*Федеральное государственное бюджетное образовательное учреждение высшего образования*

*«Владимирский государственный университет*

*имени Александра Григорьевича и Николая Григорьевича Столетовых»*

*Кафедра информационных систем и программной инженерии*

***ПОЯСНИТЕЛЬНАЯ ЗАПИСКА***

***к курсовому проекту по дисциплине   
"Распределённые программные системы"***

***на тему***

*Проектирование и разработка программной системы   
информационной системы «Автотранспортное предприятие»*

*Выполнил: ст. гр. ИСТ-121*

*Опарин Д. С.*

*Принял: Тимофеев А.А.*

*Владимир, 2024*

**Аннотация**

Курсовой проект посвящён проектированию и разработке программной системы - «Автотранспортное предприятие», которая в условиях современности поможет пользователям взаимодействовать с автотранспортным предприятием.

При работе над системой выполнены следующее этапы: анализ предметной области, выполненный на основе сравнения аналогов, проектирование программной систем, создание макетов системы и реализация системы. Программная часть системы реализована на основе платформы Java EE c использование фреймворка Spring.

Разработана на основе СУБД MySQL.

Курсовой проект представлен на 67 страницах, рисунков – 23, таблиц – 0, использованных источников – 10, приложений – 2.

**ANNOTATION**

The course project is devoted to the design and development of a software system - “Motor Transport Enterprise”, which in modern conditions will help users interact with a motor transport enterprise.

When working on the system, the following stages were completed: analysis of the subject area, performed on the basis of comparison of analogues, design of software systems, creation of system layouts and implementation of the system. The software part of the system is implemented on the basis of the Java EE platform using the Spring framework.

Developed based on the MySQL DBMS.

The course project is presented on 67 pages, figures - 23, tables - 0, sources used - 10, applications - 2.

**СОДЕРЖАНИЕ**

[1 ВВЕДЕНИЕ 5](#_Toc167737514)

[2 ПОСТАНОВКА ЗАДАЧИ 6](#_Toc167737515)

[2.1 Описание предметной области 6](#_Toc167737516)

[2.2 Основные понятия предметной области 6](#_Toc167737517)

[2.3 Функциональные требования к системе 7](#_Toc167737518)

[2.4 Нефункциональные требования к системе 7](#_Toc167737519)

[2.5 Сравнительный анализ аналогов 8](#_Toc167737520)

[3 АНАЛИЗ ЗАДАЧИ И ФУНКЦИОНАЛЬНАЯ ДЕКОМПОЗИЦИЯ СИСТЕМЫ 11](#_Toc167737521)

[3.1 Анализ бизнес-процесса «Покупка билета» 11](#_Toc167737522)

[3.2 Сценарий взаимодействия пользователя с системой 12](#_Toc167737523)

[3.3 Аутентификация пользователя в системе 14](#_Toc167737524)

[3.4 Регистрация пользователя в системе 14](#_Toc167737525)

[3.5 Просмотр каталога маршрутов 15](#_Toc167737526)

[3.6 Поиск и фильтрация каталога маршрутов 16](#_Toc167737527)

[3.7 Покупка билета 16](#_Toc167737528)

[3.8 Просмотр списка билетов 17](#_Toc167737529)

[3.9 Завершение работы с системой 17](#_Toc167737530)

[4 СТРУКТУРНАЯ ОРГАНИЗАЦИЯ СИСТЕМЫ 17](#_Toc167737531)

[4.1 Общие принципы организации системы 17](#_Toc167737532)

[4.2 Организация доступа к данным 18](#_Toc167737533)

[4.3 Организация бизнес-логики 18](#_Toc167737534)

[4.4 Организация веб-интерфейса 19](#_Toc167737535)

[4.5 Взаимодействие компонентов системы для выполнения бизнес-функции приложения 23](#_Toc167737536)

[4.6 Организация обработки ошибок 24](#_Toc167737537)

[4.7 Организация управления доступом 25](#_Toc167737538)

[5 РЕАЛИЗАЦИЯ КОМПОНЕНТОВ СИСТЕМЫ 30](#_Toc167737539)

[5.1 EJB-компоненты/компоненты бизнес-логики 30](#_Toc167737540)

[5.1.1 EJB-модуль 1 30](#_Toc167737541)

[5.1.2 EJB-модуль 2 38](#_Toc167737542)

[5.2 Веб-компоненты 41](#_Toc167737543)

[5.2.1 Управление данными и логикой представления 41](#_Toc167737544)

[5.2.2 Форма входа в систему 41](#_Toc167737545)

[5.2.3 JSP-страница register.jsp 41](#_Toc167737546)

[5.2.4 Конфигурирование JSF-приложения 42](#_Toc167737547)

[5.3 Физическая структура баз данных 44](#_Toc167737548)

[5.3.1 База данных busbus 44](#_Toc167737549)

[5.4 Обеспечение целостности данных 45](#_Toc167737550)

[5.4.1 Ограничения целостности 45](#_Toc167737551)

[5.4.2 Организация распределенных транзакций 45](#_Toc167737552)

[6 ИНСТРУКЦИЯ ПО УСТАНОВКЕ ПРИЛОЖЕНИЯ 48](#_Toc167737553)

[7 ОРГАНИЗАЦИЯ УПРАВЛЕНИЯ ПРОЕКТОМ 51](#_Toc167737554)

[7.1 Общие принципы взаимодействия в команде (работа выполнялась в одиночку) 51](#_Toc167737555)

[7.2 Распределение ролей в команде/зоны ответственности 51](#_Toc167737556)

[7.3 Календарный план работ 52](#_Toc167737557)

[8 НАГРУЗОЧНОЕ ТЕСТИРОВАНИЕ 54](#_Toc167737558)

[9 ЗАКЛЮЧЕНИЕ 60](#_Toc167737559)

[10 СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 61](#_Toc167737560)

[ПРИЛОЖЕНИЕ А Исходный код программной системы 62](#_Toc167737561)

[ПРИЛОЖЕНИЕ Б Структура таблиц БД 62](#_Toc167737562)

# 1 ВВЕДЕНИЕ

В современном мире транспорт играет важную роль. Для удобства и эффективности управления работой транспорта необходима информационная система, которая позволит автоматизировать деятельность автотранспортного предприятия.

В рамках курсового проекта будет разработана программная система для информационной системы автотранспортного предприятия.

Результатом работы будет функционирующая программная система, которая сможет облегчить процессы управления автотранспортным предприятием.

# 2 ПОСТАНОВКА ЗАДАЧИ

## 2.1 Описание предметной области

Автотранспортное предприятие *—* организация, осуществляющая перевозки автомобильным транспортом. Реализуемое приложение, позволит эффективно управлять всеми аспектами организации и проведения перевозок, взаимодействовать с клиентами, а также обеспечивать оперативный доступ к необходимым данным. Предоставляется возможность выбора различных направлений маршрутов. Веб приложение будет реализовано как многопользовательское приложение. Будут разработаны профили для диспетчера, администратора, покупателя. Каждый профиль будет иметь определенный функционал. Профили создаются для взаимодействия пользователей с сервисом. Главными функциями будут являться: предоставление информации о маршрутах и возможность покупки билета на рейс. Диспетчер производит подтверждение возврата билета на маршрут.

## 2.2 Основные понятия предметной области

Концептуальная диаграмма классов представлена на рисунке 1.

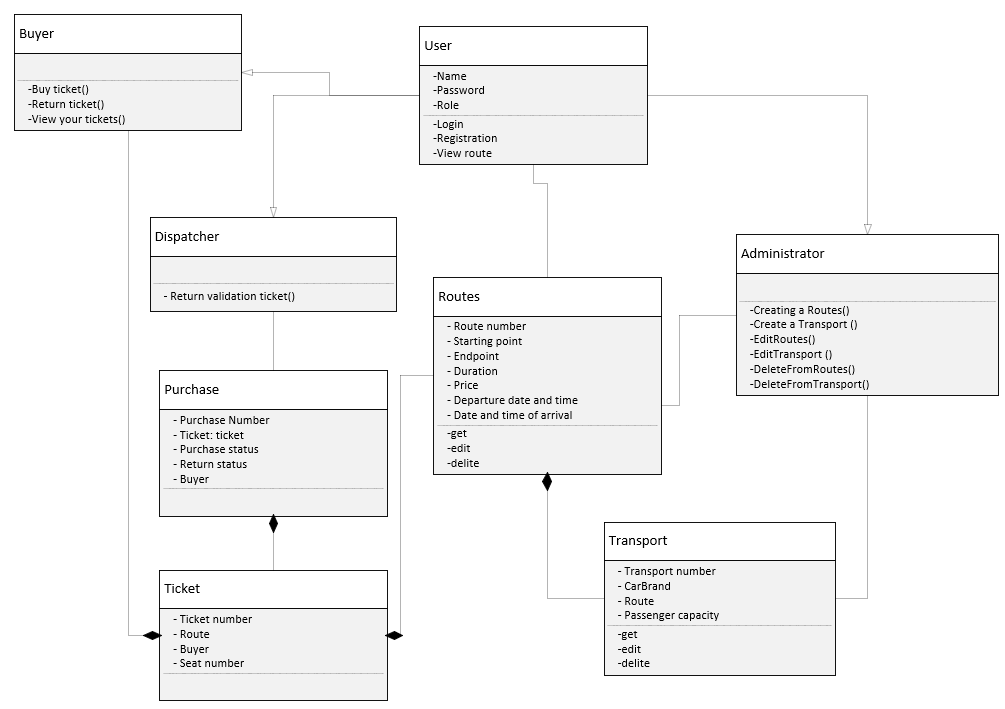
****

Рисунок 1. Концептуальная диаграмма классов

## 2.3 Функциональные требования к системе

- Возможность отображения маршрутов с указанием расписания для каждого маршрута.

- Поиск маршрутов по различным критериям, таким как пункт отправления, пункт назначения, дата отправления.

- Возможность выбора места в автобусе и отображение свободных мест, занятые места не будут отображаться.

- Предоставление информации о маршрутах.

- Отправка электронного билета в личный кабинет пользователя в раздел мои билеты.

- Возможность возврата билетов согласно правилам компании.

- Ведение учетной записи пользователя с возможностью просматривать историю покупок билетов.

## 2.4 Нефункциональные требования к системе

- Удобство использования: Система должна быть интуитивно понятной и простой в использовании

- Эффективность: Система должна быть оптимизирована для обеспечения быстрой и эффективной работы.

- Гибкость: Система должна быть легко настраиваемой и адаптируемой к изменяющимся потребностям бизнеса.

- Совместимость: Система должна быть совместима с различными устройствами и браузерами.

- Доступность: Система должна быть доступна для пользователей 24/7.

- Поддержка: Система должна иметь надежную систему поддержки для помощи пользователям.

## 2.5 Сравнительный анализ аналогов

Сравнительный анализ сервисов по онлайн покупке билетов на автобус.

Сервисы: Туту.ру, На-автобус, Автовокзалы.

Критерии сравнения: Пользовательский интерфейс, Ассортимент направлений и маршрутов, Удобство покупки, Стоимость билетов, Дополнительные функции.

**Пользовательский интерфейс:**

Туту.ру: Удобный и интуитивно понятный интерфейс, возможность поиска по карте.

На-автобус: более простой интерфейс, но менее информативный.

Автовокзалы: Информация о направлениях представлена в виде списка, поиск по карте отсутствует.

**Ассортимент направлений и маршрутов:**

Туту.ру: Самый полный ассортимент направлений и маршрутов, включая межрегиональные и международные.

На-автобус: Небольшой ассортимент внутрирегиональных маршрутов.

Автовокзалы: Информация о маршрутах, отправляющихся с конкретных автовокзалов.

**Удобство покупки:**

Туту.ру: Возможность выбора мест на схеме салона, онлайн-оплата, быстрый процесс бронирования.

На-автобус: Оплата в офисах партнеров, нет онлайн-выбора мест.

Автовокзалы: Покупка билетов за наличные в кассах автовокзалов.

**Стоимость билетов:**

Туту.ру: Цены сопоставимы с ценами в кассах автовокзалов.

На-автобус: может быть небольшая комиссия за обслуживание.

Автовокзалы: Самые низкие цены, так как отсутствует комиссия за обслуживание.

**Дополнительные функции:**

Туту.ру: Программа лояльности, возврат билетов с комиссией.

На-автобус: Информация о скидках для разных категорий пассажиров.

Автовокзалы: Информация о расписании, наличии мест и стоимости билетов.

**Выводы:**

Для путешественников, которым важны широкий выбор маршрутов, удобный интерфейс и онлайн-оплата: Туту.ру.

Для покупки билетов на внутрирегиональные маршруты по минимальной цене: На-автобус.

Для тех, кто предпочитает покупать билеты в офисах или у партнеров: Автовокзалы.

# 3 АНАЛИЗ ЗАДАЧИ И ФУНКЦИОНАЛЬНАЯ ДЕКОМПОЗИЦИЯ СИСТЕМЫ

## 3.1 Анализ бизнес-процесса «Покупка билета»

**Описание как происходит покупка билета**

1. Выбор направления: Пользователь выбирает нужное ему направление, указывая пункт отправления и пункт назначения. В приложении будет предоставлена возможность ввести свои собственные направления.

2. Выбор даты: Пользователь выбирает нужную дату (не более чем на две недели вперед). Будут предложены все варианты рейсов на выбранную дату.

3. Выбор времени: Пользователь выбирает из списка маршрутов доступное время.

4. Регистрация и вход: Приложение может потребовать от пользователя регистрацию или вход в учетную запись. В этом случае пользователь создает аккаунт и вводит свои личные данные, такие как имя и пароль.

5. Выбор места: Система проверяет какие места заняты, а какие нет, и исходя из этого из этого генерирует список доступных мест. Пользователь выбирает желаемое место в автобусе. В приложении будет выпадающий список с указанием места.

6. Покупка: Пользователь нажимает кнопку купить билет и перенаправляется в личный кабинет.

7. Получение билета: после успешной покупки пользователь получает свой билет в электронном виде в разделе мои билеты личного кабинета.

Диаграмма бизнес-процесса представлена на рисунке 2.

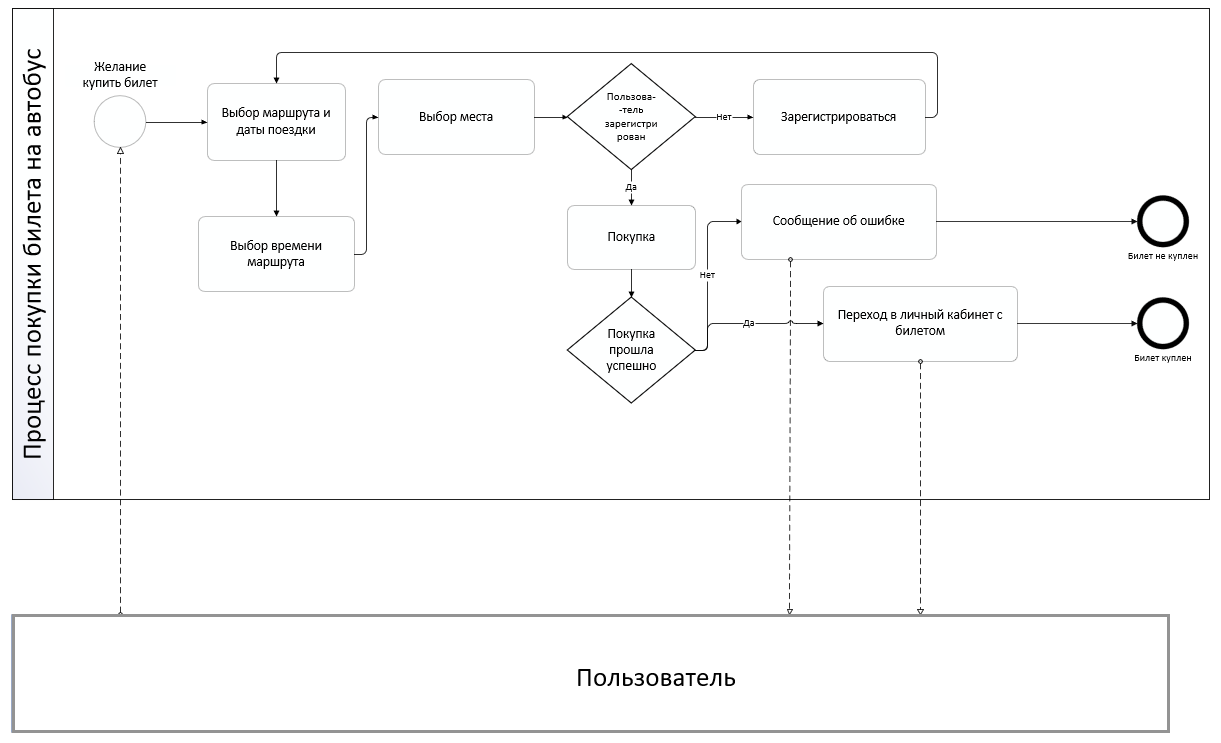


Рисунок 2. Диаграмма бизнес-процесса

## 3.2 Сценарий взаимодействия пользователя с системой

Диаграмма прецедентов представлена на рисунке 3.

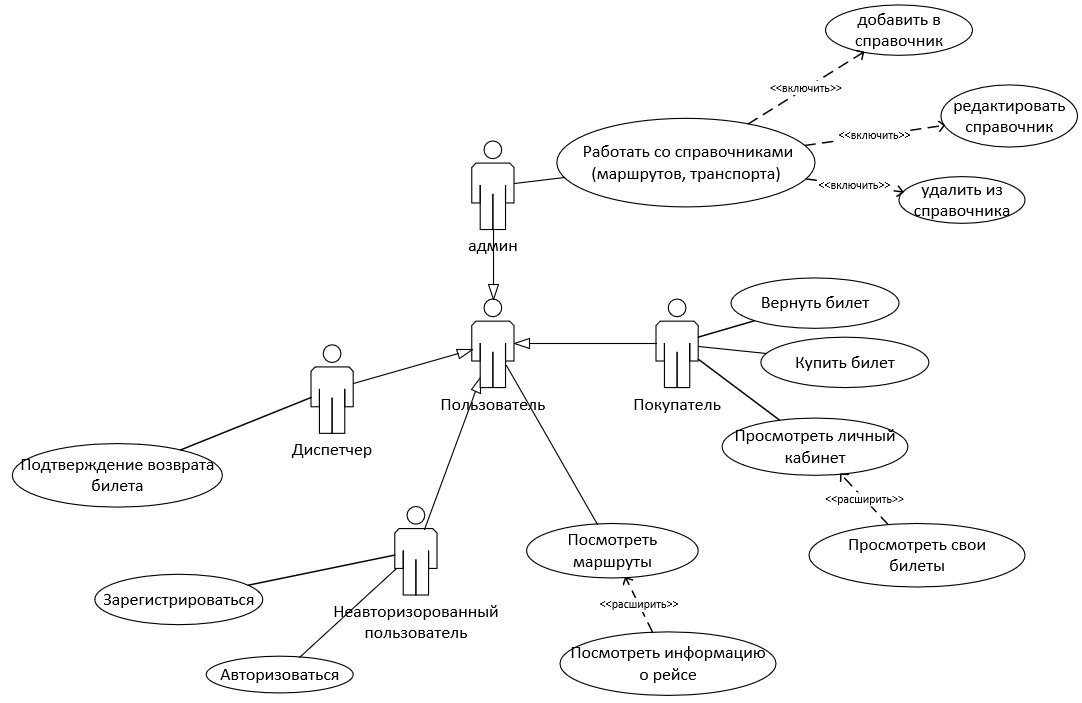


Рисунок 3. Диаграмма прецедентов

Диаграмма состояний представлена на рисунке 4.

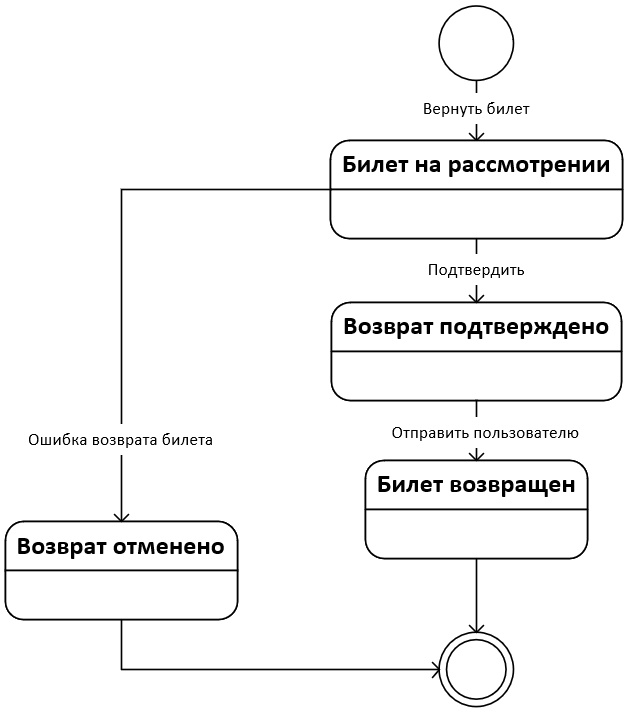


Рисунок 4. Диаграмма состояний

## 3.3 Аутентификация пользователя в системе

**Прецедент «Авторизоваться»**:

**Предусловие:** Неавторизованный пользователь находится на странице авторизации

**Действующее лицо:** Неавторизованный пользователь

**Основной поток:** Авторизация

1. Пользователь вводит свои учетные данные, такие как логин пользователя и пароль.
2. Система проверяет данные на корректность.
3. Система проверяет данные на соответствие записям в базе данных.
4. Если данные верны, система авторизует пользователя, создавая для него сеанс авторизации.
5. Система дает пользователю доступ к системе, который может использовать ее функциональность.

**Альтернативный поток 1:** Данные введены некорректно

Шаг 2 – если данные введены некорректно, система выводит сообщение об этом: «Данные введены некорректно, повторите»

**Альтернативный поток 2:** Данные не соответствуют данным из базы данных (неправильно введены данные)

Шаг 3 – если данные не соответствую данным из базы данных, система выводит сообщение об этом: «Данные введены неправильно, повторите»

**Постусловие:** Пользователя перенаправляет на главную страницу

## 3.4 Регистрация пользователя в системе

**Прецедент «Зарегистрироваться»**

**Предусловие**: Незарегистрированный пользователь находится на странице регистрации.

**Действующее лицо:** Незарегистрированный пользователь

**Основной поток**

1. Пользователь вводит свои регистрационные данные, такие как имя пользователя и пароль.

2. Система проверяет корректность данных (например, длина и наличие специальных символов в пароле).

3. Система проверяет, что имя пользователя не используется другим пользователем.

4. Если данные верны, система создает нового пользователя и сохраняет его в базе данных.

5. Система отправляет пользователю подтверждение регистрации (например, на электронную почту).

**Альтернативный поток 1:** Данные введены некорректно

Шаг 2 – если данные введены некорректно, система выводит сообщение об ошибке и возвращает пользователя на страницу регистрации

**Альтернативный поток 2:** Имя пользователя уже используется

Шаг 3 – если имя пользователя уже используется, система выводит сообщение об ошибке и возвращает пользователя на страницу регистрации.

**Постусловие**

Пользователь зарегистрирован и перенаправляется на страницу входа.

## 3.5 Просмотр каталога маршрутов

**Прецедент «Просмотреть каталог маршрутов»**

**Предусловие**: Пользователь находится на странице просмотра каталога продукции.

**Действующее лицо:** Пользователь

**Основной поток:**

1. Пользователь просматривает список продукции, отображаемый на странице.

2. Пользователь может фильтровать и сортировать продукцию по различным критериям.

3. Пользователь может просматривать подробную информацию о каждом продукте, нажимая на соответствующую кнопку.

4. Пользователь может купить билет на конкретный маршрут.

**Альтернативный поток:** если пользователь не авторизован, ему будет предложено авторизоваться или зарегистрироваться.

**Постусловие:** Пользователь остается на странице просмотра каталога продукции.

## 3.6 Поиск и фильтрация каталога маршрутов

**Предусловие**: Пользователь находится на странице каталога продукции.

**Действующее лицо:** Пользователь

**Основной поток:**

1. Пользователь вводит параметры поиска и фильтрации в форму.

2. Форма отправляет запрос на сервер.

3. Система выполняет поиск и фильтрацию продукции по указанным параметрам.

4. Система возвращает пользователю отфильтрованный список продукции.

5. Пользователь просматривает отфильтрованный список продукции.

**Альтернативный поток:** если пользователь не ввел корректные параметры поиска или фильтрации, система вернет сообщение об ошибке.

**Постусловие:** Пользователь остается на странице каталога продукции с отфильтрованным списком маршрутов.

## 3.7 Покупка билета

**Предусловие:** Пользователь просматривает страницу сведений о маршруте и хочет купить билет.

**Действующее лицо:** Пользователь

**Основной поток:**

1. Пользователь нажимает кнопку "Купить" на странице сведений о маршруте.

2. Пользователь перенаправляется на страницу покупки билета.

3. Пользователь выбирает место и подтверждает покупку.

4. Система проверяет наличие выбранного места и списывает средства с карты пользователя.

5. Система отправляет пользователю электронный билет.

**Альтернативный поток:** если выбранное место недоступно, система выведет сообщение об ошибке и предложит выбрать другое место.

Если пользователь не авторизован, система предложит ему авторизоваться или зарегистрироваться.

**Постусловие:** Пользователь получает электронный билет и становится пассажиром рейса.

## 3.8 Просмотр списка билетов

**Предусловие:** Пользователь находится в своем личном кабинете.

**Действующее лицо:** Пользователь

**Основной поток:**

1. Пользователь переходит в раздел "Мои билеты" в личном кабинете.

2. Система отображает список всех билетов пользователя.

3. Пользователь может просмотреть детали каждого билета, включая номер билета, стоимость и статус заказа.

**Альтернативный поток:** если у пользователя нет заказов, система ничего не выведет.

**Постусловие:** Пользователь остается в личном кабинете и может продолжать просматривать и управлять своими билетами.

## 3.9 Завершение работы с системой

**Предусловие:** Пользователь авторизован в системе.

**Действующее лицо:** Пользователь

**Основной поток:**

1. Пользователь нажимает на кнопку "Выход" в интерфейсе системы.

2. Система завершает сеанс пользователя и удаляет его авторизированные данные.

3. Пользователь перенаправляется на страницу входа.

**Альтернативный поток:** если пользователь не авторизован, система перенаправит его на страницу входа.

**Постусловие:** Пользователь выходит из системы и становится неавторизованным.

# 4 СТРУКТУРНАЯ ОРГАНИЗАЦИЯ СИСТЕМЫ

## 4.1 Общие принципы организации системы

Общая схема организации системы представлена на рисунке 5.

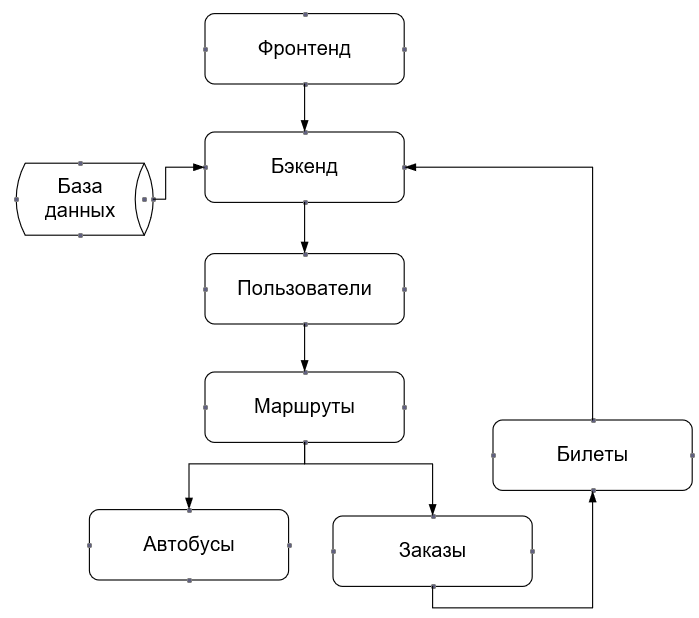
****

Рисунок 5. Общая схема организации системы

## 4.2 Организация доступа к данным

Диаграмма классов уровня доступа к данным представлена на рисунке 6.

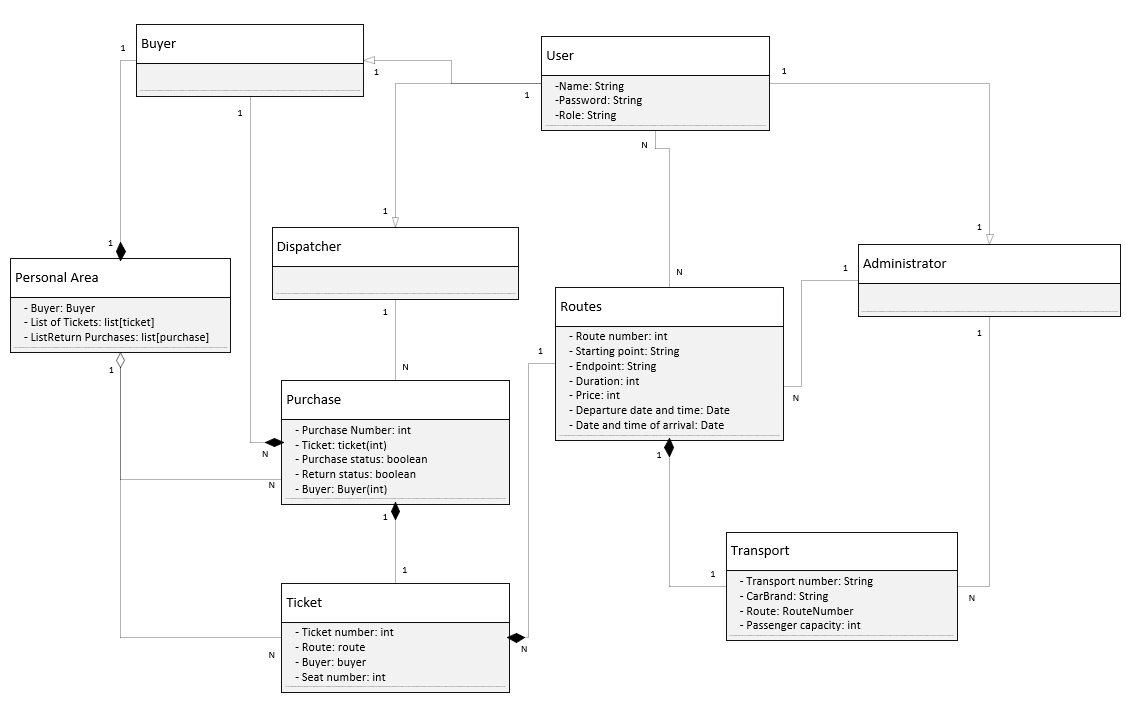
****

Рисунок 6. Диаграмма классов уровня доступа к данным

## 4.3 Организация бизнес-логики

Диаграмма классов уровня бизнес-логики представлена на рисунке 7.

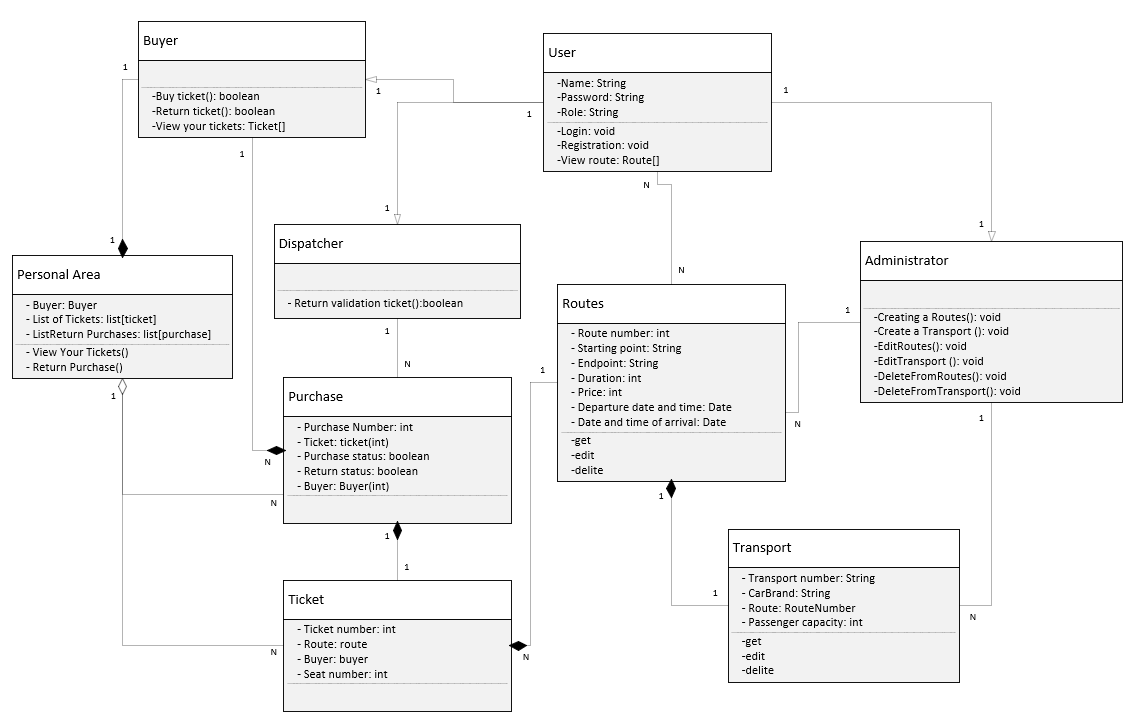
****

Рисунок 7. Диаграмма классов уровня бизнес-логики

## 4.4 Организация веб-интерфейса

Схема переходов между страницами представлена на рисунке 7.

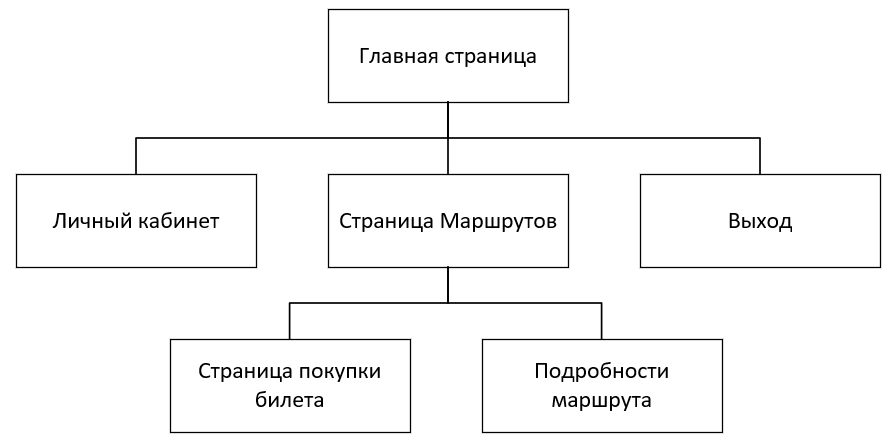
****

Рисунок 8. Схема переходов между страницами

Описание страниц:

Главная страница:

Главная страница сайта содержит следующую информацию и элементы:

1. Навигационное меню: здесь есть ссылки на разделы "Главная", "Личный кабинет", "Выйти".

2. Форма поиска маршрута: здесь пользователь может выбрать пункт отправления и пункт прибытия, а также указать дату выезда.

3. Популярные направления: указаны некоторые популярные маршруты с указанием минимальной стоимостью билетов.

4. Кнопка ссылка для подбора маршрута по параметрам с учетом сортировки и фильтрации.

Главная страница представлена на рисунке 8.

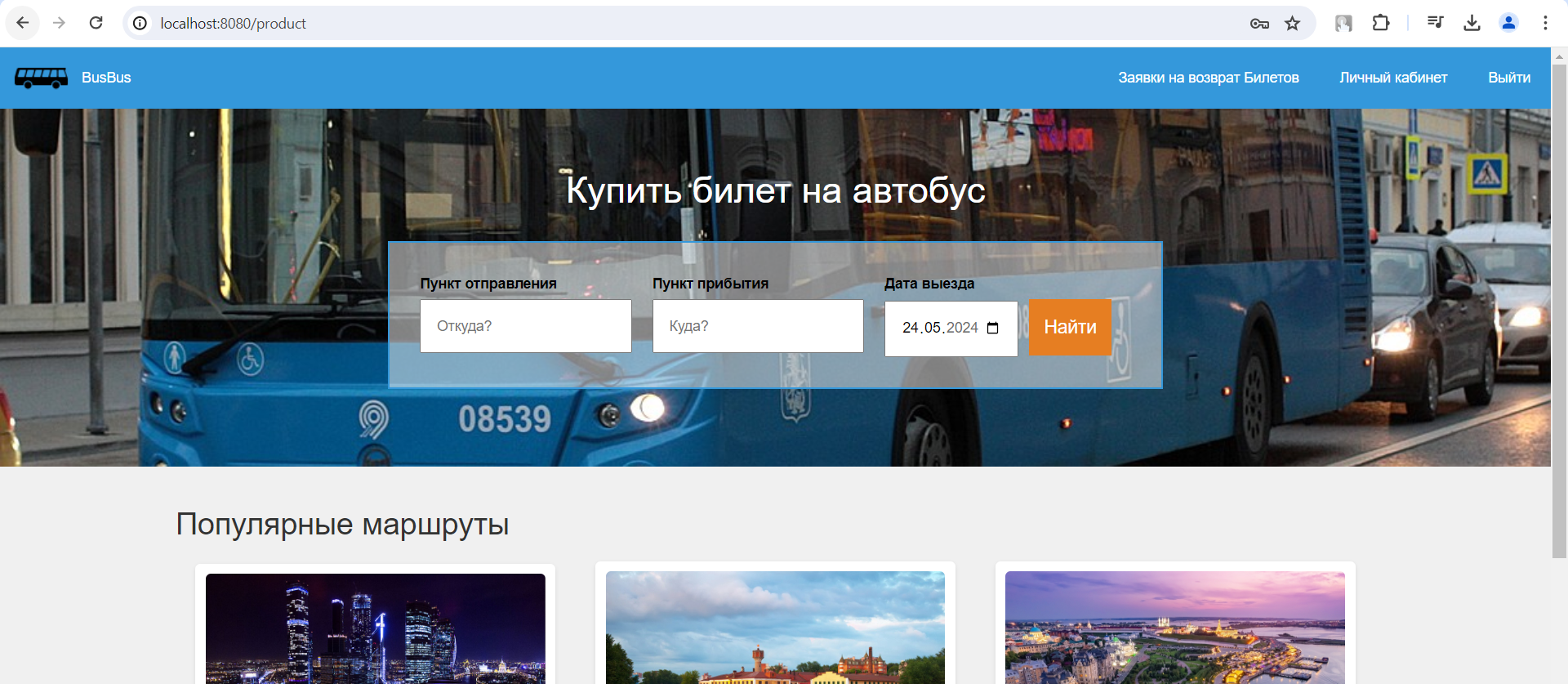


Рисунок 9. Главная страница

Страница расписания маршрута:

На данной странице сайта вы можете ознакомиться с информацией о расписание автобусов на различные направления.

Вверху страницы есть навигационное меню с ссылками на разделы "Главная", "Личный кабинет" и "Выйти".

Затем на странице предлагается выбрать пункт отправления и пункт прибытия, а также указать дату выезда, так же предоставляется возможность отсортировать маршруты по цене или по длительности.

Под этой формой находится список маршрутов. Указывается стоимость билетов, времени в пути, километраж, дата выезда и время отправления и прибытия для каждого рейса, и есть ссылка для получения подробной информации о маршруте.

Страница расписания маршрута представлена на рисунке 9.

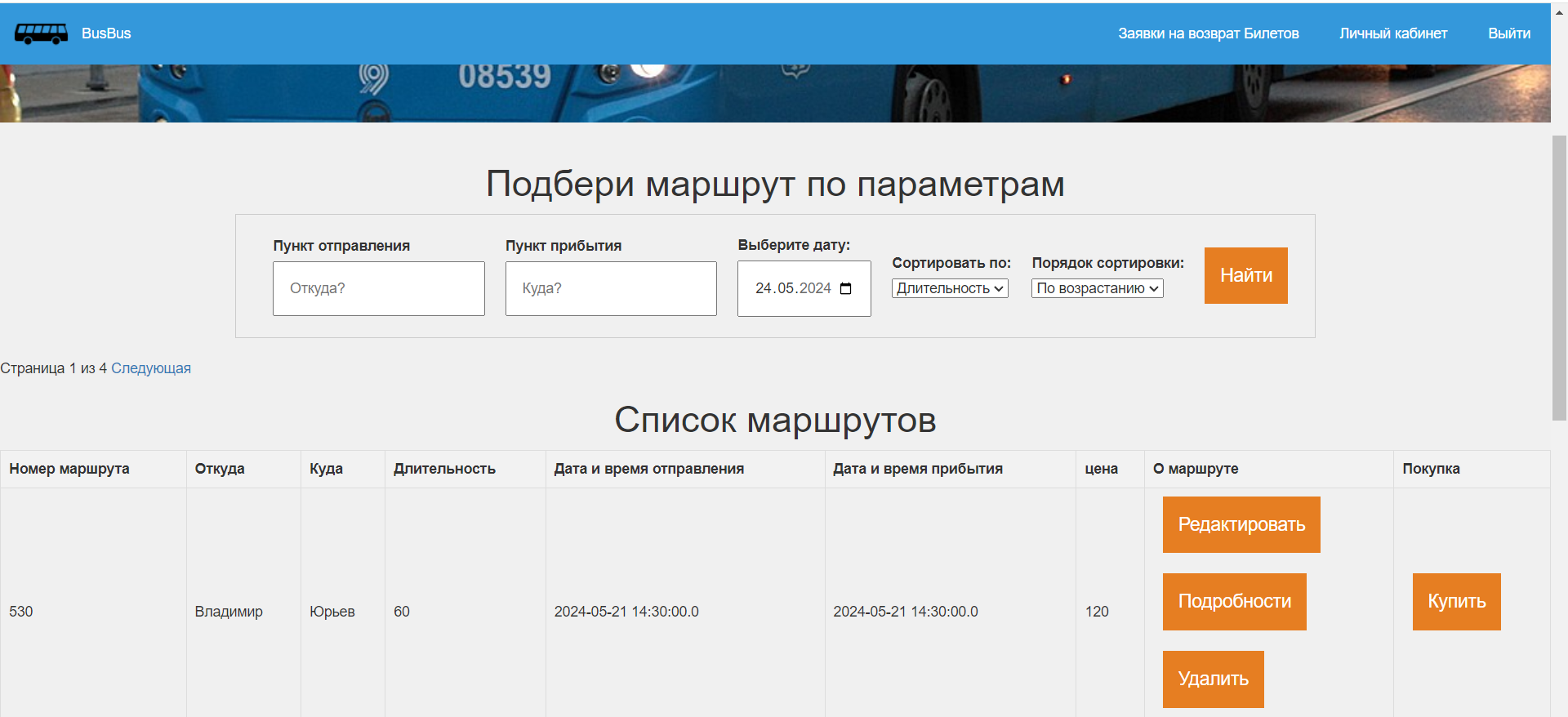


Рисунок 10. Страница расписания маршрута

Страница оформления заказа:

Данная страница открывается после нажатия кнопки купить на предыдущей странице, данные с выбранного маршрута автоматически подставляются в форму покупки билета, пользователю остается только выбрать желаемое место (купленные места будут не доступны для продажи, то есть пользователю будут отображаться только доступные места) и нажать купить.

После покупки, пользователя перенаправляет на страницу личного кабинета, где и будут отображаться его билеты.

Страница оформления заказа представлена на рисунке 10.

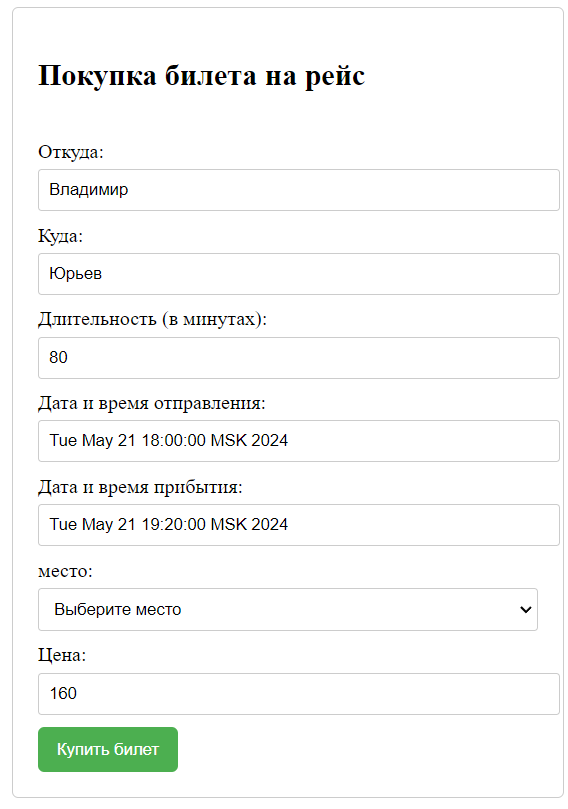


Рисунок 11. Страница оформления заказа

Страница личного кабинета пользователя:

На данной странице пользователь может видеть свои данные, а также может просматривать свои купленные билеты. У пользователя есть возможность вернуть билет, если диспетчер одобряет возврат билета, то билет будет возвращен.

Страница личного кабинета пользователя представлена на рисунке 11.

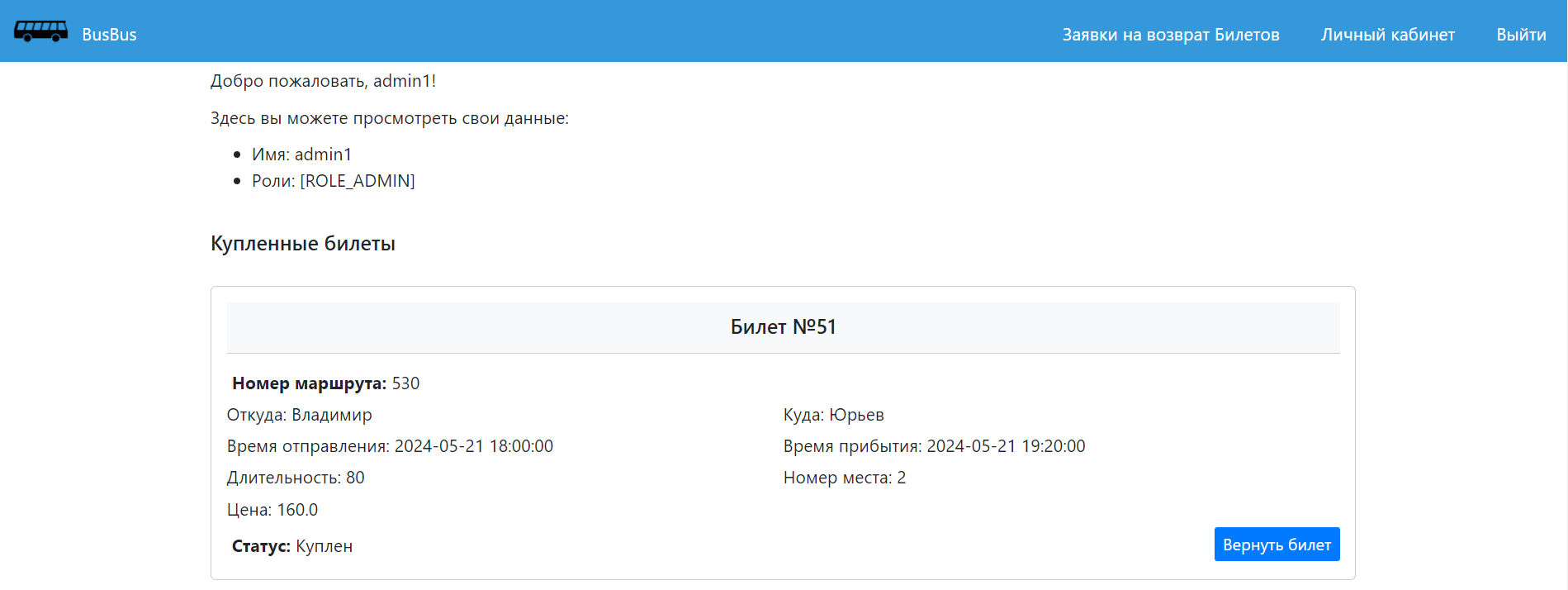


Рисунок 12. Личный кабинет

## 4.5 Взаимодействие компонентов системы для выполнения бизнес-функции приложения

Диаграмма последовательностей представлена на рисунке 12:

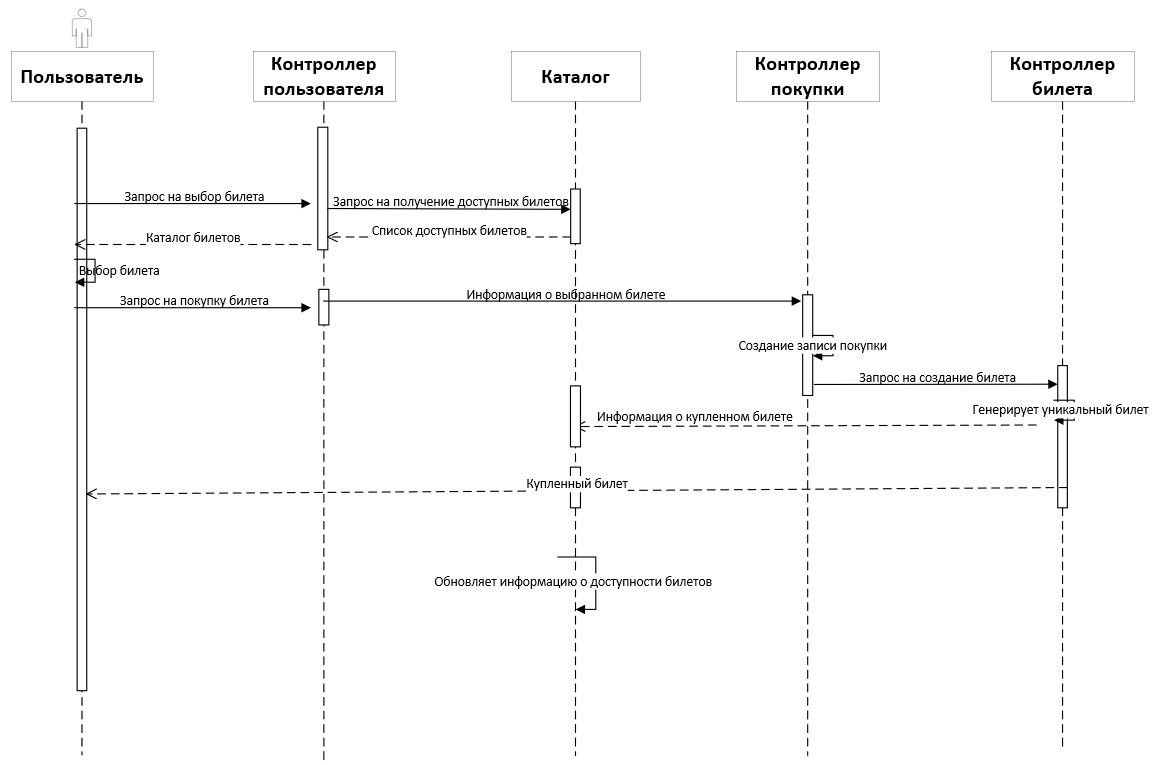


Рисунок 13. Диаграмма последовательностей

## 4.6 Организация обработки ошибок

Для обработки ошибок и проверки вводимых данных реализованы следующие механизмы:

1. Проверка вводимых данных на стороне сервера: Метод registerNewUser () проверяет соответствие пароля заданным требованиям (содержит как минимум одну букву и одну цифру, не менее 4 символов). В случае несоответствия возвращается сообщение об ошибке.

@PostMapping("")  
public ResponseEntity<String> registerNewUser(@RequestParam String name, @RequestParam String password) {  
 MyUser newUser = new MyUser();  
 newUser.setName(name);  
 newUser.setPassword(passwordEncoder.encode(password));  
 newUser.setRoles("ROLE\_USER");  
 userRepository.save(newUser);  
 return ResponseEntity.ok("User registered successfully");  
}

2. Отображение сообщений об ошибках на стороне клиента: В представлении формы реализована проверка вводимых данных с помощью атрибута `pattern` в элементах `<input>`. При несоответствии введенных данных заданным требованиям пользователю отображается сообщение об ошибке.

<label for="name">Имя пользователя:</label>  
<input type="text" id="name" name="name" required pattern="[A-Za-zА-Яа-я]+" placeholder="Только буквы">

<br><br>  
<label for="password">Пароль:</label>  
<input type="password" id="password" name="password" required pattern="(?=.\*[A-Za-zА-Яа-я])(?=.\*\d).{4,}" title="Пароль должен содержать как минимум одну букву и одну цифру, не менее 4 символов" placeholder="Буквы и цирры">

3. Использование `@ModelAttribute` для передачи данных из формы: В методе registerNewUser () данные из формы извлекаются с помощью аннотации `@ModelAttribute`. Это позволяет проводить проверку данных на стороне сервера и отображать сообщения об ошибках на стороне клиента.

Данная реализация обеспечивает контроль вводимых данных и предоставляет пользователю понятные сообщения об ошибках, что повышает удобство использования и надежность приложения.

## 4.7 Организация управления доступом

**Настройка Spring Security:** настраивается механизм авторизации Spring Security с помощью класса SecurityConfiguration.

Определяются правила доступа к различным URL-адресам:

Доступ к странице регистрации (/users) и странице добавления пользователя (/addUser) разрешен всем пользователям (permitAll()).

Доступ к странице продукта (/product) разрешен только авторизованным пользователям (authenticated()).

Настраивается форма входа в систему (formLogin).

Настраивается поставщик аутентификации DaoAuthenticationProvider, который использует пользовательскую службу безопасности и механизм кодирования паролей.

@Configuration  
public class SecurityConfig {  
 @Bean  
 public UserDetailsService userDetailsService() {  
 return new MyUserDetailsService();  
 }  
 @Bean  
 public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {  
 return http.csrf(AbstractHttpConfigurer::disable)  
 .authorizeRequests(authorize -> authorize  
 .antMatchers("/users").permitAll()  
 .anyRequest().authenticated()  
 )  
 .formLogin(form -> form  
 .permitAll()  
 .defaultSuccessUrl("/product")  
 )  
 .authenticationProvider(authenticationProvider())  
 .authorizeHttpRequests(auth -> auth.mvcMatchers("/users", "/addUser", "product").permitAll())  
 .build();  
 }  
 @Bean  
 public AuthenticationProvider authenticationProvider() {  
 DaoAuthenticationProvider provider = new DaoAuthenticationProvider();  
 provider.setUserDetailsService(userDetailsService());  
 provider.setPasswordEncoder(passwordEncoder());  
 return provider;  
 }  
 @Bean  
 public PasswordEncoder passwordEncoder() {  
 return new BCryptPasswordEncoder();  
 }  
}

**Создание пользовательской службы безопасности:** Класс MyUserDetailsService реализует интерфейс UserDetailsService и отвечает за загрузку информации о пользователе по его имени. Метод loadUserByUsername() находит пользователя в базе данных по имени и возвращает объект MyUserDetails.

@Service  
public class MyUserDetailsService implements UserDetailsService {  
 @Autowired  
 private UserRepository repository;  
 @Override  
 public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {  
 MyUser user = repository.findByName(username)  
 .orElseThrow(() -> new UsernameNotFoundException(username + " not found"));  
 return new MyUserDetails(user);  
 }  
}

**Обеспечение авторизации на основе ролей:** Пользовательская реализация MyUserDetails реализует интерфейс UserDetails и содержит метод isAdmin(), который проверяет, имеет ли пользователь роль администратора. Метод getAuthorities() возвращает список полномочий пользователя, которые определяются в сущности пользователя MyUser.

public class MyUserDetails implements UserDetails {

private MyUser user;

public MyUserDetails(MyUser user) {

this.user = user;

}

@Override

public Collection<? extends GrantedAuthority> getAuthorities() {

return user.getAuthorities();

}

public boolean isAdmin() {

return getAuthorities().stream()

.anyMatch(a -> a.getAuthority().equals("ROLE\_ADMIN"));

}

}

**Дополнительные сведения об организации управления доступом:** Использование аннотаций @PreAuthorize и @PostAuthorize: Эти аннотации можно использовать для проверки авторизации на уровне методов контроллера. Например:

@PreAuthorize("hasRole('ROLE\_ADMIN')")

public String adminMethod() {

// Доступ к этому методу имеют только пользователи с ролью администратора

}

Вместе эти компоненты обеспечивают управление доступом на основе ролей в приложении.

# 5 РЕАЛИЗАЦИЯ КОМПОНЕНТОВ СИСТЕМЫ

## 5.1 EJB-компоненты/компоненты бизнес-логики

EJB-компоненты являются основой архитектуры Java EE и представляют собой модульные компоненты, которые реализуют бизнес-логику приложения. В данном случае EJB-компоненты используются для управления данными, связанными с билетами, и предоставления абстракции для доступа к данным и выполнения бизнес-логики.

## 5.1.1 EJB-модуль 1

TicketDaoImpl: Класс TicketDaoImpl реализует интерфейс TicketDao и предоставляет методы для управления данными билетов в базе данных. Он включает методы для добавления, удаления, выборки и обновления билетов, а также проверки существования билета по его номеру.

@Repository  
public class TicketDaoImpl implements TicketDao {  
 @Override  
 @Transactional(propagation = Propagation.REQUIRED)  
 public void addTicket(Ticket ticket) {  
 String query = "INSERT INTO ticket (ticket\_number, route\_id, buyer\_id, seat\_number, departure\_date, price) " +  
 "VALUES (?, ?, ?, ?, ?, ?)";  
 try (Connection connection = DbConnection.getConnection();  
 PreparedStatement statement = connection.prepareStatement(query)) {  
 statement.setInt(1, ticket.getTicketNumber());  
 statement.setInt(2, ticket.getRoute().getId());  
 statement.setInt(3, ticket.getBuyer().getId());  
 statement.setInt(4, ticket.getSeatNumber());  
 statement.setTimestamp(5, new java.sql.Timestamp(ticket.getDepartureDateTime().getTime()));  
 statement.setDouble(6, ticket.getPrice());  
  
 statement.executeUpdate();  
 } catch (SQLException e) {  
 e.printStackTrace();  
 }  
 }  
 @Override  
 @Transactional(propagation = Propagation.REQUIRES\_NEW)  
 public boolean deleteTicket(int ticketNumber) {  
 String query = "DELETE FROM ticket WHERE ticket\_number = ?";  
 try (Connection connection = DbConnection.getConnection();  
 PreparedStatement statement = connection.prepareStatement(query)) {  
 statement.setInt(1, ticketNumber);  
  
 int rowsDeleted = statement.executeUpdate();  
 return rowsDeleted > 0;  
 } catch (SQLException e) {  
 e.printStackTrace();  
 return false;  
 }  
 }  
 @Override  
 public Ticket[] viewTickets(int userId) {  
 return new Ticket[0];  
 }  
 @Transactional(propagation = Propagation.REQUIRES\_NEW)  
 public List<Integer> getAvailableSeatNumbers(int routeId) {  
 List<Integer> availableSeats = new ArrayList<>();  
 try (Connection conn = DbConnection.getConnection()) {  
 String sqlQuery = "SELECT number " +  
 "FROM ( " +  
 " SELECT 1 AS number " +  
 " UNION SELECT 2 " +  
 " UNION SELECT 3 " +  
 " UNION SELECT 4 " +  
 " UNION SELECT 5 " +  
 " UNION SELECT 6 " +  
 " UNION SELECT 7 " +  
 " UNION SELECT 8 " +  
 " UNION SELECT 9 " +  
 " UNION SELECT 10 " +  
 " UNION SELECT 11 " +  
 " UNION SELECT 12 " +  
 " UNION SELECT 13 " +  
 " UNION SELECT 14 " +  
 " UNION SELECT 15 " +  
 " UNION SELECT 16 " +  
 ") AS all\_numbers " +  
 "WHERE all\_numbers.number NOT IN (SELECT seat\_number FROM ticket WHERE route\_id = ?) " +  
 "OR all\_numbers.number IN " +  
 " (SELECT seat\_number " +  
 " FROM ticket" +  
 " JOIN ticketSost ON ticket.ticket\_number = ticketSost.idTicket " +  
 " WHERE ticket.route\_id = ? AND ticketSost.sost = 1)";  
 PreparedStatement statement = conn.prepareStatement(sqlQuery);  
 statement.setInt(1, routeId);  
 statement.setInt(2, routeId);  
 ResultSet resultSet = statement.executeQuery();  
 System.out.println(resultSet);  
 while (resultSet.next()) {  
 int seatNumber = resultSet.getInt("number");  
 availableSeats.add(seatNumber);  
 }  
 } catch (SQLException e) {  
 e.printStackTrace();  
 }  
 return availableSeats;  
 }  
 public int max\_id() {  
 int maxId = 0;  
  
 try (Connection conn = DbConnection.getConnection()) {  
 String sqlQuery = "SELECT MAX(id) AS max\_id FROM ticket;";  
 PreparedStatement statement = conn.prepareStatement(sqlQuery);  
 ResultSet resultSet = statement.executeQuery();  
 if (resultSet.next()) {  
 maxId = resultSet.getInt("max\_id");  
 }  
 } catch (SQLException e) {  
 e.printStackTrace();  
 }  
 return maxId;  
 }  
 public List<String[]> getTicketsByUserId(int userId) {  
 List<String[]> ticketList = new ArrayList<>();  
 try (Connection conn = DbConnection.getConnection()) {  
 String sqlQuery = "SELECT "  
 + "t.ticket\_number, "  
 + "r.route\_number, "  
 + "r.start\_point, "  
 + "r.end\_point, "  
 + "r.duration, "  
 + "r.departure\_datetime, "  
 + "r.arrival\_datetime, "  
 + "t.seat\_number, "  
 + "t.price "  
 + "FROM users AS u "  
 + "JOIN ticket AS t "  
 + "ON u.id = t.buyer\_id "  
 + "JOIN route AS r "  
 + "ON t.route\_id = r.id "  
 + "WHERE "  
 + "u.id = ?;"  
 PreparedStatement statement = conn.prepareStatement(sqlQuery);  
 statement.setInt(1, userId);  
 ResultSet resultSet = statement.executeQuery();  
 while (resultSet.next()) {  
 String ticket\_number = resultSet.getString("ticket\_number");  
 String route\_number = resultSet.getString("route\_number");  
 String startPoint = resultSet.getString("start\_point");  
 String endPoint = resultSet.getString("end\_point");  
 String duration = resultSet.getString("duration");  
 String departure\_datetime = resultSet.getString("departure\_datetime");  
 String arrival\_datetime = resultSet.getString("arrival\_datetime");  
 String seat\_number = resultSet.getString("seat\_number");  
 String price = resultSet.getString("price");  
 String ticketStatus = "";  
 String ticketSostQuery = "SELECT sost FROM ticketSost WHERE idTicket = ?";  
 PreparedStatement sostStatement = conn.prepareStatement(ticketSostQuery);  
 sostStatement.setInt(1, Integer.parseInt(ticket\_number));  
 ResultSet sostResultSet = sostStatement.executeQuery();  
 if (!sostResultSet.next()) {  
 ticketStatus = "Куплен";  
 } else {  
 boolean sost = sostResultSet.getBoolean("sost");  
 if (!sost) {  
 ticketStatus = "Возврат на рассмотрении";  
 } else {  
 ticketStatus = "Билет возвращен";  
 }  
 }  
 String[] ticketInfo = {ticket\_number, route\_number, startPoint, endPoint, duration, departure\_datetime, arrival\_datetime, seat\_number, price, ticketStatus};  
 ticketList.add(ticketInfo);  
 }  
 resultSet.close();  
 statement.close();  
  
 return ticketList;  
 } catch (SQLException e) {  
 throw new RuntimeException(e);  
 }  
   
 public List<String[]> getTicketsBySOST\_0() {  
 List<String[]> ticketList = new ArrayList<>();  
  
 try (Connection conn = DbConnection.getConnection()) {  
 String sqlQuery = "SELECT "  
 + "t.ticket\_number, "  
 + "r.route\_number, "  
 + "r.start\_point, "  
 + "r.end\_point, "  
 + "r.duration, "  
 + "r.departure\_datetime, "  
 + "r.arrival\_datetime, "  
 + "t.seat\_number, "  
 + "t.price "  
 + "FROM ticket AS t "  
 + "JOIN route AS r "  
 + "ON t.route\_id = r.id "  
 + "JOIN ticketSost AS ts "  
 + "ON t.ticket\_number = ts.idTicket "  
 + "WHERE ts.sost = 0;";  
 PreparedStatement statement = conn.prepareStatement(sqlQuery);  
 ResultSet resultSet = statement.executeQuery();  
  
 while (resultSet.next()) {  
 String[] ticketData = new String[9];  
 ticketData[0] = resultSet.getString("ticket\_number");  
 ticketData[1] = resultSet.getString("route\_number");  
 ticketData[2] = resultSet.getString("start\_point");  
 ticketData[3] = resultSet.getString("end\_point");  
 ticketData[4] = resultSet.getString("duration");  
 ticketData[5] = resultSet.getString("departure\_datetime");  
 ticketData[6] = resultSet.getString("arrival\_datetime");  
 ticketData[7] = resultSet.getString("seat\_number");  
 ticketData[8] = resultSet.getString("price");  
 ticketList.add(ticketData);  
 }  
 } catch (SQLException e) {  
 e.printStackTrace();  
 }  
 return ticketList;  
 }  
  
 @Override  
 public boolean ticketExists(int ticketNumber) {  
 String query = "SELECT COUNT(\*) AS count FROM ticket WHERE ticket\_number = ?";  
 try (Connection connection = DbConnection.getConnection();  
 PreparedStatement statement = connection.prepareStatement(query)) {  
 statement.setInt(1, ticketNumber);  
  
 ResultSet resultSet = statement.executeQuery();  
 resultSet.next();  
  
 int count = resultSet.getInt("count");  
 return count > 0;  
 } catch (SQLException e) {  
 e.printStackTrace();  
 return false;  
 }  
 }  
}

TicketServiceImpl: Класс TicketServiceImpl реализует интерфейс TicketService и предоставляет методы для добавления и удаления билетов на более высоком уровне, абстрагируясь от реализации базы данных. Он также выполняет транзакционные операции, гарантируя целостность данных.

@Service  
public class TicketServiceImpl implements TicketService {  
 @Autowired  
 private TicketDao ticketDao;  
 @Transactional(propagation = Propagation.REQUIRED)  
 public void addTicket(Ticket ticket) {  
 ticketDao.addTicket(ticket);  
 }  
 @Transactional(propagation = Propagation.REQUIRES\_NEW)  
 public void deleteTicket(int ticketNumber) {  
 if (!ticketDao.ticketExists(ticketNumber)) {  
 throw new TicketNotFoundException("Ticket with id " + ticketNumber + " does not exist");  
 }  
 ticketDao.deleteTicket(ticketNumber);  
 }  
 @Transactional(propagation = Propagation.REQUIRED)  
 public void transactionalMethod() {  
 Ticket ticket = new Ticket();  
 addTicket(ticket);  
 deleteTicket(1001);  
 }  
}

## 5.1.2 EJB-модуль 2

RouteRepository: Класс RouteRepository расширяет JpaRepository для сущности Route и предоставляет методы для управления данными маршрутов в базе данных. Он включает методы для поиска маршрутов по пунктам отправления и назначения, а также для удаления маршрутов.

public interface RouteRepository extends JpaRepository<Route, Integer> {  
 @Query("SELECT r.duration FROM Route r")  
 List<Integer> getAllDurations();  
  
 @Query("SELECT r FROM Route r WHERE r.startPoint = ?1 AND r.endPoint = ?2")  
 List<Route> findRoutesByStartAndEndPoint(String startPoint, String endPoint);  
  
 @Modifying  
 @Query("DELETE FROM Route r WHERE r.id = ?1")  
 void deleteRouteById(int id);  
  
}

Ticket: Класс Ticket представляет сущность билета в базе данных и содержит поля, такие как номер билета, маршрут, пользователь, номер места, дата и время отправления и цена.

@Entity  
@Table(name = "ticket")  
public class Ticket {  
 @Id  
 @GeneratedValue(strategy = GenerationType.IDENTITY)  
 @Column(name = "ticket\_number", nullable = false)  
 private int ticketNumber;  
 @ManyToOne  
 @JoinColumn(name = "route\_id")  
 private Route route;  
 @ManyToOne  
 @JoinColumn(name = "user\_id")  
 private MyUser user;  
 @Column(name = "seat\_number", nullable = false)  
 private int seatNumber;  
 @Column(name = "departure\_date\_time", nullable = true)  
 private Date departureDateTime;  
 @Column(name = "price", nullable = false)  
 private double price;  
 public Ticket(int ticketNumber, Route route, MyUser myUser, int seatNumber, Date departureDateTime, double price) {  
 this.ticketNumber = ticketNumber;  
 this.route = route != null ? route : new Route();  
 this.user = myUser;  
 this.seatNumber = seatNumber;  
 this.departureDateTime = departureDateTime;  
 this.price = price;  
 }  
 public Ticket() {  
  
 }  
 public int getTicketNumber() {  
 return ticketNumber;  
 }  
 public void setTicketNumber(int ticketNumber) {  
 this.ticketNumber = ticketNumber;  
 }  
 public Route getRoute() {  
 return route;  
 }  
 public void setRoute(Route route) {  
 this.route = route;  
 }  
 public MyUser getBuyer() {  
 return user;  
 }  
 public void setBuyer(MyUser buyer) {  
 this.user = buyer;  
 }  
 public int getSeatNumber() {  
 return seatNumber;  
   
 public void setSeatNumber(int seatNumber) {  
 this.seatNumber = seatNumber;  
 }  
 public Date getDepartureDateTime() {  
 return departureDateTime;  
 }  
 public void setDepartureDateTime(Date departureDateTime) {  
 this.departureDateTime = departureDateTime;  
 }  
 public double getPrice() {  
 return price;  
 }  
 public void setPrice(double price) {  
 this.price = price;  
 }  
}

## 5.2 Веб-компоненты

Веб-компоненты отвечают за взаимодействие с пользователем и обработку запросов и ответов HTTP. В данном случае используются следующие веб-компоненты:

## 5.2.1 Управление данными и логикой представления

Данные и логика представления управляются с помощью следующих компонентов:

Контроллеры Spring MVC: Контроллеры обрабатывают запросы HTTP и управляют данными представления.

Сервисы: Сервисы предоставляют бизнес-логику и абстрагируют доступ к данным.

Модели: Модели представляют состояние приложения и используются для передачи данных между компонентами.

Эта структура разделяет различные аспекты приложения и облегчает разработку и обслуживание.

## 5.2.2 Форма входа в систему

Приложение использует форму входа по умолчанию от Spring Security, которая генерируется автоматически и отображается по адресу /login. Пользователи могут ввести свои учетные данные на этой форме для входа в систему.

## 5.2.3 JSP-страница register.jsp

JSP-страница register.jsp содержит форму регистрации для создания новых пользователей. Она получает данные пользователя из формы и вызывает метод registerNewUser в контроллере UserController.

<form id="registrationForm" action="/users" method="post">  
 <h1>Регистрация</h1>  
 <label for="name">Имя пользователя:</label>  
 <input type="text" id="name" name="name" required pattern="[A-Za-zА-Яа-я]+" placeholder="Только буквы">  
 <br><br>  
 <label for="password">Пароль:</label>  
 <input type="password" id="password" name="password" required pattern="(?=.\*[A-Za-zА-Яа-я])(?=.\*\d).{4,}" title="Пароль должен содержать как минимум одну букву и одну цифру, не менее 4 символов" placeholder="Буквы и цирры">  
 <br><br>  
 <button type="submit" >Зарегистрироваться</button>  
</form>

@PostMapping("")  
public ResponseEntity<String> registerNewUser(@RequestParam String name, @RequestParam String password) {  
 MyUser newUser = new MyUser();  
 newUser.setName(name);  
 newUser.setPassword(passwordEncoder.encode(password)); // Хэшируем пароль перед сохранением  
 newUser.setRoles("ROLE\_USER"); // Устанавливаем роль пользователя  
 userRepository.save(newUser);  
 return ResponseEntity.ok("User registered successfully");  
}

## 5.2.4 Конфигурирование JSF-приложения

Конфигурация JSF-приложения выполняется в классе SecurityConfig, который:

Отключает защиту от CSRF

Разрешает доступ к странице /users для всех пользователей

Настраивает механизм аутентификации с использованием DaoAuthenticationProvider и UserDetailsService

Внедряет пользовательские классы MyUserDetails и MyUserDetailsService для управления данными пользователя и процесса аутентификации

public class SecurityConfig {  
 @Bean  
 public UserDetailsService userDetailsService() {  
 return new MyUserDetailsService();  
 }  
 @Bean  
 public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {  
 return http.csrf(AbstractHttpConfigurer::disable)  
 .authorizeRequests(authorize -> authorize  
 .antMatchers("/users").permitAll()  
 .anyRequest().authenticated()  
 )  
 .formLogin(form -> form  
 .permitAll()  
 .defaultSuccessUrl("/product")  
 )  
 .authenticationProvider(authenticationProvider())  
 .authorizeHttpRequests(auth -> auth.mvcMatchers("/users", "/addUser", "product").permitAll())  
 .build();  
 }  
 @Bean  
 public AuthenticationProvider authenticationProvider() {  
 DaoAuthenticationProvider provider = new DaoAuthenticationProvider();  
 provider.setUserDetailsService(userDetailsService());  
 provider.setPasswordEncoder(passwordEncoder());  
 return provider;  
 }  
 @Bean  
 public PasswordEncoder passwordEncoder() {  
 return new BCryptPasswordEncoder();  
 }  
}

## 5.3 Физическая структура баз данных

## 5.3.1 База данных busbus

База данных busbus содержит следующие таблицы: users, route, transport, ticket, ticketSost.

Таблица данных users содержит информацию о пользователях, включая: id (первичный ключ), name (уникальный), password, roles

Таблица данных route содержит информацию о маршрутах, включая: id (первичный ключ), route\_number, start\_point, end\_point, duration, departure\_datetime, arrival\_datetime

Таблица данных transport содержит информацию о транспорте, включая: id (первичный ключ), transport\_number, car\_brand, passenger\_capacity, route\_id (иностранный ключ, ссылающийся на таблицу route)

Таблица данных ticket содержит информацию о билетах, включая:

id (первичный ключ), ticket\_number, route\_id (иностранный ключ, ссылающийся на таблицу route), buyer\_id (иностранный ключ, ссылающийся на таблицу users), seat\_number, departure\_date, price

Таблица данных ticketSost содержит информацию о состоянии билета, включая: idTicket (иностранный ключ, ссылающийся на таблицу ticket), sost (состояние билета: 0 – заявка на возврат, 1 - возвращен)

## 5.4 Обеспечение целостности данных

## 5.4.1 Ограничения целостности

Следующие ограничения целостности применяются к базам данных:

Ограничения первичного ключа: Каждая таблица имеет первичный ключ, который гарантирует уникальность каждой записи.

Ограничения внешнего ключа: Таблицы transport и ticket имеют внешние ключи, которые ссылаются на таблицу route, что гарантирует, что транспортные средства и билеты связаны с существующими маршрутами. Таблица ticket также имеет внешний ключ, который ссылается на таблицу users, что гарантирует, что билеты связаны с существующими пользователями.

Ограничения NOT NULL: Некоторые столбцы определены как NOT NULL, что гарантирует, что они всегда содержат значение.

## 5.4.2 Организация распределенных транзакций

В данном приложении используется аннотация @Transactional для управления распределенными транзакциями. Аннотация @Transactional определяет границы транзакции в методах класса RouteService. Она указывает, что все операции, выполняемые в пределах метода, должны быть частью одной транзакции.

В приведенном ниже примере код внутри метода doSomething выполняется как одна транзакция:

@Transactional(propagation = Propagation.REQUIRED)  
public void doSomething(int routeNumber, String startPoint, String endPoint, int duration, LocalDateTime departureDatetime, LocalDateTime arrivalDatetime) {  
 Route route = new Route(routeNumber, startPoint, endPoint, duration, Date.from(departureDatetime.atZone(ZoneId.systemDefault()).toInstant()), Date.from(arrivalDatetime.atZone(ZoneId.systemDefault()).toInstant()));  
 routeService.addRoute(route);  
 int routeId = route.getId();  
 Route route2 = routeRepository.findById(routeId).orElse(null);  
 System.out.println(route2.getRouteNumber());  
}

Аннотация @Transactional гарантирует, что все изменения базы данных, сделанные в пределах этого метода, будут либо полностью зафиксированы, либо полностью отменены, обеспечивая целостность данных.

Распространение транзакций

Аннотация @Transactional также позволяет указать распространение транзакции. Распространение транзакции определяет, как метод будет вести себя в отношении существующих транзакций. В данном примере используются следующие варианты распространения транзакций:

Propagation.REQUIRES\_NEW: метод создает новую транзакцию, независимо от того, существует ли уже какая-либо транзакция.

Propagation.REQUIRED: метод присоединяется к существующей транзакции или создает новую, если она не существует.

@Transactional(propagation = Propagation.REQUIRES\_NEW)  
public void addRoute(Route route) {  
 routeRepository.save(route);  
}  
  
@Transactional(propagation = Propagation.REQUIRED)  
public Route getRouteById(int id) {  
 return routeRepository.findById(id).orElse(null);  
}

Использование этих вариантов распространения транзакций гарантирует, что операции, выполняемые в методах addRoute и getRouteById, будут выполняться в рамках одной и той же транзакции, даже если вызывающий код уже выполняется в другой транзакции.

Таким образом, аннотация @Transactional является мощным инструментом для управления распределенными транзакциями в Java-приложениях. Она помогает гарантировать целостность данных и обеспечивает согласованность транзакций.

# 6 ИНСТРУКЦИЯ ПО УСТАНОВКЕ ПРИЛОЖЕНИЯ

В данной инструкции представлены шаги по установке и запуску приложения на основе Spring Boot. Процесс включает в себя загрузку исходного кода из публичного репозитория, настройку окружения и запуск приложения.

Инструкция по установке приложения:

1. Загрузка исходного кода:

- Клонируйте репозиторий с исходным кодом вашего приложения из публичного репозитория на GitHub или другой платформе.

2. Открытие проекта в IntelliJ IDEA:

- Откройте IntelliJ IDEA.

- Выберите опцию "Open" и укажите путь к каталогу с вашим проектом.

3.Настройка проекта:

- Убедитесь, что все зависимости указаны правильно в файле pom.xml.

- Проверьте конфигурацию базы данных в файле application.properties или application.yml.

4. Сборка проекта:

- Используйте Maven для сборки проекта.

- Выполните команду сборки проекта через терминал IntelliJ IDEA.

5. Запуск приложения:

- Найдите класс, содержащий метод main и запустите его.

- После успешного запуска приложения откройте браузер и перейдите по URL-адресу http://localhost:8080 для проверки работоспособности.

6. Добавление тестовых данных:

- Создайте тестовые данные в базе данных для проверки функциональности вашего приложения.

- Можно использовать SQL-скрипты, миграции или специальные инструменты для добавления тестовых данных.

7. Тестирование приложения:

- Проведите тестирование функциональности приложения, убедившись, что все работает корректно с добавленными тестовыми данными.

# 7 ОРГАНИЗАЦИЯ УПРАВЛЕНИЯ ПРОЕКТОМ

## 

## 7.1 Общие принципы взаимодействия в команде (работа выполнялась в одиночку)

**Самостоятельность и ответственность:** нес полную ответственность за все аспекты разработки проекта, принимал решения и выполнял задачи самостоятельно.

**Планирование и самоорганизация:** самостоятельно планировал свою работу, устанавливал сроки и следовал им, чтобы обеспечить своевременное завершение проекта.

**Самостоятельное обучение и поиск решений:** изучал необходимые технологии и искал решения проблем, возникающих в процессе разработки.

## 7.2 Распределение ролей в команде/зоны ответственности

**Аналитик**

В качестве аналитика я определил требования к проекту и спроектировал архитектуру приложения. Я проанализировал бизнес-потребности и разработал функциональные спецификации, которые легли в основу проектирования приложения.

**Разработчик**

Будучи единственным разработчиком, я отвечал за реализацию всех функций приложения. Я написал код, реализовал алгоритмы и интегрировал различные компоненты в единое целое.

**Тестировщик**

Я самостоятельно тестировал приложение, чтобы выявить и исправить ошибки.

**Менеджер проекта**

Я также выступил в качестве менеджера проекта, спланировав ход разработки, контролируя прогресс и принимая решения. Я установил сроки, назначил задачи и отслеживал ход проекта, чтобы обеспечить его своевременное и успешное завершение.

## 7.3 Календарный план работ

Диаграмма ганта представлена на рисунке 13:

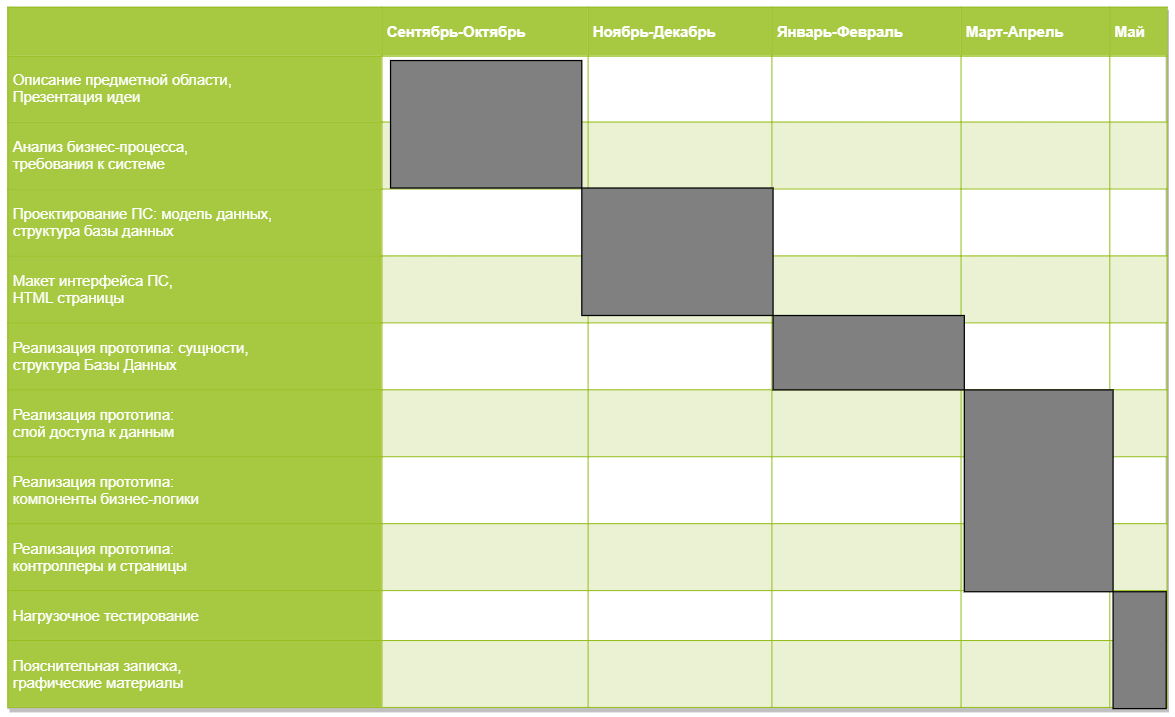
****

Рисунок 14. Диаграмма ганта

# 8 НАГРУЗОЧНОЕ ТЕСТИРОВАНИЕ

Для тестирования производительности SpringBoot веб-приложения было использовано программное обеспечение под названием Jmeter.

Для создания тестирования создадим группу потоков из 100 пользователей и 100 циклов

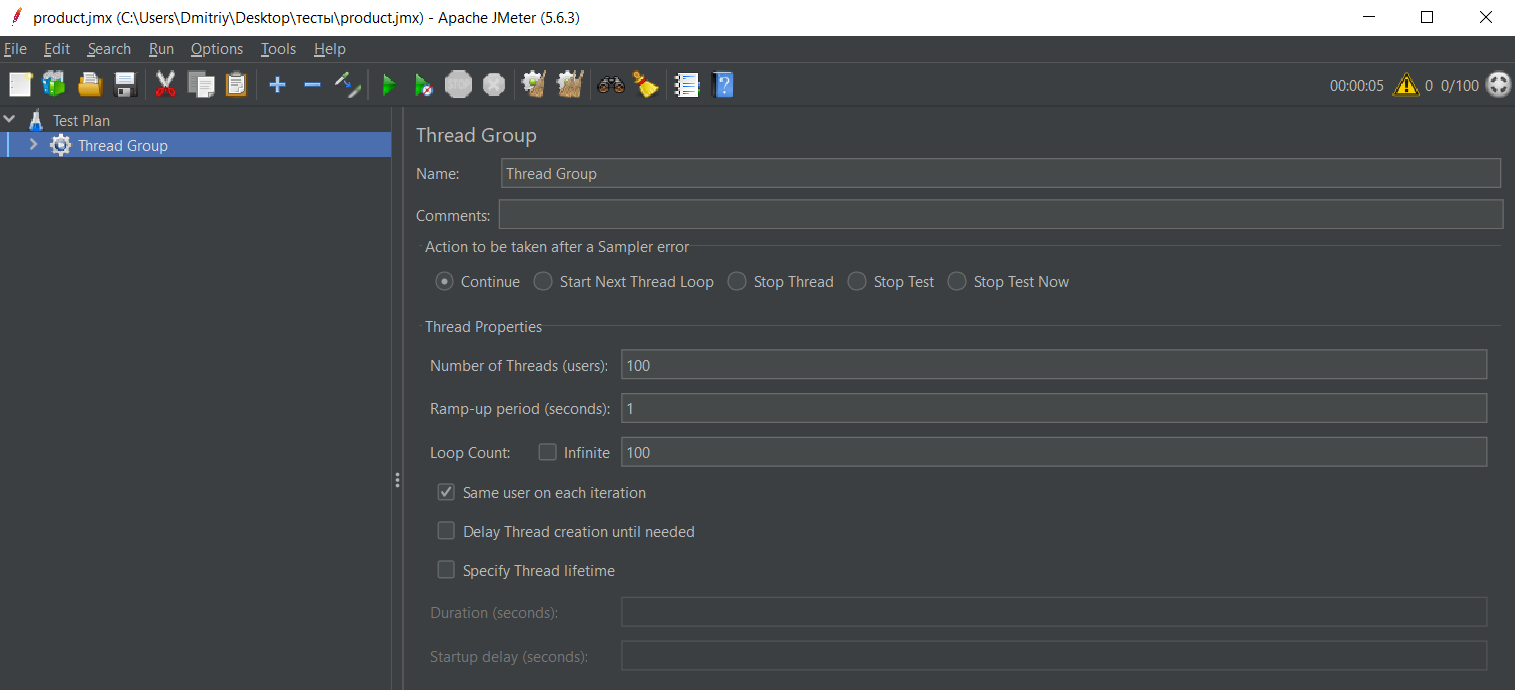
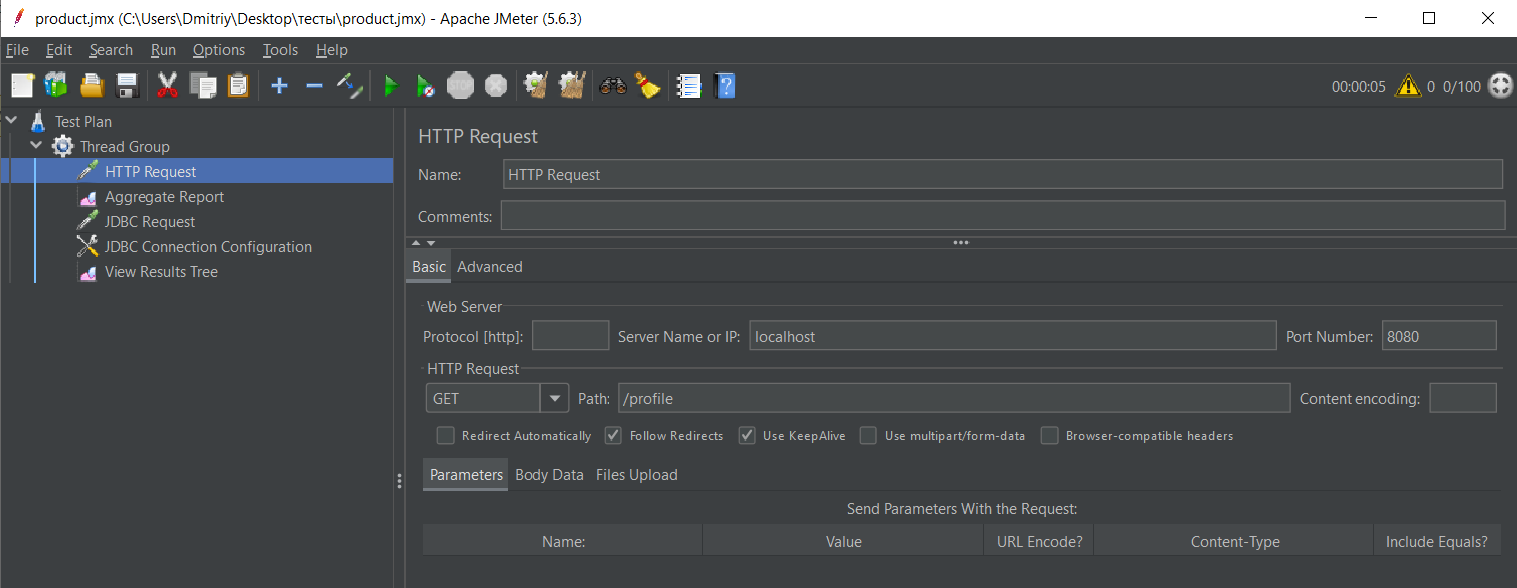


Рисунок 15. группа потоков

Указываем HTTP запрос на наше приложение, на страницу профиля

Рисунок 16. HTTP запрос

Далее указываем JDBC Configuration настройки для взаимодействия с MuSQL

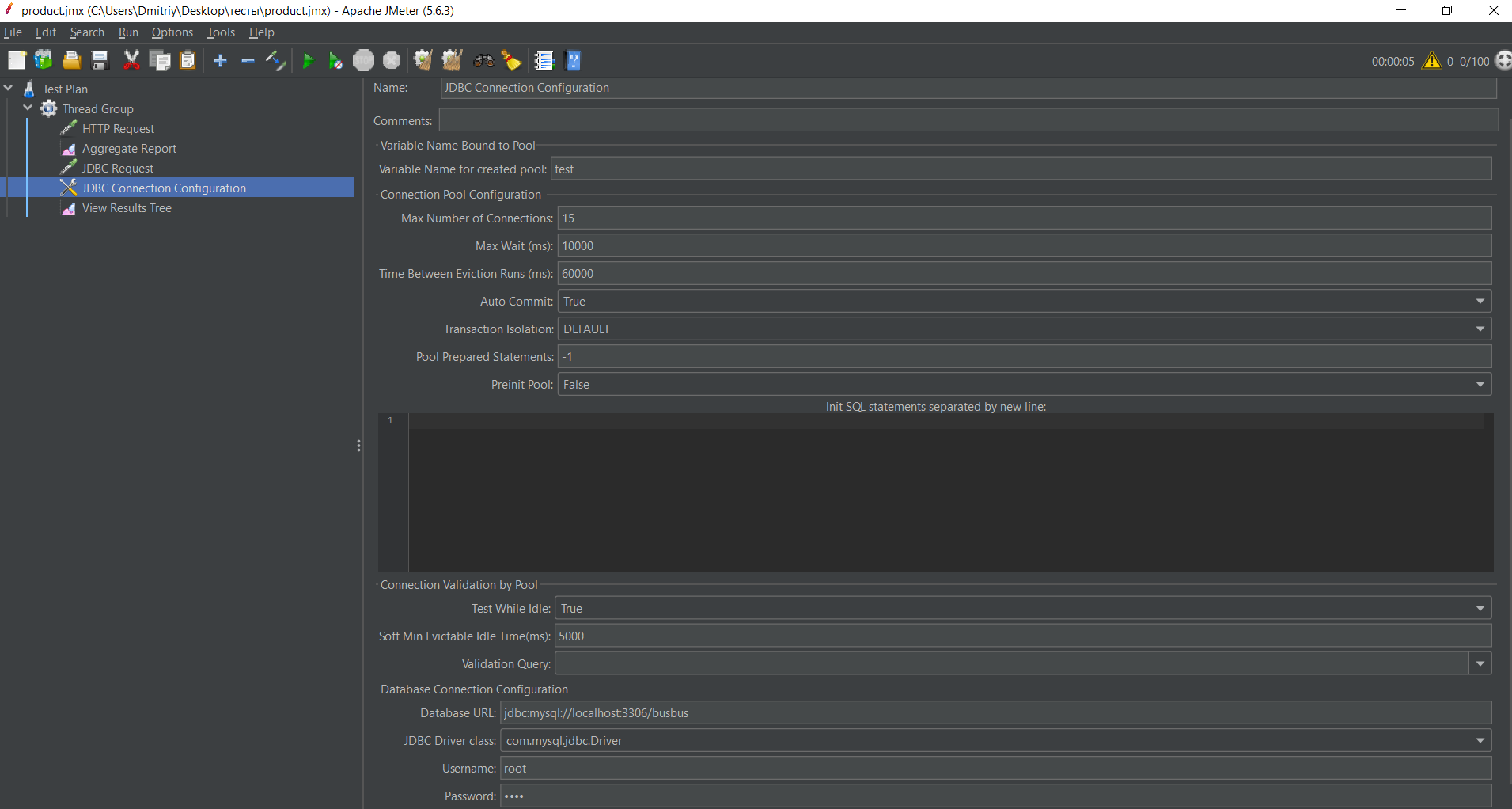


Рисунок 17. JDBC Configuration

Запрос к базе данных на получение всех билетов

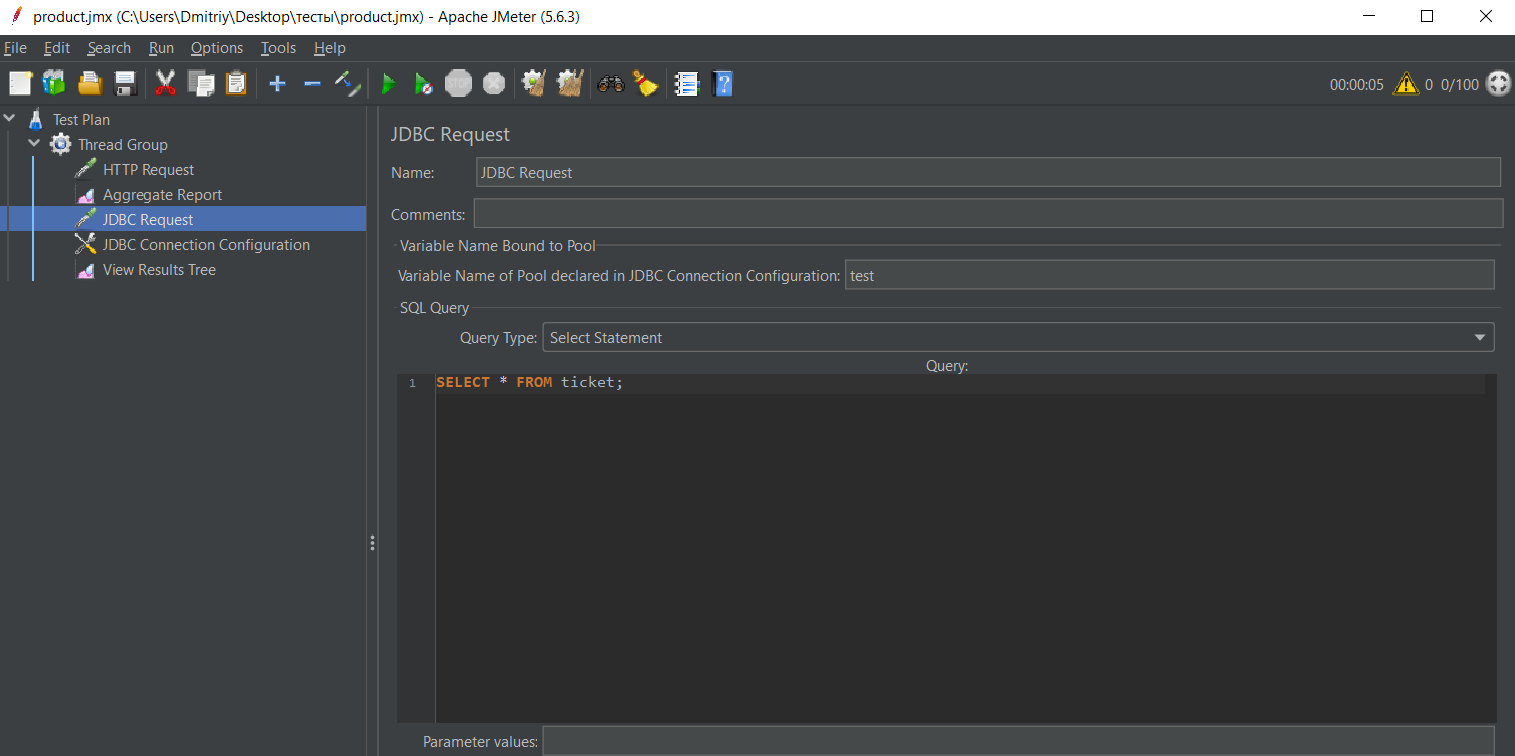


Рисунок 18. Запрос к бд

Создаем listener для отображения результатов тестирования

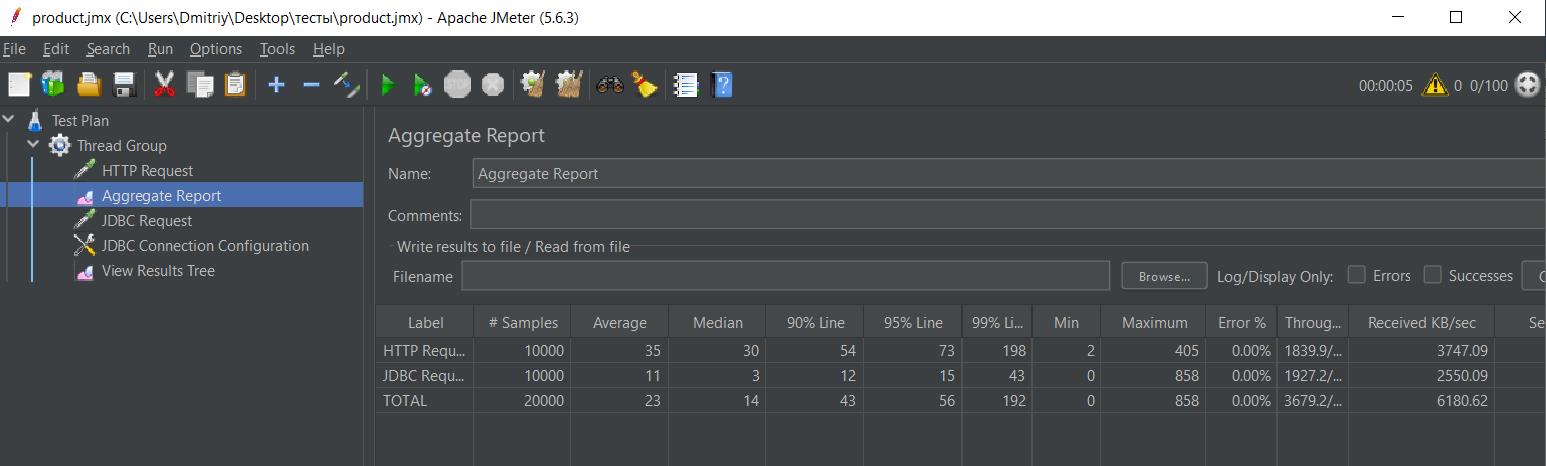


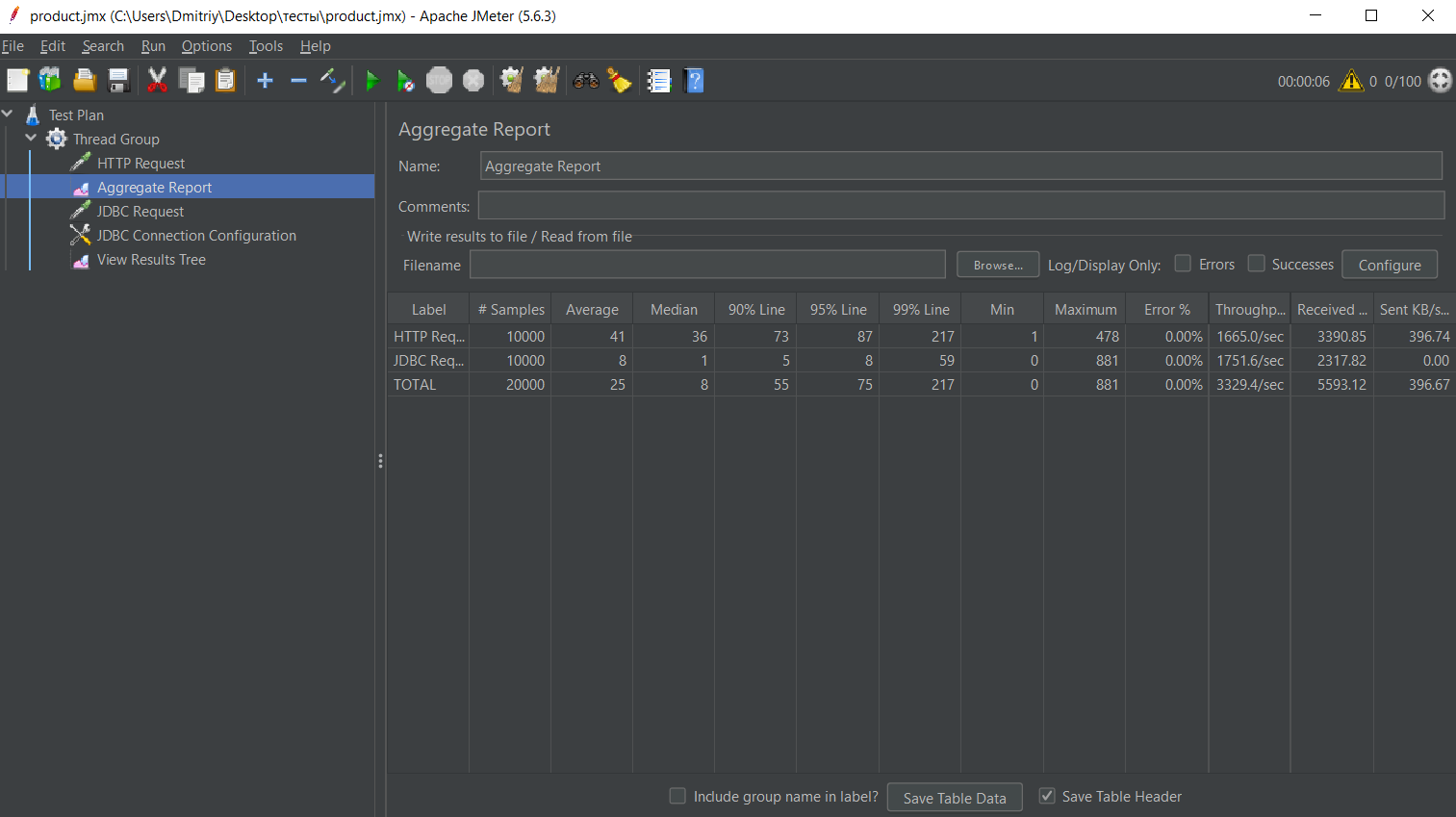
Рисунок 19. listener

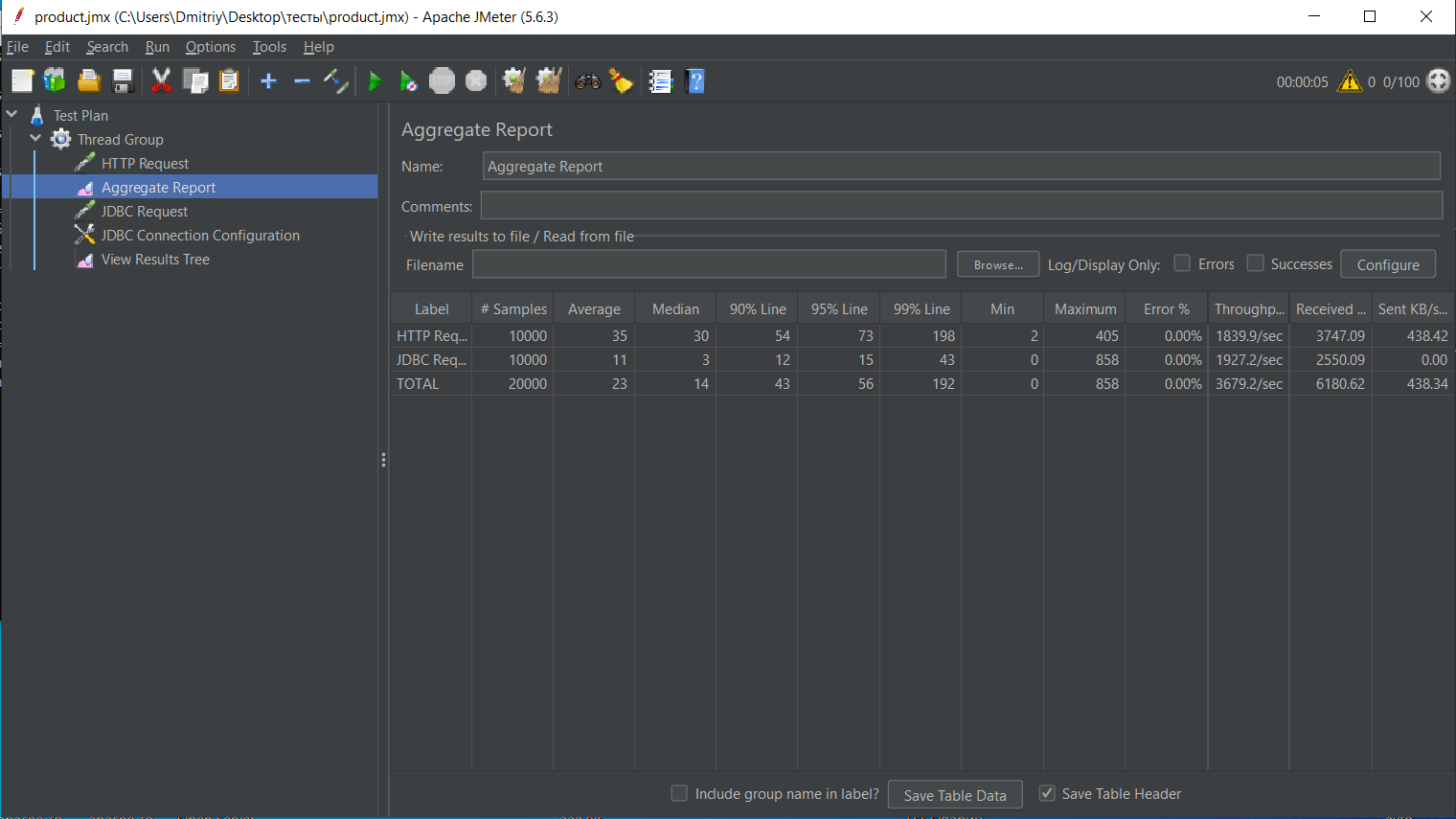
Выполним оптимизацию своего приложение и через нагрузочный тест покажем улучшения.

Внесем изменения в контроллер:

@Controller  
public class ProfileController {  
  
 @Autowired  
 private AppService service;  
  
 @Autowired  
 private AppService appService;  
  
 @GetMapping("/profile")  
 public String showProfilePage(Model model) {  
 MyUserDetails user = service.getCurrentUser();  
 model.addAttribute("user", user);  
 int userId = Math.toIntExact(user.getId()); // Получаем ID пользователя  
 TicketDaoImpl ticketDao = new TicketDaoImpl();  
 MyUserDetails currentUser = appService.getCurrentUser();  
  
 // Получаем список билетов по идентификатору пользователя  
 List<String[]> ticketList = ticketDao.getTicketsByUserId(userId);  
 // Передаем список билетов в модель  
 model.addAttribute("ticketList", ticketList);  
 model.addAttribute("hasRoleAdmin", currentUser.isAdmin());  
 return "profile";  
 }

// используем ленивую загрузку, чтобы отложить загрузку билетов до тех пор, пока они не понадобятся.  
 @LazyCollection(LazyCollectionOption.FALSE)  
 private List<String[]> ticketList;  
  
}

Рисунок 20. До оптимизации

Рисунок 21. После оптимизации

HTTP Request:

1) Среднее время отклика снизилось с 41 мс до 35 мс, а медиана - с 36 мс до 30 мс. Это указывает на общее улучшение производительности запросов HTTP.

2) Пропускная способность увеличилась с 1665,0 до 1839,9 запросов в секунду. Это означает, что сервер может обрабатывать больше запросов за то же время.

JDBC Request:

1) Пропускная способность увеличилась с 1751,6 до 1927,6 запросов в секунду. Это указывает на улучшение производительности запросов JDBC.

В целом, оптимизация с помощью ленивой загрузки привела к улучшению производительности как HTTP-запросов, так и JDBC-запросов. Это указывает на то, что оптимизация была успешной.

Определим пределы нагрузки, при которых начинаются появляться ошибки:

Предел, при котором появляются ошибки на HTTP Request это 3000пользователей на 100циклов:

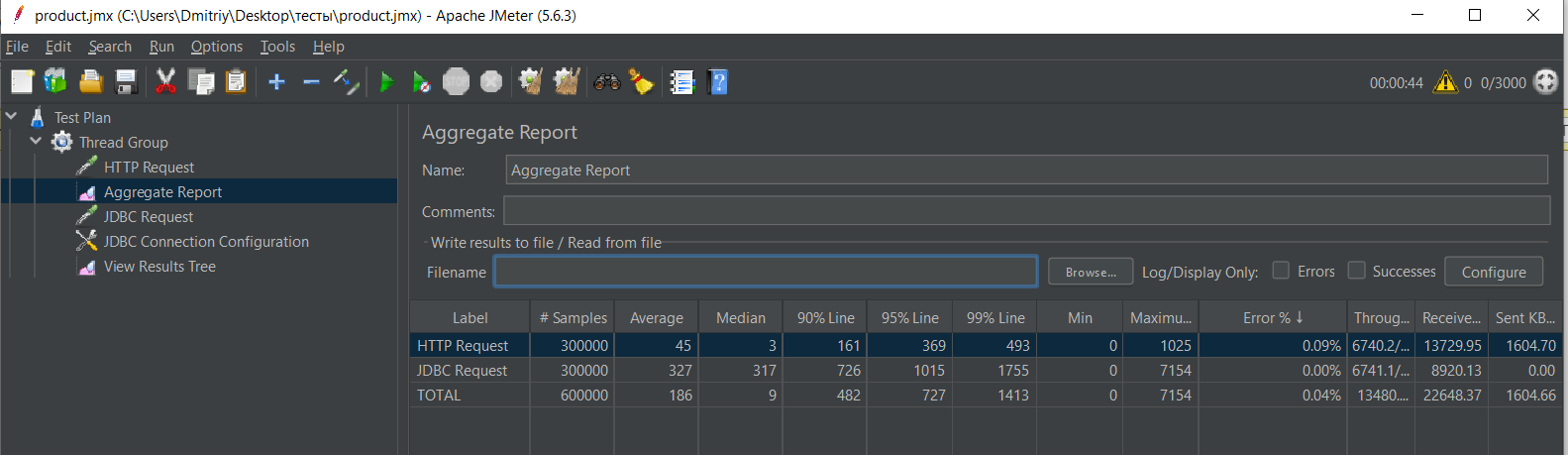


Рисунок 22. Предел HTTP Request

Предел, при котором появляются ошибки на JDBC Request это 9600пользователей на 150циклов:

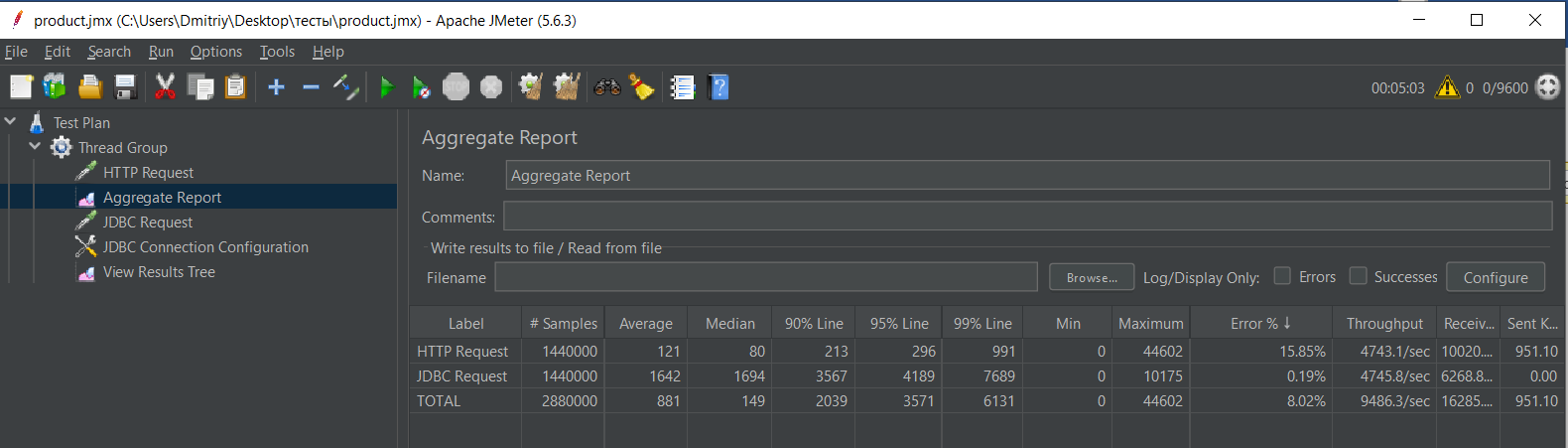


Рисунок 23. Предел JDBC Request

# 9 ЗАКЛЮЧЕНИЕ

В рамках данного курсового проекта была разработана функционирующая программная система для информационной системы автотранспортного предприятия. Система предоставляет возможность отображения маршрутов с указанием расписания, поиска маршрутов по различным критериям, выбора места в автобусе, покупки билетов, возврата билетов, просмотра истории покупок билетов и ведения учетной записи пользователя.

Система реализована как многопользовательское веб-приложение, доступное для пользователей через браузер.

Программная система полностью соответствует поставленным требованиям и способна облегчить процессы управления автотранспортным предприятием. Она может использоваться для эффективного управления перевозками, взаимодействия с клиентами и обеспечения оперативного доступа к необходимым данным.

В дальнейшем развитие системы можно продолжить, добавляя новые функциональные возможности и улучшая пользовательский интерфейс. Например, можно интегрировать систему с платежными системами для автоматизации процесса оплаты билетов, а также добавить возможность управления автопарком и персоналом предприятия.

# 10 СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1) Будилов В. А. Интернет-программирование на Java. – СПб.: БХВ-Петербург, 2003

2) Дейтел Х. М., Дейтел П. Дж., Сантри С. И. Технологии программирования на Java 2. – М.: ООО "Бином-Пресс", 2003

3) Иванова Е. Б., Вершинин М. М. Java 2 Enterprise Edition. Технологии проектирования и разработки. – СПб.: БХВ-Петербург, 2003

4) Смирнов Н. Java 2 Enterprise. Основы практической разработки распределенных корпоративных приложений. – М.: Кудиц-образ, 2002

5) ГОСТ 2.105-95 Общие требования к текстовым документам

6) СТП 71.3-04 Дипломное проектирование. Обозначения в документах выпускных квалификационных работ

7) Sun Microsystems, Inc. Enterprise JavaBeans™, Version 3.0: EJB Core Contracts and Requirements, May 2, 2006

8) Sun Microsystems, Inc. Enterprise JavaBeans™, Version 3.0: Java Persistence API, May 2, 2006

9) Sun Microsystems, Inc. Java Platform, Enterprise Edition (Java EE) Specification, v5, April 28, 2006

10) Sun Microsystems, Inc. Java Servlet Specification, v2.5, May 8, 2006

11) Sun Microsystems, Inc. JavaServer Faces Specification, v1.2, May 8, 2006

12) Sun Microsystems, Inc. JavaServer Pages Specification, v2.1, May 8, 2006

# ПРИЛОЖЕНИЕ А Исходный код программной системы

Ссылка: <https://github.com/javavlsu/course-work-team-13-oparin>

<?xml version="1.0" encoding="UTF-8"?>  
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
 xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">  
 <modelVersion>4.0.0</modelVersion>  
 <parent>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-parent</artifactId>  
 <version>2.6.2</version>  
 <relativePath/>  
 </parent>  
 <groupId>com.oparin</groupId>  
 <artifactId>busbus</artifactId>  
 <version>0.0.1-SNAPSHOT</version>  
 <name>busbus</name>  
 <description>Demo project for Spring Boot</description>  
 <properties>  
 <java.version>17</java.version>  
 </properties>  
  
 <dependencies>  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-data-jpa</artifactId>  
 </dependency>  
  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-tomcat</artifactId>  
 <version>2.6.2</version>  
 </dependency>  
  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-security</artifactId>  
 <version>2.6.2</version>  
 </dependency>

<dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-thymeleaf</artifactId>  
 </dependency>

<dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-web</artifactId>  
 </dependency>

<dependency>  
 <groupId>org.springframework.security</groupId>  
 <artifactId>spring-security-taglibs</artifactId>  
 <version>6.0.8</version>  
 </dependency>

<dependency>  
 <groupId>mysql</groupId>  
 <artifactId>mysql-connector-java</artifactId>  
 <version>8.0.28</version>  
 <scope>runtime</scope>  
 </dependency>

<dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-test</artifactId>  
 <scope>test</scope>  
 </dependency>

<dependency>  
 <groupId>org.springframework.security</groupId>  
 <artifactId>spring-security-test</artifactId>  
 <version>6.0.8</version>  
 <scope>test</scope>  
 </dependency>  
  
 <dependency>  
 <groupId>nz.net.ultraq.thymeleaf</groupId>  
 <artifactId>thymeleaf-layout-dialect</artifactId>  
 </dependency>

<dependency>  
 <groupId>org.projectlombok</groupId>  
 <artifactId>lombok</artifactId>  
 <version>1.18.30</version>  
 <scope>provided</scope>  
 </dependency>

</dependencies>  
 <build>  
 <plugins>  
 <plugin>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-maven-plugin</artifactId>  
 </plugin>  
 </plugins>  
 </build>  
</project>

spring:  
 datasource:  
 driver-class-name: com.mysql.cj.jdbc.Driver  
 url: jdbc:mysql://localhost:3306/busbus?serverTimezone=UTC  
 username: root  
 password: root  
 jpa:  
 hibernate:  
 ddl-auto: update  
 mvc:  
 view:  
 prefix: classpath:/templates/  
 suffix: .html  
 jpa.properties:  
 hibernate.packagesToScan: com.oparin.busbus.models  
 hibernate.jdbc.time\_zone: Europe/Moscow  
 hibernate.show\_sql: true

@Controller  
@RequestMapping("/users")  
public class UserController {  
  
 @Autowired  
 private AppService service;  
  
 @Autowired  
 private UserRepository userRepository;  
  
 @Autowired  
 private PasswordEncoder passwordEncoder;  
  
 @GetMapping("")  
 public String showAddUserForm(Model model) {  
 model.addAttribute("user", new MyUser()); //изменил User на MyUser  
 return "addUser"; // Возвращаем имя представления с формой  
 }  
  
 @PostMapping("")  
 public ResponseEntity<String> registerNewUser(@RequestParam String name, @RequestParam String password) {  
 MyUser newUser = new MyUser();  
 newUser.setName(name);  
 newUser.setPassword(passwordEncoder.encode(password)); // Хэшируем пароль перед сохранением  
 newUser.setRoles("ROLE\_USER"); // Устанавливаем роль пользователя  
 userRepository.save(newUser);  
 return ResponseEntity.ok("User registered successfully");  
 }  
}

@Controller  
public class TicketController {  
 private final RouteDaoImpl routeDao = new RouteDaoImpl();  
 private final TicketDaoImpl ticketDao = new TicketDaoImpl();  
 @Autowired  
 private AppService appService;  
 @PostMapping("/purchaseTicket")  
 @Transactional  
 public String purchaseTicket(@RequestParam("routeId") int routeId,  
 @RequestParam("departureFrom") String startPoint,  
 @RequestParam("departureTo") String endPoint,  
 @RequestParam("duration") int duration,  
 @RequestParam("departureDatetime") String departureDatetime,  
 @RequestParam("arrivalDatetime") String arrivalDatetime,  
 @RequestParam("price") int price,  
 @RequestParam(value="seatNumber", required = false) Integer seatNumber,  
 Model model, HttpServletRequest req) throws ParseException, RollbackException {  
 Route route = routeDao.getRouteById(routeId);  
 if (route == null) {  
 // Обработать ошибку  
 return "redirect:/errorPage";  
 }  
 int id = ticketDao.max\_id();  
 MyUserDetails currentUser = appService.getCurrentUser();  
 if (currentUser == null) {  
 return "redirect:/errorPage";  
 }  
 MyUser user = currentUser.getUser();  
 String modifiedDateString = departureDatetime.replaceFirst("-Wed-", "-");  
 SimpleDateFormat sdf = new SimpleDateFormat("EEE MMM dd HH:mm:ss zzz yyyy", Locale.ENGLISH);  
 Date date = sdf.parse(modifiedDateString);  
 Ticket ticket = new Ticket(id, route, user, seatNumber, date, (double) price);  
 ticketDao.addTicket(ticket);  
 return "redirect:/profile?refresh";  
 }  
 @PostMapping("/user\_return"  
 public String user\_return\_ticket(@RequestParam("ticketId") int ticketId) throws RollbackException {  
 String query = "INSERT INTO ticketSost (idTicket, sost) VALUES (?, 0)";  
 try (Connection connection = DbConnection.getConnection();  
 PreparedStatement statement = connection.prepareStatement(query)) {  
 statement.setInt(1, ticketId);  
 statement.executeUpdate();  
 } catch (SQLException e) {  
 e.printStackTrace();  
 }  
 return "redirect:/profile?refresh";  
 }  
 @GetMapping("/dispet\_return")  
 public String showTickets(Model model) {  
 List<String[]> ticketList = ticketDao.getTicketsBySOST\_0();  
 MyUserDetails currentUser = appService.getCurrentUser();  
 model.addAttribute("hasRoleAdmin", currentUser.isAdmin());  
 model.addAttribute("ticketList", ticketList);  
 return "dispet";  
 }  
 @PostMapping("/dispet\_return")  
 public String dispet\_return\_ticket(@RequestParam("ticketId") int ticketId) {  
 String query = "UPDATE ticketSost SET sost = 1 WHERE idTicket = ?";  
 try (Connection connection = DbConnection.getConnection();  
 PreparedStatement statement = connection.prepareStatement(query)) {  
 statement.setInt(1, ticketId);  
 statement.executeUpdate();  
 } catch (SQLException e) {  
 e.printStackTrace();  
 }  
 return "redirect:/dispet\_return";  
 }  
 @PostMapping("/Ticket/buy")  
 public String buyTicket(@RequestParam("routeId") int routeId,  
 @RequestParam("startPoint") String startPoint,  
 @RequestParam("endPoint") String endPoint,  
 @RequestParam("duration") int duration,  
 @RequestParam("departureDatetime") String departureDatetime,  
 @RequestParam("arrivalDatetime") String arrivalDatetime,  
 @RequestParam("price") int price,  
 Model model) {  
 Route route = routeDao.getRouteById(routeId);  
 TicketDaoImpl yourClass = new TicketDaoImpl();  
 List<Integer> result = yourClass.getAvailableSeatNumbers(routeId);  
 model.addAttribute("startPoint", startPoint);  
 model.addAttribute("endPoint", endPoint);  
 model.addAttribute("duration", duration);  
 model.addAttribute("departureDatetime", departureDatetime);  
 model.addAttribute("arrivalDatetime", arrivalDatetime);  
 model.addAttribute("price", price);  
 model.addAttribute("route", route);  
 model.addAttribute("seats", result);  
 return "buyTicket";  
 }  
  
 @GetMapping("/Ticket/buy") // Добавьте метод GET  
 public String buyTicketForm(Model model) {  
 List<Route> routes = routeDao.getAllRoutes();  
 model.addAttribute("routes", routes);  
 return "buyTicket";  
 }  
}

@Controller  
@RequestMapping("/routes")  
public class RouteController {  
 @Autowired  
 private AppService appService;  
 @Autowired  
 private RouteService routeService;  
 private final RouteDaoImpl routeDao = new RouteDaoImpl();  
 @GetMapping("")  
 @PreAuthorize("hasRole('ROLE\_ADMIN') || hasRole('ROLE\_USER')")  
 public String showFilteredAndSortedRoutes(@RequestParam(required = false) String from,  
 @RequestParam(required = false) String to,  
 @RequestParam(required = false) @DateTimeFormat(iso = DateTimeFormat.ISO.DATE) Date date,  
 @RequestParam(required = false, defaultValue = "ASC") String order,  
 @RequestParam(required = false, defaultValue = "0") int page,  
 @RequestParam(required = false, defaultValue = "3") int size,  
 Model model) {  
 List<Route> routes = routeService.getAllRoutes();  
 if (from != null && !from.isEmpty() && to != null && !to.isEmpty() && date != null) {  
 LocalDate localDate = date.toInstant().atZone(ZoneId.systemDefault()).toLocalDate();  
 routes = routes.stream()  
 .filter(route -> route.getStartPoint().equalsIgnoreCase(from))  
 .filter(route -> route.getEndPoint().equalsIgnoreCase(to))  
 .filter(route -> route.getDepartureDatetime().toInstant().atZone(ZoneId.systemDefault()).toLocalDate().isEqual(localDate))  
 .collect(Collectors.toList());  
 }  
 if (date != null) {  
 LocalDate localDate = date.toInstant().atZone(ZoneId.systemDefault()).toLocalDate();  
 routes = routes.stream()  
 .filter(route -> route.getDepartureDatetime().toInstant().atZone(ZoneId.systemDefault()).toLocalDate().isEqual(localDate))  
 .collect(Collectors.toList());  
 }  
 int totalPages = (int) Math.ceil((double) routes.size() / size);  
 if (page > totalPages) {  
 page = totalPages;  
 }  
 if (page < 0) {  
 page = 0;  
 }  
 List<Route> pagedRoutes = routes.subList(page \* size, Math.min(routes.size(), (page + 1) \* size));  
 if ("ASC".equals(order)) {  
 pagedRoutes.sort(Comparator.comparingInt(Route::getDuration));  
 } else if ("DESC".equals(order)) {  
 pagedRoutes.sort(Comparator.comparingInt(Route::getDuration).reversed());  
 }  
 model.addAttribute("routes", pagedRoutes);  
 model.addAttribute("currentPage", page);  
 model.addAttribute("totalPages", totalPages);  
 String greetingMessage = "";  
 MyUserDetails currentUser = appService.getCurrentUser();  
 if (currentUser != null) {  
 if (currentUser.isAdmin()) {  
 greetingMessage = "Привет, администратор!";  
 } else {  
 greetingMessage = "Привет, обычный пользователь!";  
 }  
 }  
 model.addAttribute("hasRoleAdmin", currentUser.isAdmin());  
 model.addAttribute("title", "Product Page");  
 model.addAttribute("greetingMessage", greetingMessage);  
 return "routes";  
 }  
 @PostMapping("")  
 @PreAuthorize("hasRole('ROLE\_ADMIN')")  
 public String addRoute(@RequestParam int routeNumber,  
 @RequestParam String startPoint,  
 @RequestParam String endPoint,  
 @RequestParam int duration,  
 @RequestParam @DateTimeFormat(iso = DateTimeFormat.ISO.DATE\_TIME) LocalDateTime departureDatetime,  
 @RequestParam @DateTimeFormat(iso = DateTimeFormat.ISO.DATE\_TIME) LocalDateTime arrivalDatetime) {  
 appService.doSomething(routeNumber, startPoint, endPoint, duration, departureDatetime, arrivalDatetime);  
  
 return "redirect:/routes";  
 }  
 @PostMapping("/delete")  
 @PreAuthorize("hasRole('ROLE\_ADMIN')")  
 public String deleteRoute(@RequestParam int routeId) {  
 Route route = routeDao.getRouteById(routeId);  
 if (route != null) {  
 routeService.deleteRoute(route);  
 return "redirect:/routes";  
 } else {  
 return "redirect:/routes";  
 }  
 }  
 @PostMapping("/edit")  
 public String editRoute(@RequestParam int routeId, Model model) {  
 Route route = routeService.getRouteById(routeId);  
 if (route != null) {  
 model.addAttribute("route", route);  
 return "edit-route";  
 } else {  
 return "redirect:/routes";  
 }  
 }  
 @PostMapping("/update")  
 public String updateRoute(@ModelAttribute("route") Route route) {  
 if (route != null) {  
 routeService.updateRoute(route);  
 return "redirect:/routes";  
 } else {  
 return "redirect:/routes";  
 }  
 }  
 @PostMapping("/{routeId}/update")  
 public String updateRouteById(@PathVariable int routeId,  
 @RequestParam int routeNumber,  
 @RequestParam String startPoint,  
 @RequestParam String endPoint,  
 @RequestParam int duration,  
 @RequestParam @DateTimeFormat(pattern = "yyyy-MM-dd'T'HH:mm") LocalDateTime departureDatetime,  
 @RequestParam @DateTimeFormat(pattern = "yyyy-MM-dd'T'HH:mm") LocalDateTime arrivalDatetime) {  
 Route route = routeService.getRouteById(routeId);  
 if (route != null) {  
 route.setRouteNumber(routeNumber);  
 route.setStartPoint(startPoint);  
 route.setEndPoint(endPoint);  
 route.setDuration(duration);  
route.setDepartureDatetime(Date.from(departureDatetime.atZone(ZoneId.systemDefault()).toInstant()));  
route.setArrivalDatetime(Date.from(arrivalDatetime.atZone(ZoneId.systemDefault()).toInstant()));  
 routeDao.updateRoute(route);  
 return "redirect:/routes";  
 } else {  
 return "redirect:/routes";  
 }  
 }  
 @GetMapping("/edit")  
 public String showEditRouteForm(@RequestParam int routeId, Model model) {  
 Route route = routeService.getRouteById(routeId);  
 if (route != null) {  
 model.addAttribute("route", route);  
 return "edit-route";  
 } else {  
 return "redirect:/routes";  
 }  
 }  
 @PostMapping("/details")  
 public String showRouteDetails(@RequestParam int routeId, Model model) {  
 Route route = routeService.getRouteById(routeId);  
 if (route != null) {  
 model.addAttribute("route", route);  
 return "route-details";  
 } else {  
 return "redirect:/routes";  
 }  
 }  
 @GetMapping("/routes")  
 public List<Route> findRoutesByStartAndEndPoint(@RequestParam String startPoint, @RequestParam String endPoint) {  
 return routeService.findRoutesByStartAndEndPoint(startPoint, endPoint);  
 }  
}

@Controller  
public class ProfileController {  
 @Autowired  
 private AppService service;  
 @Autowired  
 private AppService appService;  
  
 @GetMapping("/profile")  
 public String showProfilePage(Model model) {  
 MyUserDetails user = service.getCurrentUser();  
 model.addAttribute("user", user);  
 int userId = Math.toIntExact(user.getId()); // Получаем ID пользователя  
 TicketDaoImpl ticketDao = new TicketDaoImpl();  
 MyUserDetails currentUser = appService.getCurrentUser();  
  
 List<String[]> ticketList = ticketDao.getTicketsByUserId(userId);  
 model.addAttribute("ticketList", ticketList);  
 model.addAttribute("hasRoleAdmin", currentUser.isAdmin());  
 return "profile";  
 }  
 @LazyCollection(LazyCollectionOption.FALSE)  
 private List<String[]> ticketList;  
}

# ПРИЛОЖЕНИЕ Б Структура таблиц БД

CREATE DATABASE busbus;

USE busbus;

CREATE TABLE transport (

id INT NOT NULL AUTO\_INCREMENT,

transport\_number INT,

car\_brand VARCHAR(255),

passenger\_capacity INT,

route\_id INT,

PRIMARY KEY (id),

FOREIGN KEY (route\_id) REFERENCES route(id)

);

CREATE TABLE route (

id INT NOT NULL AUTO\_INCREMENT,

route\_number INT,

start\_point VARCHAR(255),

end\_point VARCHAR(255),

duration INT,

departure\_datetime DATETIME,

arrival\_datetime DATETIME,

PRIMARY KEY (id)

);

CREATE TABLE users (

id BIGINT NOT NULL AUTO\_INCREMENT,

name VARCHAR(255),

password VARCHAR(255),

roles VARCHAR(255),

PRIMARY KEY (id)

);

CREATE TABLE ticket (

id INT NOT NULL AUTO\_INCREMENT,

ticket\_number INT,

route\_id INT,

buyer\_id BIGINT,

seat\_number INT,

departure\_date DATE,

price DOUBLE,

user\_id INT,

PRIMARY KEY (id),

FOREIGN KEY (route\_id) REFERENCES route(id),

FOREIGN KEY (buyer\_id) REFERENCES users(id),

);

CREATE TABLE ticketsost (

idTicket INT,

sost TINYINT(1),

PRIMARY KEY (idTicket)

);

INSERT INTO transport (transport\_number, car\_brand, passenger\_capacity, route\_id) VALUES

(1, 'Mercedes-Benz', 50, 1),

(2, 'Volvo', 40, 2),

(3, 'MAN', 60, 3);

INSERT INTO route (route\_number, start\_point, end\_point, duration, departure\_datetime, arrival\_datetime) VALUES

(1, 'Moscow', 'Saint Petersburg', 8, '2023-03-08 10:00:00', '2023-03-08 18:00:00'),

(2, 'Saint Petersburg', 'Moscow', 8, '2023-03-10 10:00:00', '2023-03-10 18:00:00'),

(3, 'Moscow', 'Kazan', 10, '2023-03-12 10:00:00', '2023-03-12 20:00:00');

INSERT INTO users (name, password, roles, username) VALUES

('John Doe', 'password', 'ROLE\_USER'),

('Jane Doe', 'password', 'ROLE\_USER'),

('Admin', 'password', 'ROLE\_ADMIN');

INSERT INTO ticketsost (idTicket, sost) VALUES

(1, 1),

(2, 1),

(3, 1);