

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное автономное образовательное учреждение
высшего образования «Самарский национальный исследовательский
университет имени академика С.П. Королева»
(Самарский университет)

Институт естественнонаучный
Факультет механико-математический
Кафедра информатики и вычислительной математики

Направление подготовки
02.03.03 Математическое обеспечение и
администрирование информационных систем
Направленность (профиль) «Разработка и
администрирование информационных систем»

Курсовая работа по дисциплине «Базы данных и СУБД»

**Проектирование и реализация ИС на основе БД «Касса
автовокзала» в среде PostgreSQL**

Выполнил студент
курса 3 группы 4345-020303D
Хайрулов Максим Алексеевич

Научный руководитель
к.ф.-м.н., доцент
Луканов А.С.

Работа защищена
«_____» _____ 2023 г.
Оценка _____
Зав. кафедрой ИиВМ
д.ф.-м.н., профессор
Степанов А.Н.

Самара 2023

СОДЕРЖАНИЕ

ВВЕДЕНИЕ	3
1 Проектирование информационной системы «Касса автовокзала»	5
1.1 Описание предметной области.....	5
1.2 Обзор системы-аналога «Центральный автовокзал Самары»	6
1.3 Функциональные и нефункциональные требования к информационной системе	7
2 Проектирование и реализация базы данных «Касса автовокзала»	10
2.1 ER – диаграмма	10
2.2 Логическая модель базы данных.....	14
2.3 Нормализация реляционных отношений	15
2.4 Обоснование выбора СУБД.....	18
2.5 Физическая модель базы данных	19
3 Проектирование и реализация информационной системы «Касса автовокзала».....	20
3.1 Архитектура информационной системы.....	20
3.2 Диаграмма вариантов использования.....	22
3.3 Диаграмма последовательности.....	24
3.4 Выбор средств реализации	26
3.5 Реализация серверной части.....	27
3.6 Реализация клиентской части.....	31
3.7 Тестирование информационной системы	36
3.7.1 Модульное тестирование.....	36
3.7.2 Интеграционное тестирование.....	38
3.7.3 Нагрузочное тестирование	42
ЗАКЛЮЧЕНИЕ	46
СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ	49
ПРИЛОЖЕНИЕ А	50
ПРИЛОЖЕНИЕ Б.....	54

ВВЕДЕНИЕ

Информационные технологии в настоящее время широко применяются для автоматизации деятельности на различных предприятиях. Постоянно растет объем и сложность обрабатываемой информации, требуются новые виды ее представления.

Возможность быстрого доступа к любым данным, экономия трудозатрат и затрат времени на обработку информации, возможность быстрого формирования отчетов и многие другие операции производить значительно проще и удобнее, имея дело с электронными документами.

Поэтому информационные технологии внедряются в производственные, медицинские, образовательные, военные и многие другие сферы деятельности. Средства автоматизации намного облегчают жизнь людей. С каждым днем появляются новые информационные технологии, которые обрastaют новыми функциональными возможностями средств автоматизации.

В настоящее время большой процент населения России пользуется услугами автовокзалов, и с каждым годом этот процент увеличивается, в следствии чего кассы испытывают сильные нагрузки, так как невозможно обслужить каждого из них только с помощью билетных касс. И это, несомненно, является недостатком в производительности, из-за чего людям приходится стоять в больших очередях, возрастает количество ошибок.

Решение этой проблемы состоит в том, что часть сделок по продаже автобусных билетов должна осуществляться через терминалы самообслуживания, либо через онлайн сервисы.

В следствии чего возникла необходимость в разработке информационной системы для автоматизации процесса купли-продажи автобусных билетов и просмотра и поиска рейсов транспортных средств.

Целью курсовой работы в рамках освоения компетенций ОПК-3, ОПК-4 и ОПК-5 является разработка информационной системы «Касса автовокзала»,

которая позволит просматривать актуальные рейсы и иметь возможность приобретения автобусных билетов.

Для достижения поставленной цели необходимо решить следующие задачи:

- 1) Провести анализ предметной области.
- 2) Изучить существующую систему-аналог.
- 3) Сформулировать функциональные и нефункциональные требования в системе (ОПК-3).
- 4) Построить информационную модель данных (ОПК-3, ОПК-4).
- 5) Построить логическую модель данных, провести ее нормализацию (ОПК-3, ОПК-4).
- 6) Выбрать СУБД (ОПК-5).
- 7) Построить физическую модель данных (ОПК-3, ОПК-4).
- 8) Реализовать базу данных в выбранной СУБД (ОПК-3, ОПК-5).
- 9) Спроектировать приложение для работы с созданной базой данных (ОПК-3, ОПК-4).
- 10) Выбрать средства реализации приложения для работы с созданной базой данных (ОПК-5).
- 11) Реализовать приложение для работы с созданной базой данных (ОПК-3).
- 12) Провести тестирование реализованной системы.
- 13) Провести проверку работоспособности реализованной системы, оценить полученные результаты. (ОПК-3, ОПК-5).

1 Проектирование информационной системы «Касса автовокзала»

1.1 Описание предметной области

Информационная система (ИС) — система, предназначенная для хранения, поиска и обработки информации, и соответствующие организационные ресурсы (человеческие, технические, финансовые и т. д.), которые обеспечивают и распространяют информацию.

Касса автовокзала предоставляет услуги продажи билетов на рейсы, а также консультирует покупателей для предоставления им актуальной информации о маршрутах, по которым происходит движение автобусов.

Рейс – это законченный транспортный цикл по доставке пассажиров, который включает в себя движение автобуса по маршруту в одном направлении от одного конечного пункта до другого с выполнением всех промежуточных остановок для посадки и высадки пассажиров.

Таким образом, рейс содержит в себе ключевую информацию о автобусе, который осуществляет перевозку, маршруте, по которому осуществляется перевозка, и о дате и времени, когда осуществляется перевозка.

Каждый автобус имеет свои особенности, такие как модель автобуса, номерной знак.

Маршрут предоставляет из себя путь следования с указанием начального, конечного и промежуточных пунктов, в случае их наличия, по которым должен ехать автобус, при этом нельзя высаживать или забирать пассажиров вне остановки своего маршрута. Каждый маршрут имеет свой уникальный номер, а также тариф, по которому будет считаться стоимость поездки на данном маршруте.

Пунктом маршрута считается общественное место остановки транспортных средств по маршруту регулярных перевозок, оборудованное для посадки, высадки пассажиров и ожидания транспортных средств. При этом посадка на автобус может быть осуществлена только на пункте, который считается автовокзалом, высадка же производится на каждом пункте.

Пассажир, для осуществления поездки, должен сначала приобрести билет на соответствующий рейс. В билете указывается покупатель, номер рейса, пункты отправления и прибытия, время отправления и прибытия, а также номер места и стоимость.

1.2 Обзор системы-аналога «Центральный автовокзал Самары»

Web-приложение «Центральный автовокзал Самары» предназначено для просмотра актуальных рейсов и покупки билетов на выбранные места в автобусе. Web-приложение позволяет по двум пунктам, отправления и прибытия, найти рейс в нужную дату. (Рисунок 1).

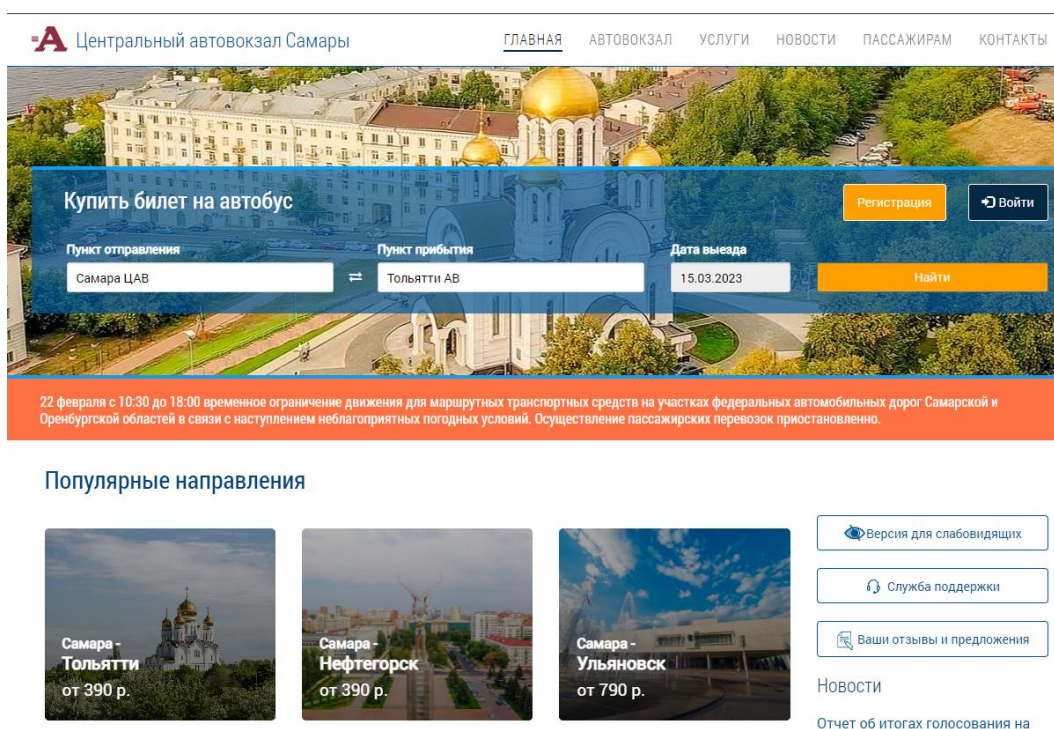


Рисунок 1 – Главная страница центрального автовокзала Самары

Плюсы системы в том, что имеется версия для слабовидящих, служба поддержки и интуитивно понятный интерфейс.

Из минусов этой системы можно выделить, что поиск нужного рейса может осуществляться только с учетом двух пунктов и даты отправления, то

есть система не способна предоставить информацию о рейсах без хотя бы одного из этих трёх данных в запросе.

1.3 Функциональные и нефункциональные требования к информационной системе

Основные функции, которые должна поддерживать разрабатываемая информационная система:

1. регистрация: создание учётной записи пользователя;
2. авторизация: авторизация в системе пользователя по адресу электронной почты и паролю;
3. просмотр рейсов: возможность просмотра списка с данными по рейсам;
4. поиск рейсов: возможность вводить начальную и/или конечную точки и/или дату рейса и по ним осуществлять поиск рейсов;
5. покупка билета: возможность приобретения билета на конкретный рейс;
6. просмотр купленных билетов: возможность просматривать все купленные билеты и данные по ним;
7. редактирование маршрута и рейса: возможность изменять информацию по маршруту и рейсу;
8. удаление маршрута и рейса: возможность удалять существующий маршрут и рейс;
9. добавление маршрута и рейса: возможность добавления маршрута и рейса со всеми соответствующими ему данными;
10. разделение уровня доступа: у информационной системы существует 3 уровня доступа: администратор, пользователь, гость, которые обладают возможностями, указанными в таблице 1;

Таблица 1 - Разделение уровня доступа

Возможность	Гость	Пользователь	Администратор
Регистрация	✓	✓	✓
Авторизация	✓	✓	✓
Просмотр рейсов	✓	✓	✓
Поиск рейсов	✓	✓	✓
Покупка билета		✓	✓
Просмотр купленных билетов		✓	✓
Редактирование маршрута и рейса			✓
Удаление маршрута и рейса			✓
Добавление маршрута и рейса			✓

Информационная система должна соответствовать следующим критериям:

- 1) интуитивно понятный интерфейс;
- 2) система должна функционировать в ОС семейства Windows, как минимум на Windows 10;
- 3) соответствие кода принципам объектно-ориентированного программирования;
- 4) шифрование пароля;
- 5) использования JWT для регистрации и авторизации пользователей;
- 6) формирование системой билета в виде документа с расширением pdf.

Были сформированы требования к тестированию:

- 1) провести интеграционное тестирование в целях проверки корректности реализации функций, требуемых техническим заданием [2];
- 2) провести модульное тестирование, которое заключается в тестировании отдельных методов и функций классов, компонентов или модулей, используемых в приложении [1];
- 3) провести первичное нагрузочное тестирование, которое заключается в определении реалистичной нагрузки путём отправки за определённое количество времени 100 запросов и последовательно увеличивая запросы за это же количество времени [4].

2 Проектирование и реализация базы данных «Касса автовокзала»

2.1 ER – диаграмма

Для создания ER-диаграммы проектируемой системы "Касса автовокзала" на основе сформулированной предметной области, используется нотация Мартина. Данная нотация является одной из наиболее известных в разработке баз данных, отражающей уровень логического представления базы данных с обозначением некоторых компонентов модели базы данных в графическом виде, облегчая отображение диаграммы в рабочем пространстве [5].

Модели такого типа менее громоздки по сравнению с моделями в нотации Питера Чена. Тем не менее, сложность предметных областей нередко мешает представлению всей модели в рамках единого рабочего пространства, что во многих средствах моделирования баз данных исправляется возможностью формирования и представления моделей базы данных в разрезе отдельных рабочих пространств, которые могут соответствовать функциональному делению предметной области или какому-либо другому фактору, уменьшающему количество рассматриваемых элементов модели базы данных.

Основу всей модели базы данных в нотации Мартина составляют элементы "Сущность". Название сущности представляет собой существительное или словосочетания, обозначающие особенности данных, которые представляются описываемой сущностью. Сущность состоит из атрибутов. Связи изображаются линиями, соединяющими сущности, вид линии в месте соединения с сущностью определяет множественность связи.

Устанавливая связи между сущностями, в нотации Мартина её смысловое наполнение можно обозначать единственной глагольной формой, имеющей смысл связи от "левой" сущности к "правой" сущности или в более общем понятии от связи "один" ко "многим". При связи "многие" ко "многим"

смысловое наполнение обозначается двумя глагольными формами, которые можно прочитать в двух направлениях.

Определим сущности и атрибуты сущностей для предметной области «Касса автовокзала».

Сущность «Пользователь» определяет человека, который желает воспользоваться электронной системой для приобретения билета. Атрибуты сущности «Пользователь»:

- имя;
- фамилия;
- отчество;
- паспорт;
- email;
- пароль;
- уровень доступа.

Сущность «Билет» определяет документ, подтверждающий оплату проезда и право пассажира на проезд в данном автобусе. Атрибуты сущности «Билет»:

- пассажир;
- номер рейса;
- пункт отправления;
- пункт прибытия;
- время отправления;
- время прибытия;
- номер места;
- платформа;
- стоимость.

Сущность «Рейс» определяет законченный транспортный цикл по доставке пассажиров. Атрибуты сущности «Рейс»:

- маршрут;
- автобус;
- время отправления;
- дата отправления.

Сущность «Автобус» определяет транспортное средство, которое перевозит пассажиров. Атрибуты сущности «Автобус»:

- модель автобуса;
- номерной знак.

Сущность «Маршрут» определяет путь следования с указанием начального, конечного и промежуточных пунктов, в случае их наличия, по которым должен ехать автобус, а также номер маршрута и его тариф. Атрибуты сущности «Маршрут»:

- номер маршрута;
- тариф;
- пункты маршрута.

Сущность «Пункт» определяет общественное место остановки транспортных средств по маршруту. Атрибуты сущности «Пункт»:

- наименование;
- тип пункта.

Сущности предметной области «Касса автовокзала» связаны следующим образом:

- сущности «Пользователь» и «Билет» связаны между собой связью «один ко многим»: один пользователь может приобрести несколько билетов, билет может быть приобретён только одним пользователем;

- сущности «Билет» и «Рейс» связаны между собой связью «один ко многим»: несколько билетов может быть на один рейс, в билете может быть указан только один рейс;
- сущности «Билет» и «Пункт» связаны между собой связью «один ко многим»: один пункт может быть указан в нескольких билетах, в билете может быть указан только один пункт отправления и прибытия;
- сущности «Рейс» и «Автобус» связаны между собой связью «один ко многим»: один автобус может участвовать в нескольких рейсах, в одном рейсе может быть только один автобус;
- сущности «Маршрут» и «Пункт» связаны между собой связью «многие ко многим»: в одном маршруте может быть несколько пунктов, один пункт может быть в нескольких маршрутах.

На рисунке 2 показана созданная ER-диаграмма проектируемой системы «Касса автовокзала».



Рисунок 2 - ER-диаграмма системы «Касса автовокзала»

2.2 Логическая модель базы данных

Построим логическую модель базы данных системы «Касса автовокзала», опираясь на ER-диаграмму. На рисунке 3 показана построенная логическая модель базы данных проектируемой системы. Созданная логическая модель базы данных содержит следующие отношения: «Тариф», «Маршрут», «Автобус», «Модель автобуса», «Тип пункта», «Пункт», «Пункт маршрута», «Расстояние и время», «Рейс», «Билет», «Пользователь», «Уровень доступа».

Для разбиения связей многие-ко-многим были введены промежуточные отношения: «Пункт маршрута». Все отношения имеют связи «один ко многим». Все отношения логической модели базы данных находятся в третьей нормальной форме, поскольку каждое отношение модели находится во второй нормальной форме (каждый неключевой атрибут каждого отношения имеет неприводимую функциональную зависимость от соответствующего первичного ключа) и каждый неключевой атрибут каждого отношения не находится в транзитивной функциональной зависимости от соответствующего первичного ключа.

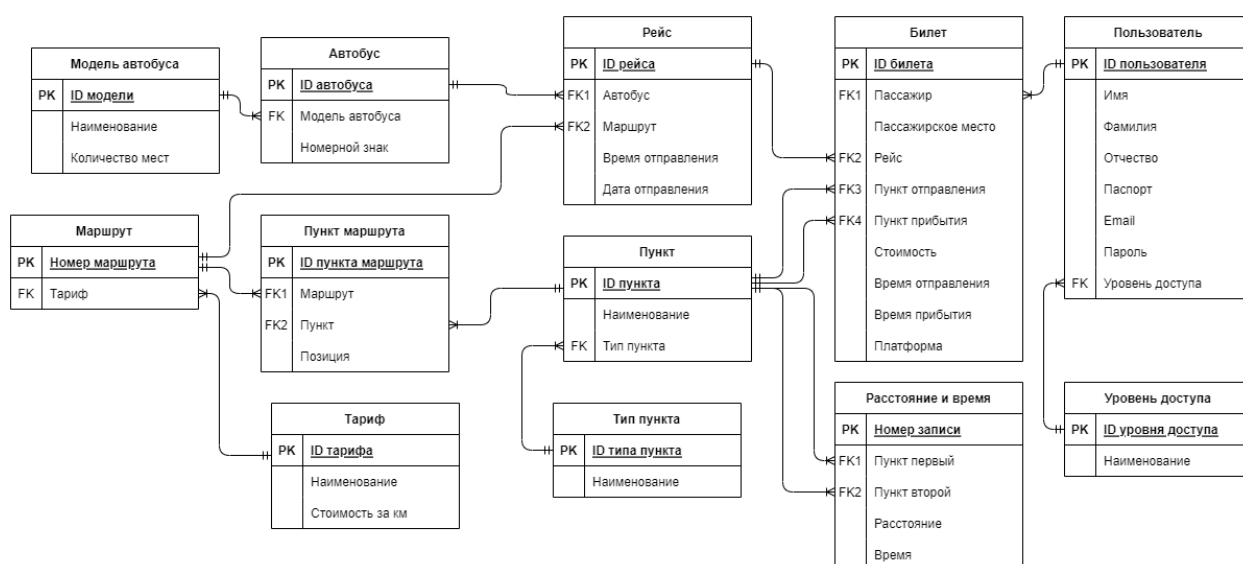


Рисунок 3 - Логическая модель базы данных системы «Касса автовокзала»

2.3 Нормализация реляционных отношений

Метод семантического моделирования, который использовался при построении логической модели данных, состоит в осмыслении искомой предметной области. Получившаяся в результате этого логическая модель, является слабо формализованной, из-за чего проектируемая база данных, может быть, некорректной и неоптимальной вследствие избыточности данных [3].

При работе с отношениями, в которых присутствует избыточность данных, могут возникнуть аномалии:

- 1) аномалия обновления;
- 2) аномалия включения;
- 3) аномалия удаления.

Для устранения аномалий, следует воспользоваться теорией нормализации. Теория нормализации, в отличие от метода семантического моделирования, является строго формализованной.

В теории нормализации все реляционные отношения разбиваются на категории, и каждая такая категория объявляется нормальной формой. Процесс проектирования БД с использованием метода нормализации заключается в том, что исходное отношение путем декомпозиции переводят из младшей нормальной формы в более высокую нормальную форму. Если база данных находится хотя бы в третьей нормальной форме, ее можно считать нормализованной.

Для реализуемой системы достаточно привести все отношения к 3 нормальной форме. То есть проверить для 1 НФ, является ли все атрибуты атомарными. Далее необходимо привести отношение ко второй и третьей нормальной форме.

Отношение находится в 2НФ, если оно находится в 1НФ и каждый неключевой атрибут функционально полно зависит от его потенциального ключа. Функциональная полнота означает, что если потенциальный ключ

является составным, то простые атрибуты зависят от всего ключа, а не от его отдельных компонентов.

Отношение находится в 3Нф, если оно находится в 2Нф и нет транзитивных функциональных зависимостей неключевых атрибутов от ключевых.

Для примера, рассмотрим отношение «Пользователь» (Рисунок 4).

Пользователь	
	Имя
	Фамилия
	Отчество
	Паспорт
	Email
	Пароль
	Уровень доступа

Рисунок 4 - Отношение «Пользователь»

Все атрибуты рассматриваемого отношения скалярны, следовательно, отношение «Пользователь» находится в первой нормальной форме.

После чего проверяем на 2 НФ и 3 НФ. Потенциальным ключом является «Email» и «Паспорт», но в данном случае, в качестве первичного ключа, лучшим выбором будет использование искусственного ключа «ID пользователя», можно аргументировать выбор следующими причинами:

- 1) в случае, когда «Email» или «Паспорт» может быть изменен, использование их в качестве первичного ключа может быть проблематичным, то есть их изменение может потребовать обновления всех связанных записей и внешних ключей, что может

быть трудоемким и затратным процессом, в таких случаях использование искусственного ключа может облегчить управление и поддержку данных, поскольку обновление «Email» или «Паспорт» не повлечет за собой изменение основного ключа;

- 2) искусственный ключ, как правило, имеет простую и фиксированную структуру, что облегчает его использование в индексировании и ускоряет процессы поиска и сортировки данных, он может быть более компактным и эффективным для работы с базой данных, особенно при большом объеме данных;
- 3) искусственный ключ, такой как «ID пользователя», обеспечивает гарантированную уникальность каждой записи в таблице. «Email» или «Паспорт» может быть уникальным, но в некоторых случаях могут возникать ошибки, которые при использовании искусственного ключа исключены.

Таким образом, добавляется атрибут «ID пользователя», который будет являться первичным ключом. Каждый неключевой атрибут неприводимо зависит от потенциального ключа. Следовательно, отношение «Пользователь» находится во второй нормальной форме.

Также каждый неключевой атрибут отношения нетранзитивно зависит от потенциального ключа. То есть, неключевые атрибуты не зависят друг от друга, но зависят от потенциального ключа. Значит, отношение находится в третьей нормальной форме. (Рисунок 5).

Пользователь	
РК	<u>ID пользователя</u>
	Имя
	Фамилия
	Отчество
	Паспорт
	Email
	Пароль
	Уровень доступа

Рисунок 5 - Отношение «Автобус» в 3 НФ

Аналогичным образом была проведена проверка остальных отношений. Все отношения находятся в третьей нормальной форме, следовательно модель базы данных является нормализованной.

2.4 Обоснование выбора СУБД

Для реализации проектируемой системы была выбрана СУБД PostgreSQL. PostgreSQL является свободной объектно-реляционной системой управления базами данных, которая широко используется на множестве платформ, включая UNIX-подобные системы (AIX, BSD, HP-UX, IRIX, Linux, macOS, Solaris/OpenSolaris, Tru64, QNX) и Microsoft Windows. Ее выбор обоснован рядом преимуществ и характеристик, которые предоставляет PostgreSQL.

Одним из достоинств PostgreSQL является его мультиплатформенность, что позволяет разворачивать и использовать систему на различных операционных системах. Кроме того, PostgreSQL использует многопоточность и быструю систему выделения памяти на основе потоков, что способствует повышению производительности и эффективности работы.

PostgreSQL также обладает богатым набором разнообразных типов данных, что обеспечивает гибкость при моделировании и хранении информации. Более того, система поддерживает функцию разделения прав на основе ролей и паролей, обеспечивая контроль доступа к данным в соответствии с требованиями безопасности проекта.

Кроме того, PostgreSQL предоставляет интерфейсы API для множества популярных языков программирования, таких как C, C++, PHP, Perl, Java и другие. Это облегчает интеграцию СУБД с различными компонентами системы и разработку приложений на различных платформах.

Построим физическую модель базы данных проектируемой системы для СУБД PostgreSQL с использованием приложения pgModeler. Построенная модель показана на рисунке 6.

Рисунок 6 – Физическая модель базы данных системы «Касса автовокзала»

3 Проектирование и реализация информационной системы «Касса автовокзала»

3.1 Архитектура информационной системы

Разрабатываемая информационная система представляет собой веб-приложение, основанное на клиент-серверной архитектуре. Веб-приложение включает клиентскую часть, которая работает в веб-браузере, и серверную часть, которая запускается на веб-сервере.

Клиент-серверная архитектура веб-приложения в данном случае является трехуровневой [6]. Для веб-приложения трехуровневая архитектура обычно включает следующие компоненты:

1) клиентская часть (Presentation Layer):

Отвечает за отображение пользовательского интерфейса в веб-браузере и обработку пользовательского ввода. Клиентская часть веб-приложения взаимодействует с серверной частью через HTTP-запросы и получает от неё данные для отображения пользователю;

2) веб-сервер (Business Logic Layer):

Содержит бизнес-логику приложения, включая обработку запросов, выполнение расчетов, взаимодействие с базой данных и другими внешними сервисами;

3) сервер базы данных (Data Layer):

Хранит и обрабатывает данные приложения. Сервер базы данных может быть реализован с использованием различных СУБД, таких как MySQL, PostgreSQL, Oracle и других.

Комбинация трехуровневой архитектуры и клиент-серверной архитектуры позволяет создавать масштабируемые, модульные и гибкие приложения, где каждый компонент выполняет свою функцию и может быть развернут на отдельном сервере или компьютере.

Разрабатываемая информационная система в основе своей на веб-сервере использует паттерн проектирования MVC (Model-View-Controller). Этот паттерн позволяет эффективно организовать структуру приложения, разделяя его на три основных компонента: модель, представление и контроллер.

Модель представляет собой логику работы с данными и описание фундаментальных данных. В контексте информационной системы, модель отвечает за управление и обработку данных, а также за бизнес-логику приложения. Она предоставляет методы и функциональность, необходимые для доступа к данным, их изменения и обработки.

Представление является частью графического интерфейса приложения, которая получает данные от модели и визуализирует их.

Контроллер является посредником между моделью и представлением. Он отвечает за обработку пользовательского ввода и управление потоком данных между моделью и представлением. Контроллер реагирует на действия пользователя, вызывает соответствующие методы модели для обработки данных и обновляет представление с учетом полученных результатов.

В результате определение MVC для веб-сервера в реализуемой информационной системы можно сформулировать следующим образом:

MVC для веб-сервера представляет собой архитектурный шаблон, где модель, представление и контроллер работают в тесной связи для обработки запросов и предоставления данных клиентской части приложения. Контроллер принимает запросы от клиента, обрабатывает их и взаимодействует с моделью для получения или обновления данных. Затем контроллер выбирает соответствующее представление, которое использует полученные данные для формирования ответа, который будет отправлен обратно клиенту.

Использование паттерна MVC в разрабатываемой информационной системе имеет ряд преимуществ. Во-первых, он способствует разделению логики приложения от визуальной части, что упрощает поддержку и развитие

системы. Во-вторых, он облегчает расширение функциональности приложения путем модификации отдельных компонентов без влияния на остальные. Также, паттерн MVC повышает переиспользуемость кода, поскольку компоненты могут быть заменены или использованы повторно в других проектах.

Архитектура разрабатываемого веб-приложения показана на рисунке 7.



Рисунок 7 – Архитектура приложения информационной системы «Касса автовокзала»

3.2 Диаграмма вариантов использования

Диаграмма вариантов использования (Use case diagram) является графическим инструментом, который отражает взаимодействие между актерами и функциональными возможностями системы [9]. Она позволяет заказчику, конечному пользователю и разработчику совместно обсуждать поведение системы и ее функциональность. Диаграмма вариантов использования является важным инструментом для моделирования системы, и ее использование позволяет достичь следующих целей:

- 1) разработка начальной концептуальной модели системы:

Диаграмма вариантов использования помогает создать общее представление о функциональности системы и ее взаимодействии с актерами. Это предоставляет основу для дальнейшей детализации модели системы в форме логических и физических моделей;

- 2) определение общих границ функциональности системы:

Диаграмма вариантов использования помогает определить, какие функции и возможности должны быть включены в систему, и какие актеры будут взаимодействовать с этими функциями. Она помогает установить общие границы функциональности проектируемой системы;

3) спецификация требований к функциям системы:

Диаграмма вариантов использования позволяет конкретизировать требования к функциям системы в форме вариантов использования. Каждый вариант использования представляет собой сценарий, описывающий, как актеры взаимодействуют с системой для достижения определенных целей. Это помогает команде разработчиков лучше понять требования к системе и создать соответствующий функционал.

На рисунке 8 показана диаграмма вариантов использования системы «Касса автовокзала».

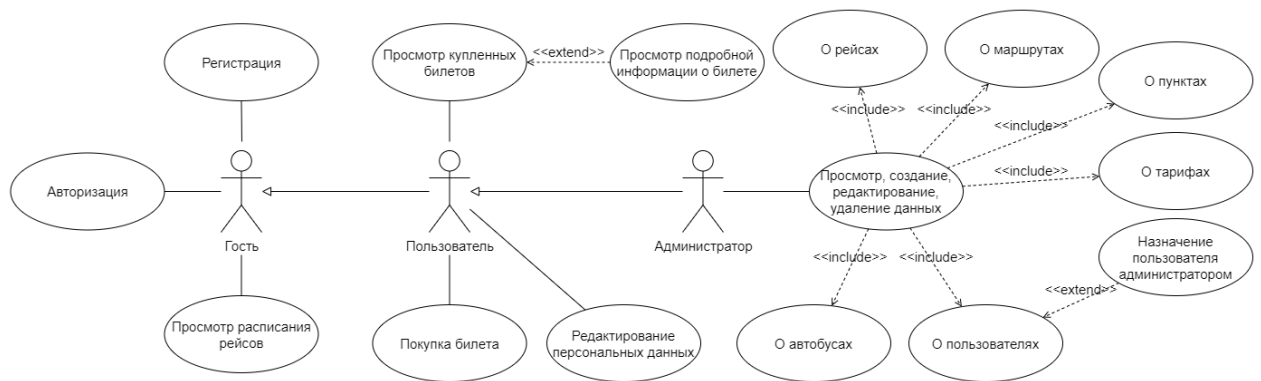


Рисунок 8 – Диаграмма вариантов использования информационной системы «Касса автовокзала»

Проектируемую систему могут использовать 3 различных актёра: «Гость», «Пользователь», «Администратор».

Актеру «Гость» доступны варианты использования: «Регистрация». «Авторизация». «Просмотр расписания рейсов».

Актеру «Пользователь» доступны варианты использования актёра «Гость», а также: «Покупка билета», «Редактирование персональных данных», «Просмотр купленных билетов», «Просмотр подробной информации о билете.

Актеру «Администратор» доступны варианты использования актёра «Пользователь», а также: «Просмотр, создания, редактирование, удаление данных о рейсах, маршрутах, пунктах, тарифах, пользователях, автобусах», включая «Назначение пользователя администратором».

3.3 Диаграмма последовательности

Диаграмма последовательности (Sequence diagram) является одной из диаграмм взаимодействия UML, используемых для моделирования динамических аспектов системы [10]. Она подчеркивает временной порядок сообщений и представляется в виде таблицы, где объекты размещены вдоль оси X, а сообщения упорядочены по ходу времени вдоль оси Y. Диаграмма последовательности позволяет описать поведение системы на уровне отдельных объектов, которые обмениваются сообщениями между собой для достижения определенной цели или реализации определенного варианта использования.

Основные особенности диаграммы последовательности:

1) описание последовательности сообщений:

Диаграмма последовательности позволяет представить последовательность сообщений, которые передаются между объектами системы. Каждое сообщение указывает отправителя, получателя и тип сообщения. Это помогает визуализировать взаимодействие между объектами и последовательность выполнения операций;

2) изображение объектов и жизненных циклов:

На диаграмме последовательности объекты системы представлены в виде вертикальных линий, называемых экземплярами объектов. Жизненные циклы объектов отображаются в виде прямоугольников или пунктирных линий, показывающих активность объекта в определенный момент времени;

3) учет условий и ветвлений:

Диаграмма последовательности также может включать условия и ветвления во время выполнения. Это позволяет описать различные сценарии выполнения и учесть разные варианты поведения системы в зависимости от условий.

Диаграмма последовательности на рисунке 9 изображает взаимодействие объектов и последовательность сообщений для варианта использования системы «Процесс покупки билета».

Важно, что на диаграмме присутствует платёжный сервис, который обязательно должен был присутствовать в разрабатываемой системе для полного функционирования, но так как разрабатывается учебный проект, то интеграция с такими системами как платёжный сервис опускается, потому что это не является целью работы.

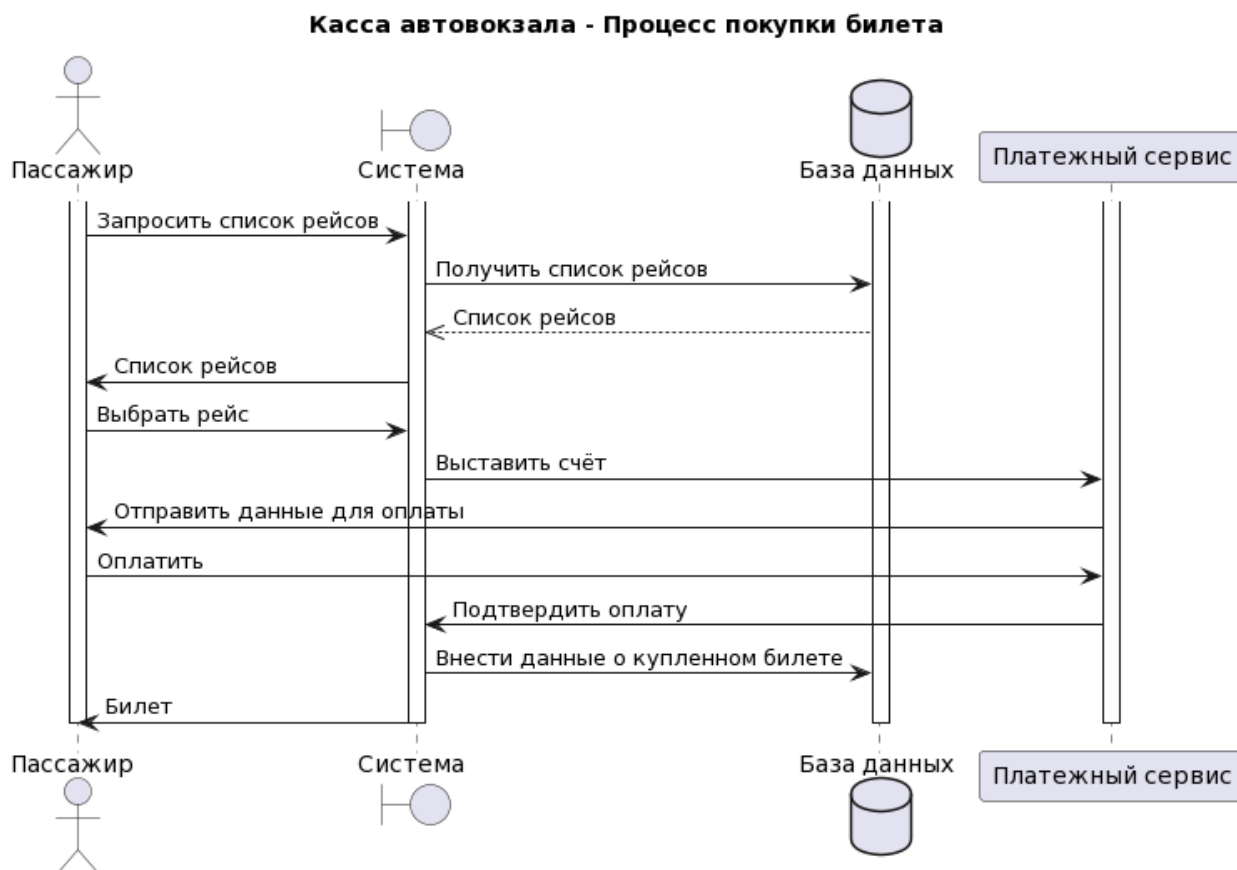


Рисунок 9 - Диаграмма последовательности для варианта использования «Процесс покупки билета»

3.4 Выбор средств реализации

Для разработки клиентской части информационной системы были выбраны следующие технологии: HTML, CSS, JavaScript, EJS и Axios.

HTML является языком гипертекстовой разметки и используется для создания структуры и контента веб-страниц. CSS отвечает за оформление и внешний вид веб-страниц, а JavaScript обеспечивает интерактивность и динамическое поведение страницы [8]. EJS (Embedded JavaScript) — это шаблонизатор, позволяющий встраивать JavaScript код в HTML для создания динамических страниц. Axios - это библиотека JavaScript для выполнения HTTP-запросов с использованием промисов, облегчающая взаимодействие с сервером и обмен данными.

Для серверной части информационной системы были выбраны следующие технологии: Node.js и Express.js. Node.js - это среда выполнения JavaScript, позволяющая запускать JavaScript код на стороне сервера [7]. Express.js - это минималистичный фреймворк для создания веб-приложений на Node.js, предоставляющий базовые возможности для разработки веб-серверов.

В качестве базы данных для проектируемой информационной системы была выбрана PostgreSQL. PostgreSQL - это мощная реляционная база данных с открытым исходным кодом, обладающая полным набором команд SQL и поддерживающая широкий спектр функций для хранения и обработки данных.

Для обмена данными между клиентской и серверной частями информационной системы используется библиотека Axios.

Для разработки и создания локального сервера была выбрана среда разработки WebStorm. WebStorm - это интегрированная среда разработки (IDE), предоставляющая удобные инструменты для разработки веб-приложений с поддержкой JavaScript, HTML, CSS и других технологий.

3.5 Реализация серверной части

Серверная часть - это компонент архитектуры веб-приложения, который обрабатывает запросы от клиентов, выполняет бизнес-логику и взаимодействует с базой данных или другими внешними сервисами.

В рамках серверной части происходит обработка HTTP-запросов, поступающих от клиентов. Для примера рассмотрим GET-запрос следующего формата: «/». Этот запрос должен вернуть основное представление, что является главной страницей приложения.

Для работы сервера требуется установка и настройка минимального набора инструментов, таких как Node.js и Express.js. Ниже приведен пример настройки:

```
const app = express();  
const PORT = process.env.PORT || 8080;
```

```
app.listen(PORT, () => console.log(`Сервер запущен на порту ${PORT}`));
```

Здесь создаётся express-приложение, указывается порт для сервера и запускается сервер на указанном порту.

Затем необходимо определить маршрут для HTTP-запроса и указать, какой контроллер будет обрабатывать этот запрос. Так как приложение разрабатывается в модульном формате, также необходимо подключить модули соответствующих контроллеров:

```
const mainController = require("../controller/main.controller");  
app.get('/', mainController.getMain);
```

В данном коде подключается контроллер и указывается, что по HTTP-запросу «/» вызывается метод у контроллера mainController, а именно getMain.

Метод getMain выглядит следующим образом:

```
async getMain(req, res) {  
  const role = req.user.role;  
  let name, patronymic;  
  if (role !== 'Гость') {  
    const userData = await user.getName(req.user.user_id);  
    name = userData.name;  
    patronymic = userData.patronymic;  
  }  
  const template = path.join(__dirname, '..', 'views', 'main', 'main.ejs');  
  const html = await ejs.renderFile(template, { name, patronymic, role });  
  
  res.send(html);  
}
```

Как можно видеть, этот метод контроллера получает роль пользователя из запроса и если пользователь не является гостем, то вызывается метод «getName» у модели «user» для получения имени и отчества из базы данных по «user_id», который передаётся как параметр.

Тут важно отметить, что контроллер не взаимодействует с базой данных, всю работу проделывает модель, как было указано в архитектуре приложения.

После контроллер передаёт имя, отчество, роль пользователя в представление, которое в зависимости от этих данных меняет своё содержимое. И в конечном итоге готовое представление отправляется клиенту в ответ на его запрос.

Тут важно отметить, что представление не взаимодействует напрямую с моделью, а получает данные от контроллера, который является посредником.

Также рассмотрим задействованный в контроллере метод «getName» у модели «user»:

```
async getName(id) {  
    const find = await pool.query('SELECT first_name AS "name",  
patronymic FROM "user" WHERE user_id = $1;', [id]);  
    return find.rows[0];  
}
```

Данный метод модели осуществляет запрос в СУБД PostgreSQL и возвращает имя и отчество по указанному id пользователя в таблице «user». После чего возвращает эти данные. Все методы у модели «user» указаны в ПРИЛОЖЕНИИ А.1.

Результат выполнения рассмотренного HTTP-запроса показан на рисунке 10 для пользователя с ролью «Гость» и на рисунке 11 для пользователя с ролью «Администратор».

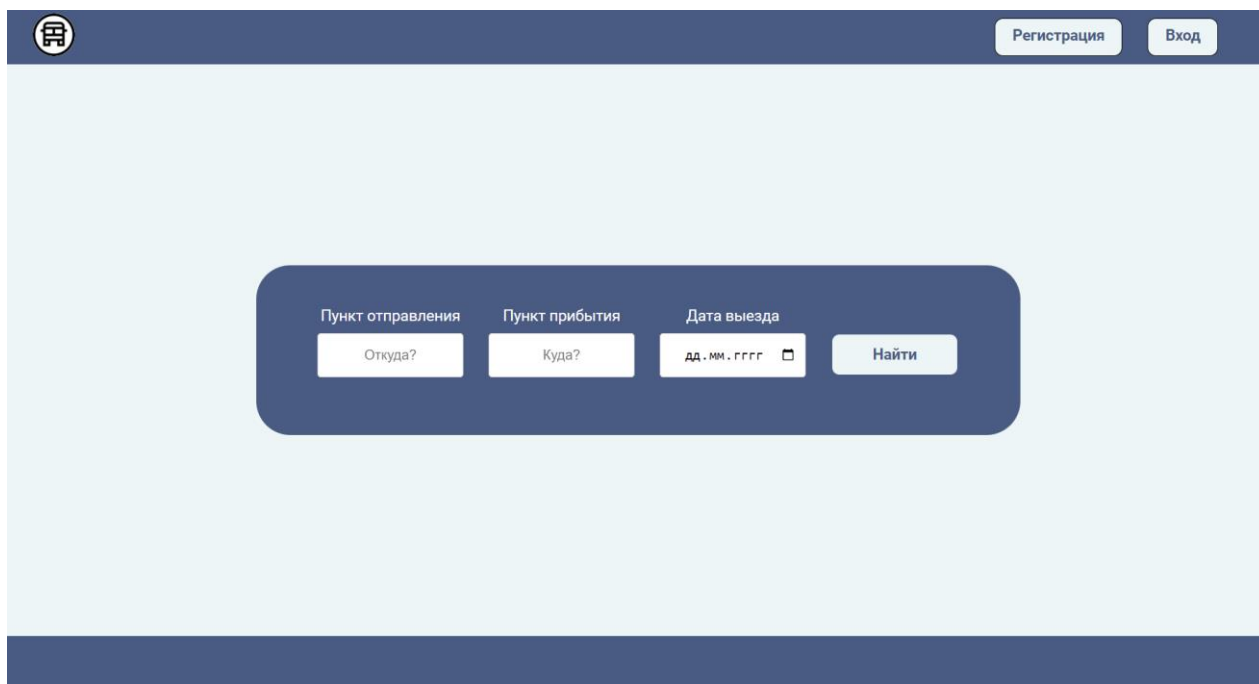


Рисунок 10 – Результат выполнения HTTP-запроса «/» для пользователя с ролью «Гость»

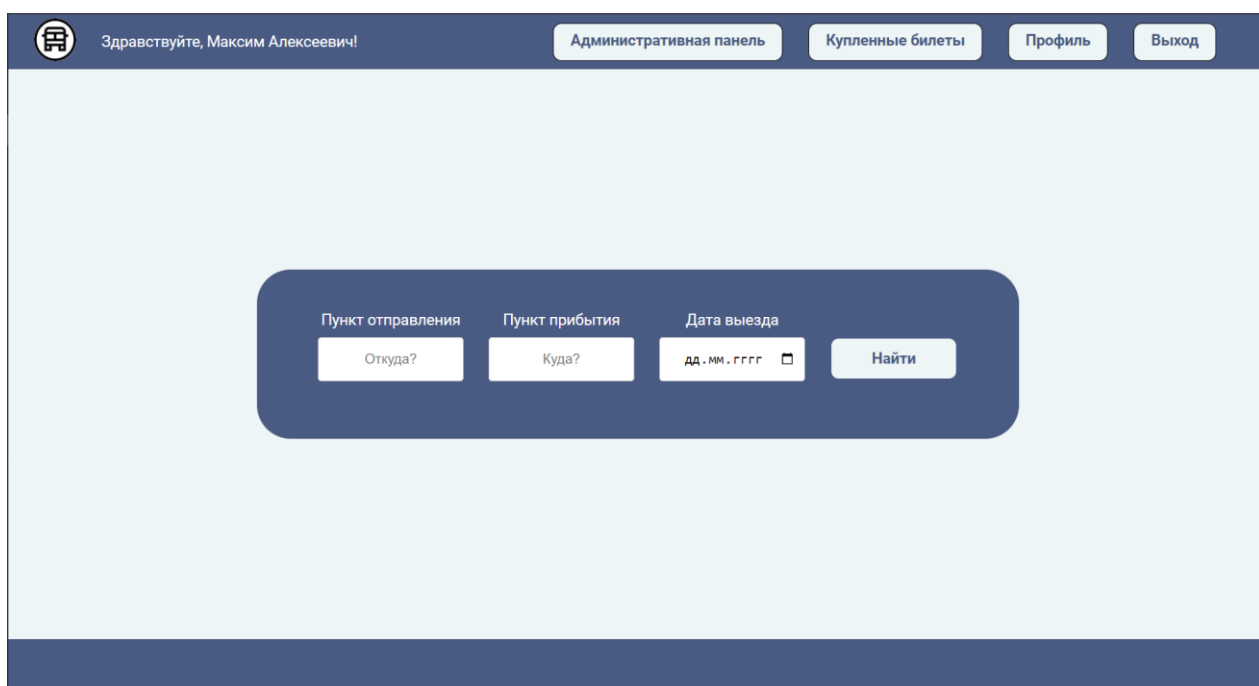


Рисунок 11 – Результат выполнения HTTP-запроса «/» для пользователя с ролью «Администратор»

Серверная часть приложения построена на принципе обработки HTTP-запросов от пользователей с использованием маршрутизатора, который выбирает соответствующий контроллер для обработки каждого запроса. Контроллер взаимодействует с моделью для получения необходимых данных

и передает их в представление. Представление, в свою очередь, формирует окончательный HTML-код в зависимости от полученных данных, который контроллер отправляет обратно клиенту в ответ на его запрос.

В результате были реализованы следующие функциональности: регистрация пользователей, авторизация, редактирование личных данных, поиск рейсов по заданным параметрам, отображение билетов и их покупка, а также административная панель. Административная панель предоставляет возможности удаления, добавления и редактирования информации о маршрутах, пунктах, автобусах, тарифах, рейсах и пользователях.

Таким образом, реализованные функциональности позволяют пользователям взаимодействовать с приложением, выполнять поиск рейсов, просматривать информацию о билетах, а администраторам предоставляется управление различными аспектами системы.

3.6 Реализация клиентской части

Клиентская часть разрабатываемого веб-приложения включает HTML, CSS и JavaScript, которые открываются в браузере для отображения веб-страниц и управления их поведением при взаимодействии пользователя со страницей.

Для генерации динамических веб-страниц на стороне сервера используется шаблонизатор EJS (Embedded JavaScript). EJS позволяет встраивать JavaScript-код в HTML-шаблоны для создания содержимого на сервере.

В контексте серверной части, представления (views) содержат HTML-код, который генерируется с использованием EJS. Одно и то же представление может генерировать разный HTML-код в зависимости от различных условий или данных, полученных с сервера.

Из рисунков 10 и 11 видно, что в зависимости от разной роли, получается разный HTML-код от одного и того же представления. Рассмотрим

этот случай, путём анализа части кода из представления «main», где происходит условная проверка роли пользователя:

```
<% if (role !== 'Гость') { %>
    <p class="hello">Здравствуйте, <%= name.charAt(0).toUpperCase() +
name.substr(1).toLowerCase() + ' ' +
    patronymic.charAt(0).toUpperCase() +
patronymic.substr(1).toLowerCase()%>!</p>
<% } %>
```

В данном примере, с использованием конструкции `<% %>` в EJS, можно встраивать JavaScript-код для выполнения условных проверок. Внутри конструкции `<%= %>` передается значение переменных, которые были переданы в представление контроллером.

В результате, если роль пользователя не является "Гость", то в HTML-коде, отправляемом клиенту, будет отображаться приветствие пользователя по имени и отчеству. Если же роль пользователя "Гость", то соответствующее приветствие не будет отображаться, что является логичным, так как для гостей нет информации о пользователе.

Полный код шаблона главной страницы указан в ПРИЛОЖЕНИИ А.3.

Внутри блока кода в примере выше используется элемент `<p>` с атрибутом `class="hello"`. Атрибут `class` указывает на использование класса CSS для этого элемента. В данном случае, класс `hello` используется для стилизации элемента и применения определенных стилей к нему.

CSS-код, который связан с классом "hello", определяет внешний вид элемента:

```
.hello {
    font-size: 18px;
    color: white;
    margin-top: 10px;
}
```


В этом примере определены стили для класса "hello". Шрифт будет иметь размер 18 пикселей, цвет текста будет белым, и будет установлен верхний отступ в 10 пикселей.

CSS позволяет определить множество других свойств, таких как фон, отступы, границы, выравнивание и многое другое, которые могут быть применены к элементам с помощью классов или других селекторов. Это позволяет контролировать стиль и внешний вид элементов на веб-странице, чтобы создать желаемый пользовательский опыт.

В разрабатываемом веб-приложении используется комбинация HTML, CSS и JavaScript для достижения дополнительных визуальных эффектов и управления поведением страницы. Одним из примеров использования JavaScript является библиотека Axios, которая позволяет отправлять асинхронные запросы на сервер без перезагрузки страницы. Такой подход позволяет обновлять содержимое страницы динамически и обрабатывать данные, не требуя полной перезагрузки.

JavaScript также предоставляет возможность реализации различных визуальных эффектов, анимаций и интерактивности на странице, которые недоступны с использованием только CSS. Например, с помощью JavaScript были созданы плавные анимированные переходы между состояниями элементов, динамически изменяется содержимое страницы в ответ на действия пользователя и многое другое.

Комбинация HTML, CSS, JavaScript и EJS в клиентской части приложения создает синергию, которая позволяет разработать интерактивные и привлекательные пользовательские интерфейсы. Этот подход обеспечивает удобство использования и предоставляет богатый пользовательский опыт, позволяя взаимодействовать с контентом страницы, получать актуальную информацию и наслаждаться эффектными визуальными элементами.

В результате разработки были созданы следующие страницы: страница регистрации, страница авторизации, главная страница с функционалом поиска

рейсов и покупки билетов, а также административная панель, предоставляющая полную информацию и контроль над системой.

Страница регистрации позволяет пользователям создавать новые учетные записи, вводя необходимую информацию (Рисунок 12). Страница авторизации обеспечивает аутентификацию зарегистрированных пользователей, позволяя им получить доступ к персональным данным и функциям приложения (Рисунок 13).

Регистрация

Имя
Марсель

Фамилия
Жуков

Отчество
Сергеевич

Email
zhukov@mail.ru

Серия и номер паспорта
0123 456789

Пароль
...

Пароль
...

Регистрация

Рисунок 12 – Страница регистрации

Вход

Email
zhukov@mail.ru

Пароль
...

Вход

Рисунок 13 – Страница авторизации

Главная страница является центральным местом для поиска рейсов и покупки билетов. Здесь пользователи могут указывать параметры поиска, такие как место отправления и прибытия, дату, чтобы найти подходящие рейсы. После выбора рейса пользователи могут просмотреть информацию о билетах и совершить покупку (Рисунок 14).

Здравствуйте, Марсель Сергеевич! Купленные билеты Профиль Выход

Пункт отправления Пункт прибытия Дата выезда

Самара Тольятти 01.06.2023 Найти

08:00 1 июня 2 ч 10:00 1 июня
Самара ЦАВ Тольятти АВ Выбрать место

Выберите пассажирское место: 8 Стоимость билета: 990 руб. Купить

22:00 1 июня 2 ч 00:00 2 июня
Самара ЦАВ Тольятти АВ Выбрать место

Рисунок 14 – Главная страница

Административная панель предоставляет полный обзор и управление всей информацией в системе. Администраторы могут просматривать, создавать, удалять и редактировать данные о маршрутах, пунктах, автобусах, рейсах и пользователях, обеспечивая эффективное управление функциональностью и настройками приложения (Рисунок 15).

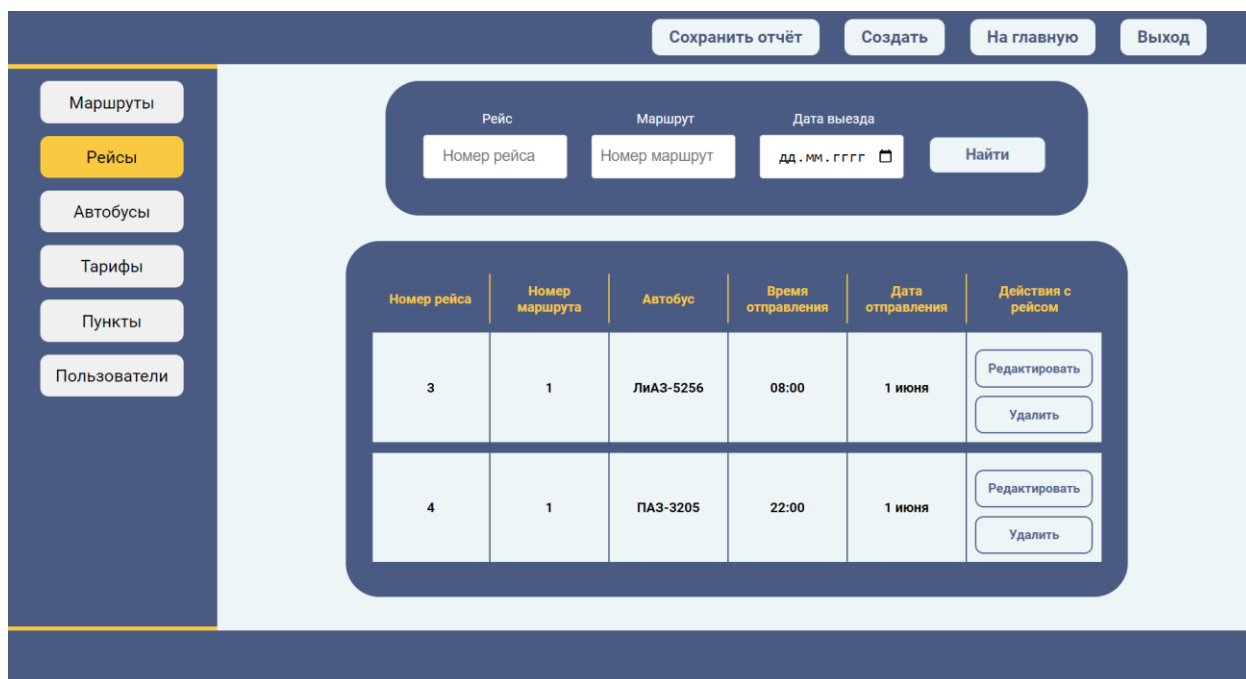


Рисунок 15 – Административная панель

Таким образом, разработанные страницы и функциональности позволяют пользователям взаимодействовать с приложением, находить нужные рейсы, приобретать билеты, а администраторам предоставляется полный контроль над системой, обеспечивая ее эффективное функционирование и удовлетворение потребностей пользователей.

3.7 Тестирование информационной системы

3.7.1 Модульное тестирование

Модульное тестирование (Unit Testing) представляет собой важный тип тестирования программного обеспечения, который фокусируется на проверке отдельных модулей или компонентов программного кода. Его основная цель состоит в том, чтобы убедиться, что каждый компонент программы

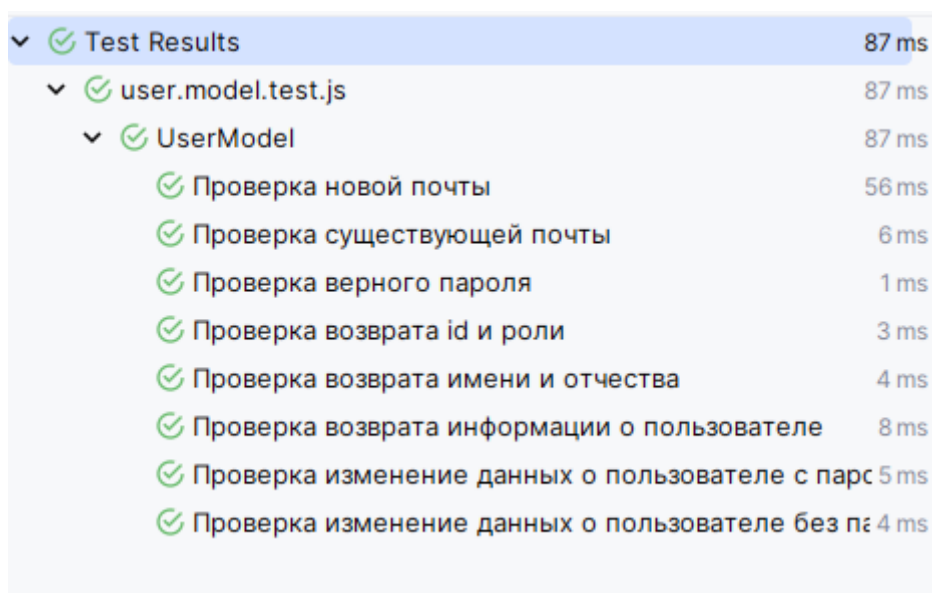
функционирует должным образом. Модульные тесты разрабатываются и выполняются разработчиками на этапе написания приложения, и они направлены на изоляцию и проверку работоспособности конкретных частей кода, таких как функции, методы, процедуры, модули или объекты.

Для проведения модульного тестирования разработанной информационной системы была использована библиотека Jest. На рисунке 16 представлены результаты тестирования функций модели «user». Эти тесты позволяют убедиться в правильной работе функционала модели при работе с данными пользователей.

Пример кода первого теста:

```
it('Проверка новой почты', async () => {  
  const result = await UserModel.checkEmail('nonexisting@example.com');  
  expect(result).toBe(false);  
});
```

В этом тесте проверяется функция проверки новой почты в базе данных, если почта не будет найдена в базе данных, то функция вернёт false, что будет успешным прохождением теста. Тесты были написаны на каждую функцию в модели «user», что показывает покрытие на рисунке 17. Полный список тестов указан в ПРИЛОЖЕНИИ А.2.



Test Results	87 ms
user.model.test.js	87 ms
UserModel	87 ms
Проверка новой почты	56 ms
Проверка существующей почты	6 ms
Проверка верного пароля	1 ms
Проверка возврата id и роли	3 ms
Проверка возврата имени и отчества	4 ms
Проверка возврата информации о пользователе	8 ms
Проверка изменение данных о пользователе с парс	5 ms
Проверка изменение данных о пользователе без парс	4 ms

Рисунок 16 – Результаты тестирования модели «user»

File		Statements		Branches		Functions		Lines	
user.model.js	<div></div>	100%	20/20	100%	2/2	100%	7/7	100%	19/19

Рисунок 17 – Покрытие тестирования модели «user»

Таким образом, модульное тестирование с использованием библиотеки Jest играет важную роль в разработке информационной системы и повышении надежности ее компонентов. Однако, для полноценного тестирования программного обеспечения необходимо также применять интеграционное тестирование.

3.7.2 Интеграционное тестирование

Интеграционное тестирование играет важную роль в разработке программного обеспечения. Оно позволяет проверить взаимодействие между различными компонентами системы и удостовериться в их корректной работе вместе. Было рассмотрено использование инструмента Postman для проведения интеграционного тестирования.

Целью данного интеграционного тестирования с использованием Postman было проверить работоспособность API приложения и удостовериться в корректном взаимодействии его компонентов.

Однако, при проведении интеграционного тестирования вручную возникают некоторые ограничения. В случае расширения системы или изменения ее компонентов, ручное тестирование может стать трудоемким и времязатратным процессом. Кроме того, повторное выполнение большого количества интеграционных тестов может быть подвержено человеческим ошибкам.

В свете этих факторов, переход на автоматизированное тестирование становится важным шагом. Автоматизация интеграционного тестирования с помощью инструментов, таких как Postman, позволяет снизить ручной труд,

ускорить процесс тестирования и повысить его надежность. Автоматизированные тесты могут быть легко настроены для повторного выполнения и интеграции в процесс разработки, что сокращает время и ресурсы, затрачиваемые на тестирование.

Были написаны тесты на проверку API и корректного ответа от сервера для главной страницы системы, а также для модуля авторизации (Рисунок 18).

GET Главная страница localhost:8080/	200 OK 12 ms 2.297 KB
PASS Проверка успешного статуса ответа	
PASS Проверка наличия блока поиска на странице	
PASS Проверка наличия кнопки войти для перехода на авторизацию	
POST Авторизация в системе http://localhost:8080/auth/login	200 OK 89 ms 520 B
PASS Проверка успешного статуса ответа	
PASS Проверка наличия поля 'токен' в ответе на верные данные	
PASS Проверка наличия ответного сообщения 'Пользователь успешно авторизован'	
POST Ошибочная авторизация http://localhost:8080/auth/login	200 OK 5 ms 319 B
PASS Проверка успешного статуса ответа	
PASS Проверка наличия поля 'токен' в ответе на неверные данные	
PASS Проверка наличия ответного сообщения 'Пользователь с таким email не существует'	

Рисунок 18 – Результаты интеграционного тестирования

Пример тестов для запроса показаны на рисунке 19.



Рисунок 19 – Пример тестов для запроса

Также были написаны тесты на проверку API и корректного ответа от сервера для административной панели системы (Рисунок 20).

		1
▼ GET	Административная панель: маршруты	5 0
	Pass Проверка успешного статуса ответа	
	Pass Проверка наличия активной кнопки 'Ма...	
	Pass Проверка наличия блока поиска	
	Pass Проверка наличия таблицы с данными	
	Pass Проверка наличия кнопки 'Сохранить от...	
▼ GET	Административная панель: рейсы	5 0
	Pass Проверка успешного статуса ответа	
	Pass Проверка наличия активной кнопки 'Рей...	
	Pass Проверка наличия блока поиска	
	Pass Проверка наличия таблицы с данными	
	Pass Проверка наличия кнопки 'Сохранить от...	
▼ GET	Административная панель: автобусы	5 0
	Pass Проверка успешного статуса ответа	
	Pass Проверка наличия активной кнопки 'Авт...	
	Pass Проверка наличия блока поиска	
	Pass Проверка наличия таблицы с данными	
	Pass Проверка наличия кнопки 'Сохранить от...	
▼ GET	Административная панель: тарифы	5 0
	Pass Проверка успешного статуса ответа	
	Pass Проверка наличия активной кнопки 'Тар...	
	Pass Проверка наличия блока поиска	

Рисунок 20 – Результаты интеграционного тестирования

В сумме было написано 121 тест (Рисунок 21).

Source	Environment	Iterations	Duration	All tests	Avg. Resp. Time
Runner	none	1	5s 665ms	121	12 ms

Рисунок 21 – Статистика интеграционного тестирования в инструменте
Postman

В результате тестирования были обнаружены следующие ошибки:

- 1) при покупке билета гостем, билет не покупался, но сообщение об этом не показывалось. Решение проблемы: передача сообщения «Билет могут приобретать только авторизированные пользователи» с сервера на клиент и отображение его в форме отображения сообщений;
- 2) результаты поиска рейсов выводились некорректно из-за разных регистров и зависимости от расположения слов в запросе. Решение проблемы: при выполнении поиска все символы приводятся к нижнему регистру, а поиск осуществляется на основе вхождения искомого запроса, независимо от его расположения.

Эти решения позволяют улучшить функциональность системы, обеспечивая более точное и понятное взаимодействие с пользователем.

Интеграционное тестирование играет важную роль в разработке программного обеспечения, позволяя проверить взаимодействие между компонентами системы и удостовериться в их корректной работе вместе. Однако, для обеспечения стабильной и высокопроизводительной работы приложения необходимо дополнить интеграционное тестирование нагрузочным тестированием.

Нагрузочное тестирование важно после интеграционного, так как оно позволяет оценить, как система будет вести себя при реальных условиях эксплуатации и при высоких нагрузках.

3.7.3 Нагрузочное тестирование

Нагрузочное тестирование является важной частью проверки производительности и стабильности разработанного программного обеспечения. Его основной целью является оценка поведения системы при реальных условиях использования и определение предельных нагрузок, которые система может выдержать.

В начальной фазе нагрузочного тестирования проводится первичное тестирование с целью определения предельных нагрузок системы. Это позволяет выявить точку, на которой система начинает испытывать проблемы с производительностью или неспособна обрабатывать запросы пользователей с требуемой отзывчивостью. В результате первичного нагрузочного тестирования определяются максимальные нагрузки, при которых система продолжает работать стабильно и эффективно.

Для проведения нагрузочного тестирования разработанной информационной системы было использовано приложение Apache JMeter версии 5.5. Была создана нагрузка, состоящая из GET-запросов, отправленных в течение 5 секунд. Результаты теста представлены в таблице 2.

Таблица 2 – Результаты нагрузочного тестирования

Номер теста	Кол-во запросов	Среднее время отклика, мс	Мин. время отклика, мс	Макс. время отклика, мс	Процент ошибок, %
1	100	5	3	16	0
2	500	8	2	93	0
3	1000	6	2	46	0
4	1500	276	7	427	0
5	2000	308	8	512	0
6	2500	1771	8	5035	1.64

Наглядно можно видеть результат нагрузочного тестирования на рисунке 22.

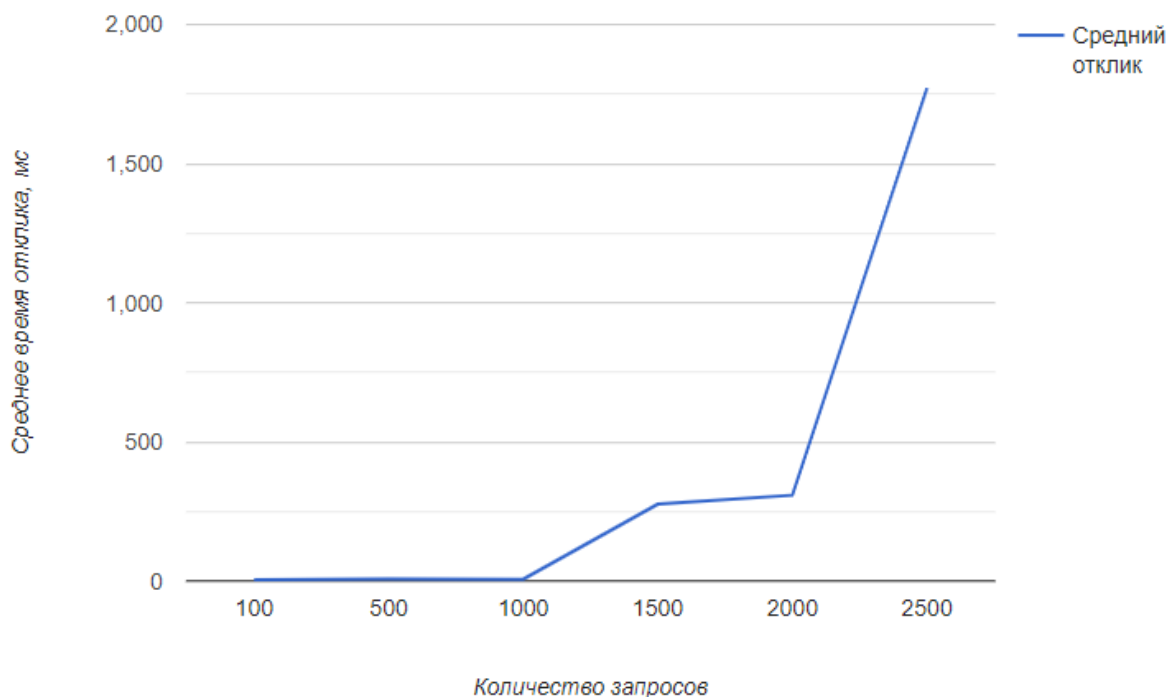


Рисунок 22 – Линейный график нагрузочного тестирования

Анализируя данные, можно сделать следующие выводы:

- 1) среднее время отклика системы до 1500 запросов оставалось в пределах очень быстрого ответа системы, не превышая нескольких миллисекунд. Это говорит о хорошей производительности системы при небольших и средних нагрузках;
- 2) при увеличении количества запросов от 1500 и до 2000 время отклика системы начинает заметно возрастать, достигая нескольких сотен миллисекунд. Это может указывать на наличие узких мест или проблем с масштабируемостью системы при высоких нагрузках;
- 3) в последнем тесте, когда количество запросов составило 2500, было замечено наличие ошибок в виде 1.64% неуспешных запросов. Это может свидетельствовать о достижении предельной границы производительности системы и необходимости оптимизации или масштабирования ее компонентов.

В целом, проведенное нагрузочное тестирование позволило оценить производительность и временные характеристики разработанной информационной системы при различных нагрузках. Полученные результаты могут послужить основой для дальнейшей оптимизации и улучшения производительности системы с целью обеспечения более стабильной и отзывчивой работы при высоких нагрузках.

ЗАКЛЮЧЕНИЕ

В рамках данной работы была разработана информационная система "Касса автовокзала", которая значительно сокращает временные затраты пассажиров автовокзала, повышает их удобство и комфорт пользования автовокзалами, сокращает трафик на автовокзалах, разгружает работников автовокзала.

Система позволяет пользователям регистрироваться и авторизоваться осуществлять поиск по актуальным рейсам и покупать на них билеты, в любой момент иметь доступ ко всем купленным билетам. Администраторам же, в лице работников автовокзала, позволяет удобно редактировать, удалять, добавлять новые маршруты, рейсы, автобусы, тарифы, и пункты. В итоге была получена автоматизированная информационная система.

В ходе выполнения работы были проведены следующие этапы:

- 1) проведён анализ предметной области;
- 2) сформулированы задачи, которые требовалось решить в рамках курсовой работы;
- 3) на основе анализа предметной области была построена информационная и логическая модели базы данных;
- 4) в СУБД PostgreSQL реализована физическая модель базы данных;
- 5) спроектирована информационная система на основе технологии баз данных с трехзвенной архитектурой клиент-сервер и паттерном проектирования MVC;
- 6) с использованием технологий HTML, CSS, JS, EJS, AXIOS.JS, NODE.JS, EXPRESS.JS и многими вспомогательными библиотеками было реализовано клиентское приложение для работы с созданной базой данных;
- 7) реализованы функциональности авторизации и регистрации пользователей с помощью JWT токена, который обеспечивает безопасность передачи информации между клиентским приложением и сервером, а также возможности редактирования, добавления и

просмотра данных о рейсах, маршрутах, автобусах, пунктах, покупка билета, просмотр всех купленных билетов и конкретно одного билета;

- 8) реализовано хеширование паролей пользователей для повышения безопасности личных данных;
- 9) проверка работоспособности системы проводилась на тестовой задаче, в рамках которой в базу данных было введено суммарно 250 кортежей, поведение системы совпало с ожидаемым;
- 10) реализовано формирование системой билета в виде документа с расширением pdf, который может быть просмотрен, сохранен в удобном месте для пользователя или распечатан, если в этом есть необходимость;
- 11) проведено первичное нагрузочное тестирование, которое показало, что система способна максимально обработать различные запросы к базе данных до 2000 виртуальных пользователей в течении 5 секунд, оставаясь при этом работоспособной;
- 12) проведено модульное и автоматизированное интеграционное тестирование, в результате которых были выявлены и исправлены 2 дефекта.

Таким образом, выполнение данной курсовой работы позволило развить навыки постановки задачи, оценки полученных результатов, документирования разрабатываемого ПО, проектирования и реализации программного обеспечения в различных предметных областях, а также проведения тестирования и оценки качества программного обеспечения. Кроме того, были приобретены навыки использования инструментальных средств и языков программирования для разработки программного обеспечения, а также применения этих навыков в различных предметных областях. Это свидетельствует о том, что студент овладел компетенциями,

связанными с разработкой программного обеспечения (ОПК-4, ОПК-9 и ОПК-11).

СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

1. Модульное тестирование [Электронный ресурс] URL: <https://college.arthur-nesterenko.dev/unit-tests> (Дата обращения: 03.04.2023)
2. Теория тестирования ПО просто и понятно [Электронный ресурс] URL: <https://habr.com/ru/articles/587620/> (Дата обращения: 07.04.2023)
3. Кузнецов С.Д. Базы данных. — М. : Издательский центр «Академия», 2012. — 496 с.
4. 50 оттенков нагрузочного тестирования [Электронный ресурс] URL: <https://habr.com/ru/companies/ozontech/articles/662800/> (Дата обращения: 10.04.2023)
5. Нотации модели сущность-связь – [Электронный ресурс] – URL: <https://pro-prof.com/archives/8126> (Дата обращения: 15.04.2023)
6. Архитектура клиент-сервер: двухзвенная и трехзвенная – [Электронный ресурс] – URL: <https://www.freepapers.ru/28/serveryprilozhenij/260084.1722109.list1.html> (Дата обращения: 20.04.2023)
7. Руководство по Node.js – [Электронный ресурс] – URL: <https://metanit.com/web/nodejs/> (Дата обращения: 25.04.2023)
8. Современный учебник JavaScript – [Электронный ресурс] – URL: <https://learn.javascript.ru/> (Дата обращения: 27.02.2023)
9. Использование диаграммы вариантов использования UML при проектировании программного обеспечения – [Электронный ресурс] – URL: <https://habr.com/ru/articles/566218/> (Дата обращения: 02.05.2023)
10. Учебное пособие по диаграммам последовательностей: полное руководство с примерами – [Электронный ресурс] – URL: <https://creately.com/blog/ru/диаграмма/учебное-пособие-по-последовательной/> (Дата обращения: 10.05.2023)

ПРИЛОЖЕНИЕ А

Листинги хранимых процедур

```
const pool = require('../config/db.config');

class UserModel {

  async checkEmail(email) {
    const find = await pool.query('SELECT "user".email FROM "user" ' +
      'WHERE "user".email = $1', [email]);
    return Boolean(find.rows[0]);
  }

  async createUser({first_name, last_name, patronymic, passport, email,
    hashPassword}) {
    await pool.query('INSERT INTO "user" (first_name, last_name,
      patronymic, passport, email, password, access_id)\n' +
      ' VALUES ($1, $2, $3, $4, $5, $6, 1) RETURNING *;',
      [first_name, last_name, patronymic, passport, email, hashPassword]);
  }

  async getPassword(email) {
    const find = await pool.query('SELECT password FROM "user" WHERE
      email = $1', [email]);
    return find.rows[0].password;
  }

  async getIdAndRole(email) {
    const find = await pool.query('SELECT user_id, access_name AS "role"
      FROM "user", access_level' +
      ' WHERE email = $1 AND "user".access_id =
      access_level.access_id;', [email]);
    return find.rows[0];
  }

  async getName(id) {
    const find = await pool.query('SELECT first_name AS "name",
      patronymic FROM "user"' +
      ' WHERE user_id = $1;', [id]);
    return find.rows[0];
  }

  async getUser(user_id) {
    let select = 'select * from "user" where "user".user_id = $1;';
    const rows = await pool.query(select, [user_id]);

    return rows.rows[0];
  }

  async updateUser(user_id, name, surname, patronymic, passport, email,
    password) {
    let select = 'update "user" set first_name = $1, last_name = $2,
      patronymic = $3, ' +
      ' passport = $4, email = $5';
    if (password) select += ' , password = '${password}'';
    select += ' where "user".user_id = $6;';
    await pool.query(select, [name, surname, patronymic, passport, email,
      user_id]);
  }
}
```

```

        return 'OK';
    }

}

module.exports = new UserModel();

```

Листинг А.1 - модель «user», отвечающая за взаимодействия с данными пользователей

```

const UserModel = require('../model/user.model');
const pool = require('../config/db.config');

describe('UserModel', () => {
    // Создание тестового пользователя
    const testUser = {
        first_name: 'John',
        last_name: 'Doe',
        patronymic: 'Smith',
        passport: 'ABC123',
        email: 'test@example.com',
        hashPassword: 'password'
    };

    beforeAll(async () => {
        // Подключение к тестовой базе данных
        await pool.connect();
    });

    it('Проверка новой почты', async () => {
        const result = await UserModel.checkEmail('nonexisting@example.com');
        expect(result).toBe(false);
    });

    it('Проверка существующей почты', async () => {
        await UserModel.createUser(testUser);
        const result = await UserModel.checkEmail(testUser.email);
        expect(result).toBe(true);
    });

    it('Проверка верного пароля', async () => {
        const password = await UserModel.getPassword(testUser.email);
        expect(password).toBe(testUser.hashPassword);
    });

    it('Проверка возврата id и роли', async () => {
        const { user_id, role } = await
UserModel.getIdAndRole(testUser.email);
        expect(user_id).toBeDefined();
        expect(role).toBe('Пользователь');
    });

    it('Проверка возврата имени и отчества', async () => {

```

```

    const { user_id } = await UserModel.getIdAndRole(testUser.email);
    const { name, patronymic } = await UserModel.getName(user_id);
    expect(name).toBe('John');
    expect(patronymic).toBe('Smith');
  });

  it('Проверка возврата информации о пользователе', async () => {
    const { user_id } = await UserModel.getIdAndRole(testUser.email);
    const user = await UserModel.getUser(user_id);
    expect(user).toBeDefined();
    expect(user.email).toBe('test@example.com');
  });

  it('Проверка изменение данных о пользователе', async () => {
    const { user_id } = await UserModel.getIdAndRole(testUser.email);
    const updatedName = 'UpdatedName';
    const updatedSurname = 'UpdatedSurname';

    await UserModel.updateUser(user_id, updatedName, updatedSurname, 1,
1, 1, 1);

    const user = await UserModel.getUser(user_id);
    expect(user.first_name).toBe(updatedName);
    expect(user.last_name).toBe(updatedSurname);
  });
});

```

Листинг А.2 - модульные тесты модели «user»

```

<!DOCTYPE html>
<html lang="ru">
<head>
  <meta charset="UTF-8">
  <title>Касса автовокзала</title>
  <link rel="stylesheet" href="/css/main.css">
</head>
<body>
<header>
  <a href="#">
    
  </a>
  <% if (role !== 'Гость') { %>
    <p class="hello">Здравствуйте, <%= name.charAt(0).toUpperCase() +
name.substr(1).toLowerCase() + ' ' +
    patronymic.charAt(0).toUpperCase() +
patronymic.substr(1).toLowerCase() %>!</p>
  <% } %>
  <nav class="header-nav">
    <ul class="header-nav-list">
      <% if (role === 'Администратор') { %>
        <li class="header-nav-item">
          <a href="/admin">Административная панель</a>
        </li>

```

```

    <% } %>
    <% if (role !== 'Гость') { %>
        <li class="header-nav-item show-ticket">
            <a class="show-ticket">Купленные билеты</a>
        </li>
        <li class="header-nav-item">
            <a class="show-profile">Профиль</a>
        </li>
        <li class="header-nav-item">
            <a href="/logout">Выход</a>
        </li>
    <% } else { %>
        <li class="header-nav-item">
            <a href="../auth/registration">Регистрация</a>
        </li>
        <li class="header-nav-item">
            <a href="../auth/login">Вход</a>
        </li>
    <% } %>
</ul>
</nav>
</header>
<main>
    <section class="search">
        <form action="/" method="GET">
            <div class="search-from">
                <label for="search_from">Пункт отправления</label>
                <input type="text" id="search_from" name="search-from"
placeholder="Откуда?">
            </div>
            <div class="search-to">
                <label for="search_to">Пункт прибытия</label>
                <input type="text" id="search_to" name="search-to"
placeholder="Куда?">
            </div>
            <div class="date">
                <label for="date">Дата выезда</label>
                <input type="date" id="date" name="date-to">
            </div>
            <input type="button" name="submit" value="Найти" class="input-
btn" onclick="showFlights()">
        </form>
    </section>
</main>
<footer>
</footer>
<script src="https://unpkg.com/axios/dist/axios.min.js"></script>
<script
src="https://cdnjs.cloudflare.com/ajax/libs/pdfmake/0.2.2/pdfmake.min.js"></s
cript>
<script
src="https://cdnjs.cloudflare.com/ajax/libs/pdfmake/0.2.2/vfs_fonts.js"></scr
ipt>
<script src="/js/main/flights.js"></script>
<script src="/js/main/ticket.js"></script>
<script src="/js/main/profile.js"></script>
</body>
</html>

```

Листинг А.3 – еjs шаблон главной страницы

ПРИЛОЖЕНИЕ Б

Клиентская часть системы

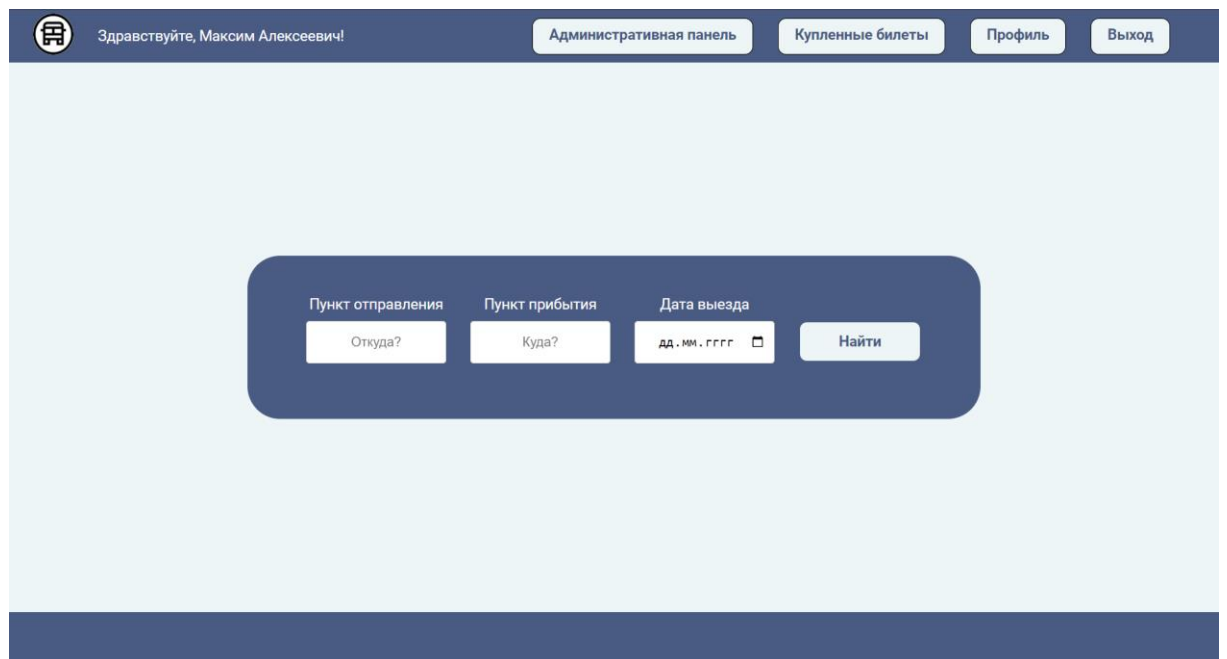


Рисунок Б.1 – Главная страница

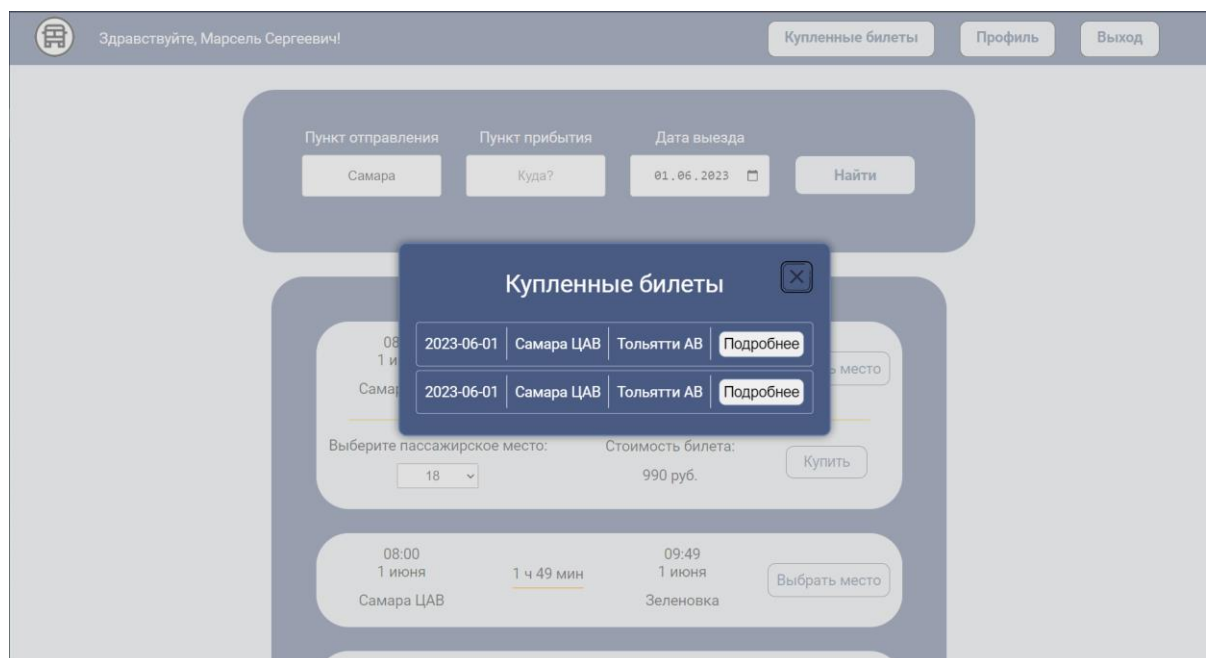


Рисунок Б.2 – Список купленных билетов

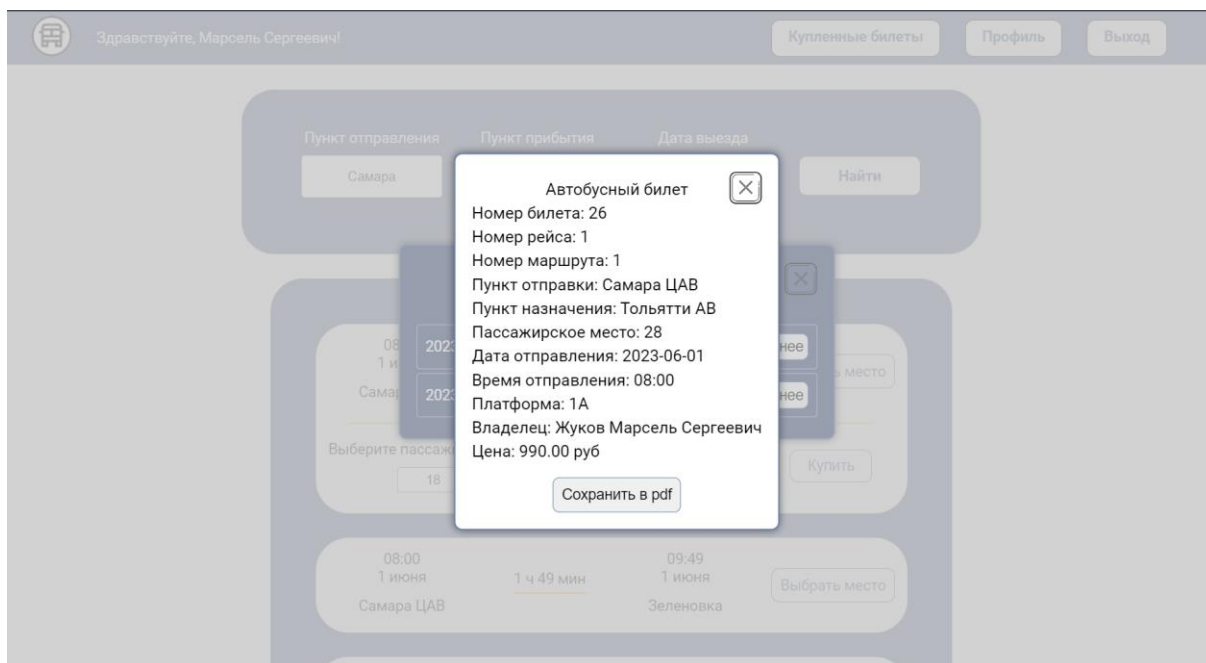


Рисунок Б.3 – Информация по конкретному билету

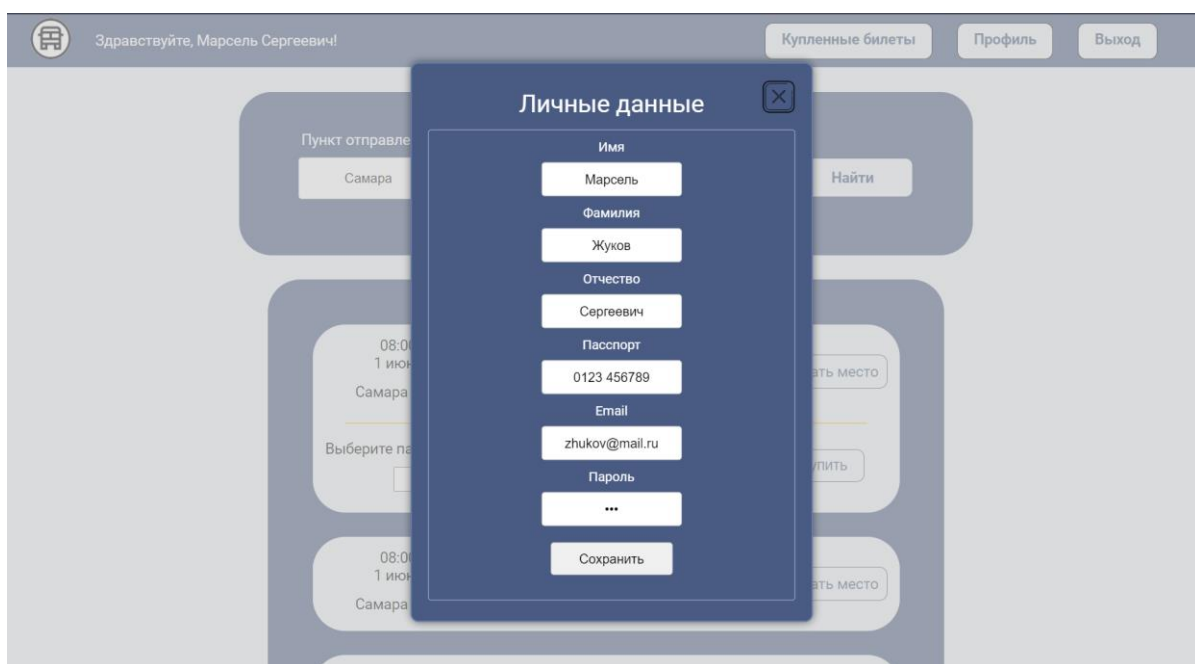


Рисунок Б.4 – Редактирование личных данных

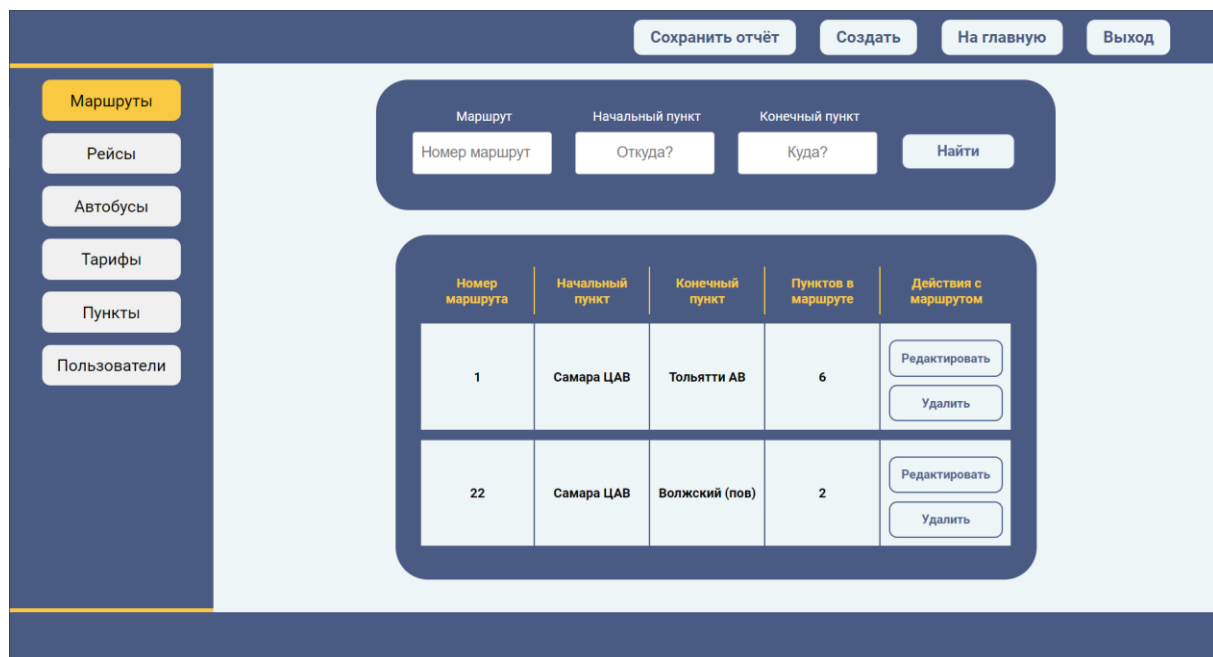


Рисунок Б.5 – Административная панель, редактирование маршрутов



Рисунок Б.6 – Административная панель, редактирование пунктов