  LogName = client.Name;  LogEmail = client.Email;  LogSurname = client.Surname;  ClientID = client.Id;  TempNickname = LogNickname;  TempName = LogName;  TempSurname = LogSurname;  TempEmail = LogEmail;  TempPassword = LogPassword;  for(int i = 0; i < LogPassword.Length; i++)  {  LogSecuredPassword += '\*';  }  // Получаем все записи из таблицы Payments, где клиент совпадает с искомым клиентом.  var payments = db.Payments.Where(p => p.ClientId == client.Id).ToList();  // Получаем все рейсы, которые соответствуют найденным записям в таблице Payments.  var flights = new List<Flight>();  foreach (var payment in payments)  {  var flight = db.Flights.FirstOrDefault(f => f.Id == payment.FlightId && f.DepartureDate > DateTime.Today);  if (flight != null)  {  flights.Add(flight);  }  }  if (flights.Count > 0)  Flights = new ObservableCollection<Flight>(flights);  else  {  TicketsNotFoundVisibility = Visibility.Visible;  CurrentTicketsVisibility = Visibility.Collapsed;  }  if(client.Role == "admin")  {  isAdmin = true;  AdminButtonVisibility = Visibility.Visible;  }  }  else  {  MessageBox.Show("Не найдено совпадений");  }  }  }    }, parameter => parameter is PasswordBox box);  public ICommand RegButton => \_regButton ??= new DelegateCommand<object>(parameter =>  {  PasswordBox box = (PasswordBox)parameter;  string LogPassword = GetSecureHash(box.SecurePassword);  if (string.IsNullOrEmpty(LogNickname) || string.IsNullOrEmpty(LogName) ||  string.IsNullOrEmpty(LogSurname) || string.IsNullOrEmpty(LogEmail) ||  string.IsNullOrEmpty(LogPassword))  {  MessageBox.Show("Please fill in all the fields.");  }  if (!string.IsNullOrEmpty(LogEmail))  {  if (!LogEmail.Contains("@") || !LogEmail.Contains("."))  {  MessageBox.Show("Please enter a valid email address.");  }  else  {  using (var db = new AviaBookingDbContext())  {  var checkClient = db.Clients.FirstOrDefault(p => p.Nickname == LogNickname || p.Email == LogEmail);  if(checkClient == null)  {  var client = new Client();  client.Nickname = LogNickname;  client.Name = LogName;  client.Surname= LogSurname;  client.Email= LogEmail;  client.Password= box.Password;  client.Role = "user";  db.Clients.Add(client);  db.SaveChanges();  RegButtonVisibility = Visibility.Collapsed;  LogButtonVisibility = Visibility.Visible;  }  else  {  MessageBox.Show("Пользователь с таким именем или почтой уже существует");  }  }  }  }  }, parameter => parameter is PasswordBox box);  // Функция выхода  public ICommand \_exitButton { get; set; }  public ICommand ExitButton => \_exitButton ??= new DelegateCommand<object>(parameter =>  {  PasswordBox box = (PasswordBox)parameter;  box.Password = "";  CurrentTicketsVisibility = Visibility.Collapsed;  PersonalDataVisibility = Visibility.Collapsed;  ChangePersonalDataVisibility = Visibility.Collapsed;  HistoryVisibility = Visibility.Collapsed;  LoginPageVisibility = Visibility.Visible;  AccountPageVisibility = Visibility.Collapsed;  LogNickname = "";  LogName = "";  LogSurname = "";  LogEmail = "";  LogPassword = "";  LogSecuredPassword = "";  }, parameter => parameter is PasswordBox box);  #endregion  #region Хэширование пароля  private string GetSecureHash(SecureString secureString)  {  SHA256 sha256 = SHA256.Create();  Span<byte> hashBytes = stackalloc byte[sha256.HashSize >> 3];  IntPtr ptr = Marshal.SecureStringToBSTR(secureString);  unsafe  {  ReadOnlySpan<byte> source = new ReadOnlySpan<byte>((void\*)ptr, secureString.Length \* sizeof(char));  sha256.TryComputeHash(source, hashBytes, out \_);  }  Marshal.ZeroFreeBSTR(ptr);  return HashToString(hashBytes);  }  private string HashToString(ReadOnlySpan<byte> bytes)  {  StringBuilder sb = new StringBuilder(bytes.Length \* 2);  for (int i = 0; i < bytes.Length; i++)  sb.Append(bytes[i].ToString("x2"));  return sb.ToString();  }  #endregion  } |