МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РФ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ

Факультет вычислительной техники

Кафедра МОиПЭВМ

Направление подготовки 09.04.04 «Программная инженерия»

Профиль Программное обеспечение вычислительной техники и автоматизированных систем

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА МАГИСТРА

на тему

Управление мобильными устройствами в сети

Студент _____ Ионкин Илья Алексеевич
(подпись, дата) (ФИО полностью)

Руководитель _____ Шашков Б.Д.
(подпись, дата) (фамилия, инициалы)

Нормоконтролер _____ Такташкин Д.В.
(подпись, дата) (фамилия, инициалы)

Рецензент нач. отдела НПТП „Криптософт" _____ 28.06.21 Прокин А.Ю.
(подпись, дата) (фамилия, инициалы)

Работа допущена к защите (протокол заседания кафедры от _____ № _____)

Заведующий кафедрой _____ Макаричев К.А.
(подпись) (фамилия, инициалы)

Работа защищена с отметкой 5 (протокол заседания ГЭК от 30.06.21 № 3)

Секретарь ГЭК _____ Балова И.А.
(подпись) (фамилия, инициалы)

г. Пенза, 2021 г.

Содержание

Введение 4
Актуальность темы 4
1 Анализ предметной области. Выбор технологий и инструментальных средств 7
1.1 Анализ процессов системы контроля общественного транспорта и пассажиропотока 7
1.2 Обзор существующих информационных систем контроля общественного транспорта и учета пассажиропотока 10
1.2.1 «Яндекс. Транспорт» 11
1.2.2 «Умный транспорт» 12
1.2.3 «Hubb: городской транспорт» 13
1.3 Постановка задачи 16
1.4 Выбор технологий и средств разработки 16
1.4.1 Средства для анализа системы 16
1.4.2 Средства реализации серверной части 18
1.4.3 Система управления базами данных и инструмент проектирования базы данных 20
1.4.4 Средства реализации тонкого клиента 21
1.4.5 Средства реализации толстого клиента 22
1.4.6 Среды разработки 25
1.4.7 Средства моделирования 26
Выводы 26
2 Разработка и исследование математической модели системы контроля общественного транспорта и пассажиропотока 28
2.1 Математическая модель системы контроля общественного транспорта и пассажиропотока в архитектуре клиент-сервер 28
2.2 Построение и исследование имитационной модели 38
Выводы 40
3 Разработка информационной системы контроля общественного транспорта и пассажиропотока 42

3.1 Архитектура информационной системы контроля общественного транспорта и пассажиропотока 42
3.2 Разработка базы данных системы 45
3.2.1 Разработка логической модели базы данных 45
3.2.2 Анализ уровня нормализации логической модели базы данных 47
3.2.3 Разработка физической модели базы данных 49
3.2.4 Пользовательский интерфейс клиентского приложения 51
3.3 Разработка программного обеспечения 55
3.3.1 Разработка программного обеспечения серверной части 55
3.3.2 Разработка программного обеспечения тонкого клиента 60
3.3.3 Разработка программного обеспечения мобильного приложения 60
3.4 Контроль качества программного обеспечения 61
3.5 Функциональное тестирование программного обеспечения 61
Выводы 63
Заключение 65
Список использованных источников 66
Приложение А ГЛОССАРИЙ 69
Приложение Б ЛИСТИНГ СЕРВЕРНОГО ПРИЛОЖЕНИЯ 72
Приложение В ЛИСТИНГ КЛИЕНТСКОГО ПРИЛОЖЕНИЯ 78
Приложение Г РЕЗУЛЬТАТЫ ФУНКЦИОНАЛЬНОГО ТЕСТИРОВАНИЯ 87

Введение

Актуальность темы. В современном мире всё большую значимость приобретает транспорт, проникая во все отрасли нашей жизни. Система общественного транспорта стала неотъемлемой частью современного города. Основной задачей транспортной системы является обеспечения потребности поселения в перевозках. По мере роста численности населения и городской территории, растет подвижность и средняя дальность поездок жителей города.

Общественный транспорт в городе можно разделить на несколько видов. Это разные виды автобусов, троллейбусы, трамваи, такси и маршрутные такси. Именно последние являются наиболее популярным видом общественного транспорта. Данный вид транспорта обладает рядом преимуществ, такими как высокая скорость доставки, широкий охват транспортной сети, относительный уровень комфорта. Также маршрутные такси обладают также рядом недостатков. Например, водители маршрутных такси достигают высоких скоростей сообщения не за счет уменьшения количества остановок, а за счет скоростных качеств транспортного средства. Такое поведение вызвано постоянной конкуренцией за пассажира и стремлением совершить большее количество поездок. Это приводит к «очередям» из общественного транспорта на остановках и ведет к ухудшение качества обслуживания пассажиров [1]. Подтвердив это результатами имитационного моделирования, актуально создать информационную систему с толстым и тонким клиентами, которая обеспечит контроль и учет пассажиропотока.

Развитие современных технологий создают новый подход к использованию более эффективных средств поиска, обработки, хранения, предоставления и передачи информации. Большой вклад в автоматизацию различных процессов вносят мобильные устройства. Современные мобильные устройства по своим характеристикам не уступают вчерашним настольным компьютерам. Использование мобильного приложения облегчает нагрузку на сервер и позволяет осуществить доступ к ИС в любом месте с доступом в сеть Интернет.

К настоящему времени имеется ряд систем, обеспечивающих контроль

общественного транспорта и пассажиропотока, таких как «Яндекс. Транспорт», «Умный транспорт», «Hubb: городской транспорт» и др.

А. А. Барсегян, И. Ф. Кодд, Г. Уильман, Дж. Маккенел, Т. Л. Саати, А. Бергер и др. внесли большой вклад в решение проблемы информационных систем, которые содержат эффективные средства сбора данных. Дж. Уолрэнд, В. В. Крылов, А. К. Эрланг, В. Г. Башарин, В. Л. Брейдо, А. А. Марков, Б. С. Лившиц также внесли значительный вклад в развитие имитационных и математических методов исследования характеристик информационных систем.

Цель работы – Рассмотреть проблему балансировки загруженности сервера в зависимости от толщины клиента. Создать систему для контроля общественного транспорта и учета пассажиропотока, с использованием сервера, браузера и мобильного клиента (смартфон, планшет).

Задачи исследования. В соответствии с целями решены следующие задачи:

– провести исследование балансировки загрузки сервера в зависимости от толщины клиента;

– разработать программное обеспечение системы контроля общественного транспорта и учета пассажиропотока.

Объект исследования. Информационная система контроля пассажиропотока и общественного транспорта, позволяющая пассажиру отслеживать движение общественного транспорта, а водителю общественного транспорта отслеживать количество ожидающих на ближайшей остановке.

Методы исследования. Исследования основаны на теории информационных систем, теории имитационного моделирования, теории массового обслуживания.

Научная новизна работы заключается в следующем:

а) разработана имитационная модель системы контроля общественного транспорта и пассажиропотока;

б) проведено имитационное моделирование системы контроля общественного транспорта и пассажиропотока;

в) применена методика анализа свойств имитационной модели;

г) произведены сравнительные измерения основных характеристик для клиентов разной функциональной загруженности.

Практическая значимость исследований. Полученные в диссертации практические результаты позволяют значительно расширить функциональные возможности информационной системы контроля общественного транспорта и пассажиропотока. Разработанная система контроля общественного транспорта и пассажиропотока с толстым и тонким клиентом позволяют уменьшить нагрузку на сервер.

Результаты, выносимые на защиту:

– имитационная модель системы контроля общественного транспорта и пассажиропотока;

– информационная система, которая позволяет отслеживать перемещение общественного транспорта, выбирать необходимый маршрут, отслеживать количество ожидающих пассажиров на ближайшей остановке.

Апробация работы. Основные результаты диссертационной работы докладывались и обсуждались на следующих конференциях: VII Всероссийской межвузовской научно-практической конференции «Информационные технологии в науке и образовании. Проблемы и перспективы», (Пенза, 2020); VIII Всероссийской межвузовской научно-практической конференции «Информационные технологии в науке и образовании. Проблемы и перспективы», (Пенза, 2021).

Публикации. По теме диссертационного исследования опубликовано 2 печатные работы: 2 статьи в материалах российских и международных конференций (РИНЦ).

Структура и объём работы. Диссертация состоит из введения, трех глав, заключения, списка литературы, заключения и 4 приложений. Объём работы: 71 страница основного текста, включающего 41 рисунок, 5 таблиц и 22 страницы приложений.

1 Анализ предметной области. Выбор технологий и инструментальных средств

Рассматриваются существующие системы контроля общественного транспорта и пассажиропотока. Определяется набор требований. Производится выбор технологий, средств проектирования, разработки и моделирования информационных систем подобного рода.

1.1 Анализ процессов системы контроля общественного транспорта и пассажиропотока

Система контроля общественного транспорта и пассажиропотока будет выполнять следующие функции:

- отображение существующих маршрутов общественного транспорта на карте;
- выбор необходимого маршрута;
- отображение движения общественного транспорта на карте;
- просмотр списка активных маршрутов;
- отображение положение пользователя на карте;
- отображение остановок на карте;
- отображение количество ожидающих пассажиров на остановке;
- добавление новых и изменение остановок;
- добавление новых и изменение маршрутов.

Система рассчитана на 3 группы пользователей: водитель общественного транспорта, пассажир, сотрудник контроля маршрутов. Сотрудник контроля маршрутов после авторизации, может добавлять новые и изменять остановки, добавлять и изменить маршруты, назначать водителя ТС на определенный маршрут, регистрировать новое ТС, осуществлять регистрацию водителя, удалять водителя. Водитель общественного транспорта может отправляться на маршрут, видеть других водителей ТС своего маршрута, видеть количество ожидающих пассажиров на ближайшей остановке. Пассажир может выбирать необходимый маршрут, видеть активные общественные ТС, писать отзывы по работе приложения.

Краткое описание основных вариантов использования разрабатываемого программного обеспечения для пользователя сотрудник

контроля маршрутов представлено в таблице 1.

Таблица 1 – Краткое описание вариантов использования

Цель	Краткое описание
Добавить маршрут	Пользователь добавляет маршрут в БД
Добавить остановку	Пользователь добавляет остановку в БД
Регистрация водителя	Пользователь регистрирует нового водителя
Добавление нового ТС	Пользователь регистрирует новое ТС
Изменение маршрута	Пользователь изменяет маршрут
Закрытие остановки	Пользователь закрывает остановку
Удаление водителя	Пользователь удаляет водителя
Удаление остановки	Пользователь удаляет остановку
Назначение водителя на маршрут	Пользователь назначает водителя на маршрут
Просмотреть список доступных маршрутов	Пользователь просматривает список доступных маршрутов
Просмотр ожидающих пассажиров	Пользователь видит число ожидающих пассажиров
Выбор необходимого маршрута	Пользователь выбирает маршрут из списка маршрутов
Просмотр остановок	Пользователь просматривает ближайшие остановки

Помимо основных функции система должна хранить информацию, которую тоже необходимо занести в базу данных приложения.

Диаграмма прецедентов разрабатываемого программного обеспечения для обозначенных групп пользователей, с учетом дополнительных вариантов использования представлена на рисунках 1 и 2.

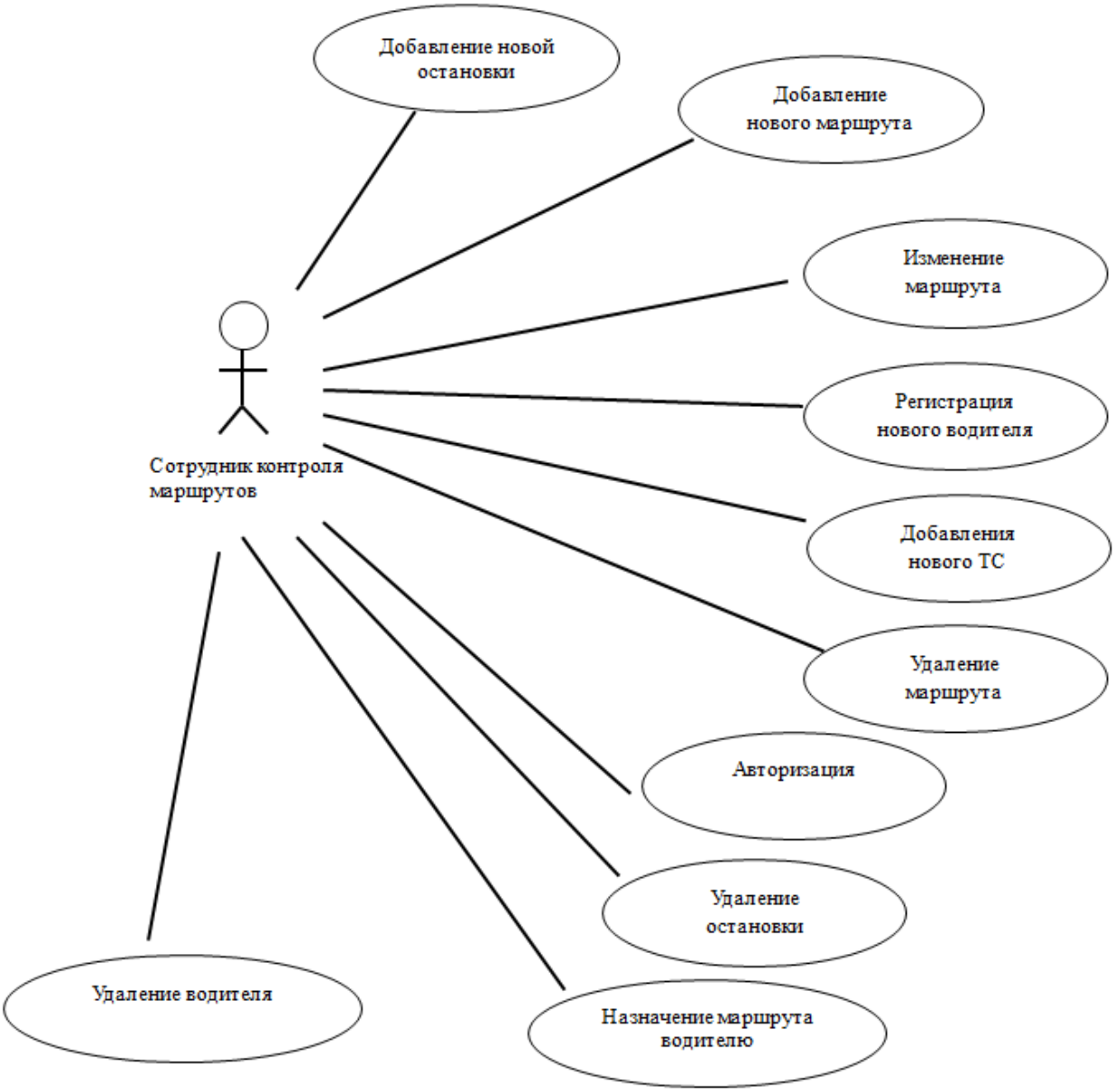

Рисунок 1 – Диаграмма прецедентов

Варианты использования для мобильного приложения ограничиваются вариантами использования пассажира и водителя общественного транспорта, На рисунке 2 представлена диаграмма прецедентов мобильного приложения. Данная диаграмма показывает варианты взаимодействия с системой пассажиров и водителей общественного транспорта. Водитель может осуществить выход на маршрут, отметив его как активный, выбирать необходимый маршрут и видеть количество пассажиров ожидающих его на следующей остановке.

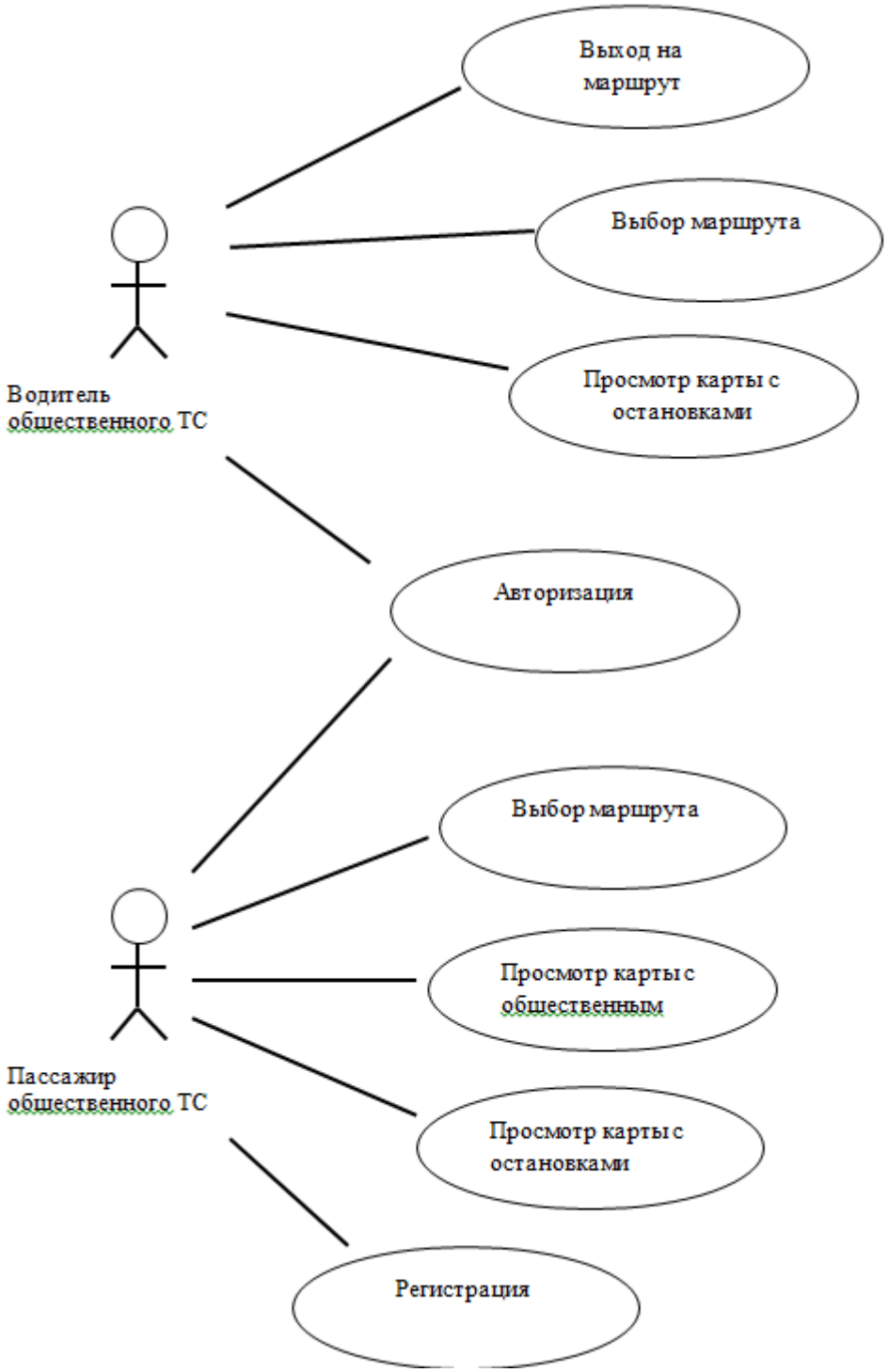

Рисунок 2 – Диаграмма прецедентов мобильного приложения

Пользователь может авторизоваться, зарегистрироваться, видеть список доступных маршрутов, выбирать необходимый маршрут.

1.2 Обзор существующих информационных систем контроля общественного транспорта и учета пассажиропотока

Прежде, чем разрабатывать какое – либо программное обеспечение, необходимо провести обзор подобных программных продуктов.

1.2.1 «Яндекс. Транспорт»

«Яндекс. Транспорт» - навигационное приложение для перемещения по городу без личного автомобиля, разработанное российской интернет-компанией «Яндекс». На рисунке 3 изображен интерфейс рассматриваемой программы.

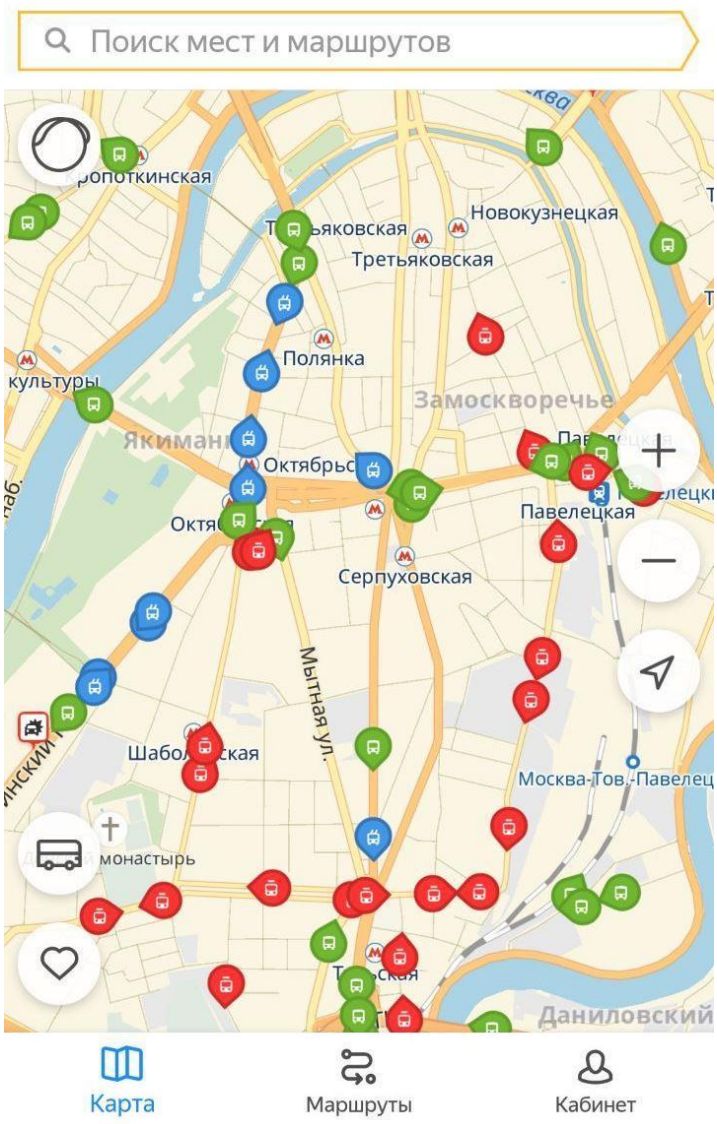

Рисунок 3 – Яндекс.Транспорт

Преимущества программного обеспечения:

- удобный интерфейс;
- перевод на несколько языков;
- отслеживание большинства видов общественного транспорта;
- большой охват доступных городов.

Недостатки программного обеспечения:

- отсутствие учета пассажиропотока.

1.2.2 «Умный транспорт»

Мобильное приложение «Умный транспорт» – это приложение для слежения за работой общественного транспорта. На рисунке 4-5 отображен интерфейс рассматриваемой программы.

Рисунок 4 – Отображение карты приложения «Умный транспорт»

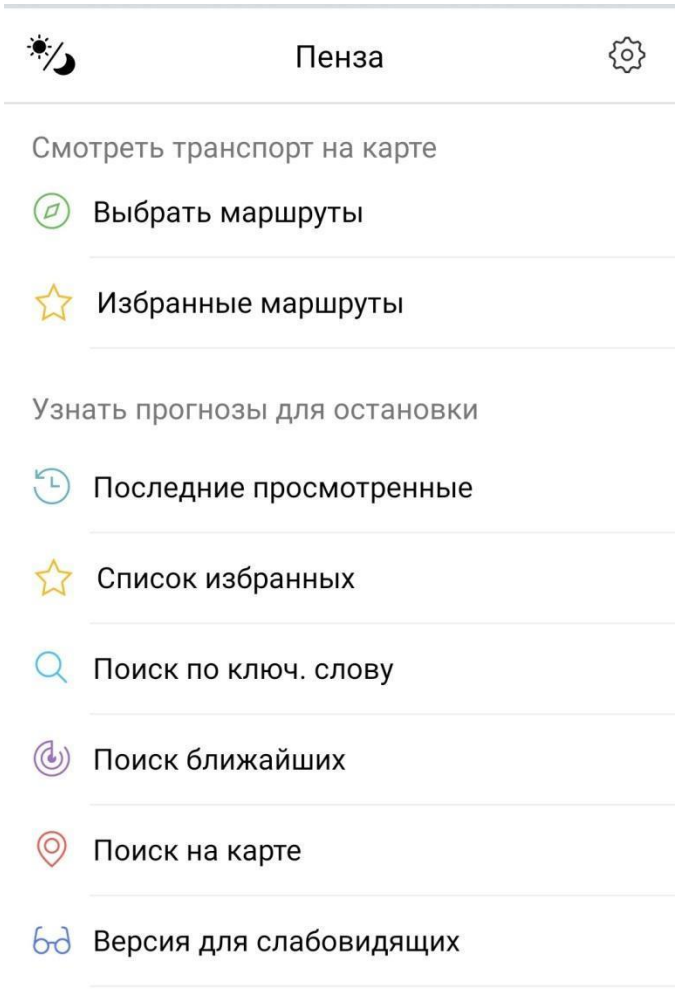

Рисунок 5 – Отображение меню приложения «Умный транспорт»

Преимущества программного обеспечения:

- простой и удобный интерфейс;
- возможность добавлять маршруты в избранное;
- демонстрация прогноза прибытия ТС на остановку.

Недостатки программного обеспечения:

- отсутствует возможность учета пассажиропотока;
- небольшой список обслуживаемых городов;
- полное отключение функционала при обрыве соединения с Интернетом.

1.2.3 «Hubb: городской транспорт»

Мобильное приложение «Hubb: городской транспорт» – приложение для слежения за работой общественного транспорта. На рисунке 6 и 7 изображен пользовательский интерфейс рассматриваемой программы.

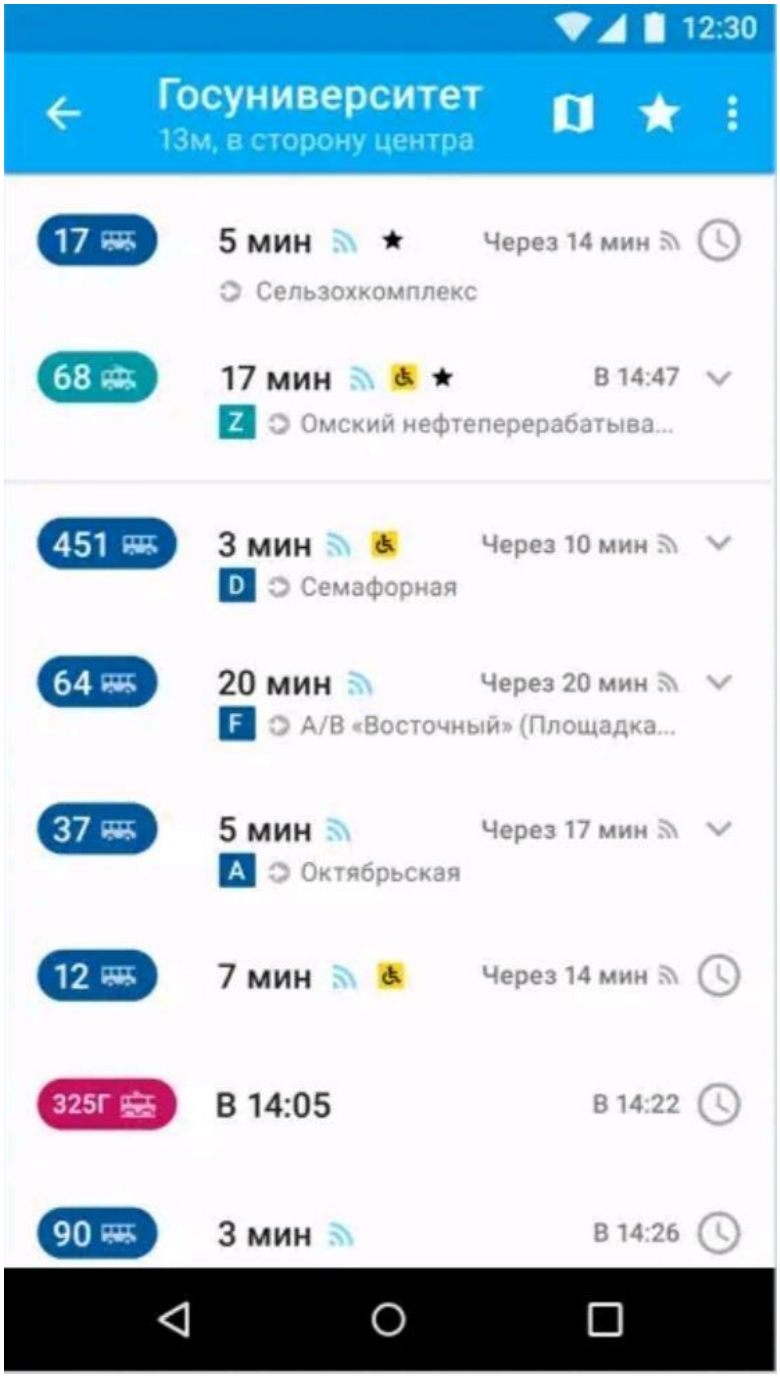

Рисунок 6 – Отображение меню приложения «Hubb: городской транспорт»

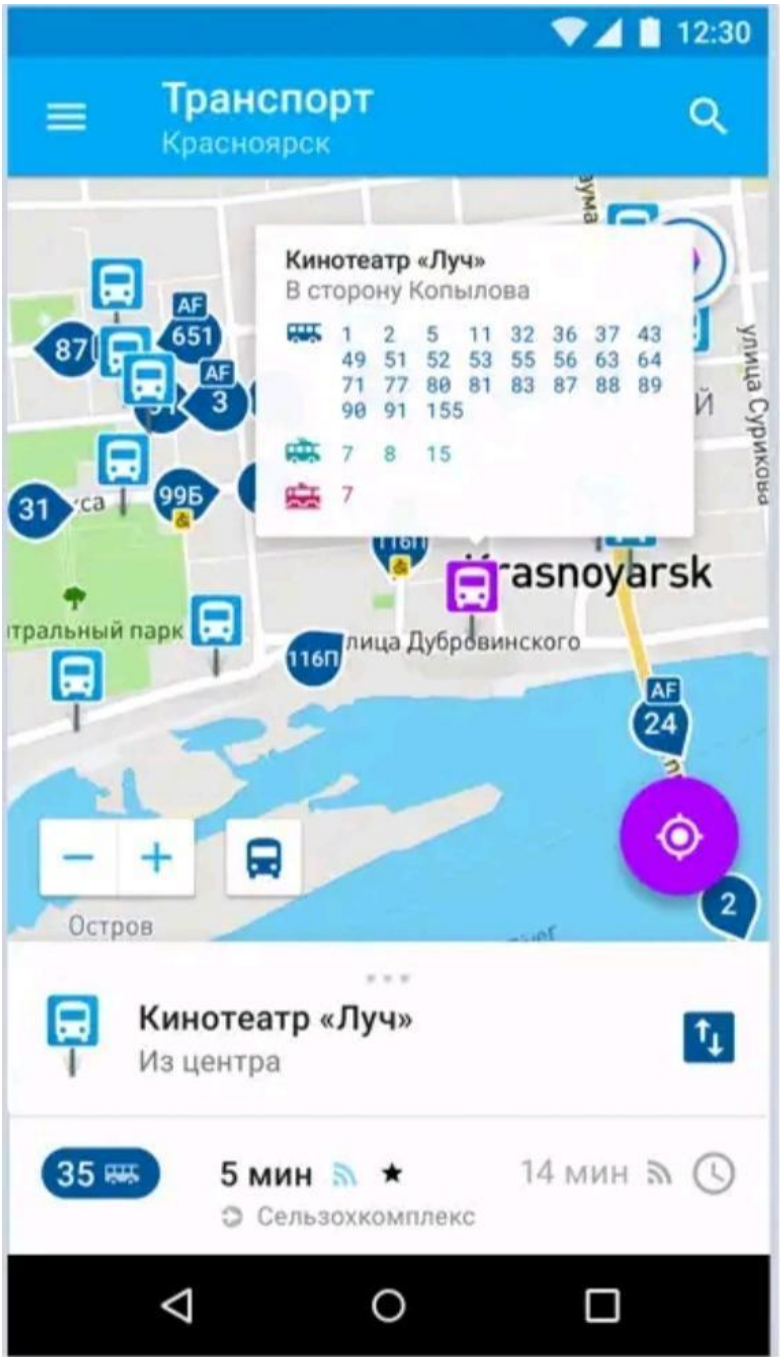

Рисунок 7 – Отображение карты приложения «Hubb: городской транспорт»

Преимущества программного обеспечения:

– удобный интерфейс;

Недостатки программного обеспечения:

– отсутствует учет пассажиропотока;

– отсутствует возможность добавлять маршруты в избранное;

– небольшое количество доступных городов;

– отключения функционала при разрыве соединения с Интернетом.

Общим недостатком у всех рассмотренных систем является то, что они создавались для получения информации об общественном транспорте пользователю. Ни одна из рассмотренных систем не позволяет водителю

получить информацию о количестве ожидающих пассажиров.

1.3 Постановка задачи

После проведенного обзора аналогичных систем, было принято решение создать удобную систему для мониторинга общественного транспорта и учета пассажиропотока, с использованием толстого и тонкого клиента.

Удобство системы заключается в использовании мобильного клиента (смартфон, планшет), что несомненно увеличивает мобильность использования приложения. Также водители общественного транспорта получат возможность видеть количество пассажиров ожидающих на остановке.

В этом заключается практическая цель данной работы.

На базе решения данной практической цели необходимо рассмотреть проблему балансировки загруженности сервера в зависимости от толщины клиента.

Для достижения поставленных целей следует решить следующие задачи:

а) разработать программное обеспечение системы;

б) провести исследование балансировки загрузки сервера в зависимости от толщины клиента.

1.4 Выбор технологий и средств разработки

Для выполнения поставленных задач необходимо определиться с используемыми инструментальными средствами проектирования, разработки программного обеспечения и моделирования информационной системы.

1.4.1 Средства для анализа системы

CASE-средства позволяют проектировать любые системы на компьютере. Они всегда являются необходимым элементом системного и структурно-функционального анализа. Также CASE-средства помогают моделировать различные деятельность и структуру организаций, компоненты программного обеспечения, бизнес-процессы и базы данных.

Для средства для построения диаграмм UML были использованы такое программное обеспечение, как Microsoft Visio и Rational Rose.

Microsoft Visio. Visio – это решение для построения диаграмм от Microsoft. Visio помогает отобразить бизнес-концепции или технические концепции в визуальную форму. Microsoft Visio обладает следующими основными свойствами:

– используя предопределенные фигуры Visio Professional , drag-and-drop и мастера, можно быстро и просто создавать понятные и информативные диаграммы;

– возможности Visio можно легко расширять, используя новые шаблоны бизнес-диаграмм;

– в Visio можно прототипировать интерфейс приложений с помощью встроенных шаблонов пользовательского интерфейса Microsoft Windows;

– можно легко рисовать диаграммы сетевых ресурсов, иллюстрирующие развертывание нового ПО на существующие сетевые ресурсы;

– Visio Professional также тесно интегрируется с Microsoft Office Project, что позволяет, например, импортировать оттуда задачи для членов команды;

– с помощью шаблонов UML можно создавать UML-диаграммы статической структуры ПО или проводить обратное проектирование с помощью Visio 2003 Reverse Engineer Wizard [2].

Rational Rose - современное и мощное средство анализа, моделирования и разработки программных систем. Rational Rose пригодится при решении практически любых задач проектирования информационных систем: от анализа бизнес-процессов до кодогенерации на определенном языке программирования. Такой арсенал позволит не только спроектировать новую систему, но и доработать старую, произведя процесс обратного проектирования. Вот основные возможности продукта:

– прямое и обратное проектирование на языках: ADA, Java, C, C++, Basic;

– поддержка технологий COM, DDL, XML;

– возможность генерации схем БД Oracle и SQL [3].

Также Rational Rose имеет открытый API, позволяющий самому создавать модули для других языков программирования [4].

Продукт является, по сути, стандартом де-факто в области UML-проектирования, однако не слишком интуитивен и удобен в использовании.

1.4.2 Средства реализации серверной части

Информационная система учета и контроля пассажиропотока должна удовлетворять следующим требованиям:

− система должна иметь возможность обслуживания сразу множество одновременных соединений;

− система должна быть доступна для администрирования, добавления новых маршрутов и остановок в БД, с персонального компьютера;

− система должна обеспечивать хранение и обработку данных в базе данных;

− система должна быть устойчивой к разрывам доступа к сети;

− система должна отображать корректные данные;

− система должна обладать удобным и простым интерфейсом;

− система должна учитывать возможное расширения существующего функционала и добавления новых возможностей.

Для достижения данных требований, было принято решение реализовывать систему как Web-приложение.

Web-приложение – это клиент-серверное приложение. В качестве клиента может выступать браузер или мобильное приложение. Сервером выступает Web-сервер. Обмен информацией между сервером и клиентами происходит по локальной сети, либо с использованием глобальной сети Интернет.

Для реализации серверной части были выбраны следующие средства.

Язык программирования Python**.** Одним из популярных языков программирования для написания Web-серверов является Python. высокоуровневый язык программирования общего назначения. Python является скриптовым языком используемым для управления web-сервером и

работы с базами данных. Python поддерживается подавляющим большинством представителей хостингов.

Сервер приложений Nginx. Nginx HTTP-сервер – свободный веб-сервер. Является веб-сервером и почтовым прокси-сервером, работающим на Unix-подобных операционных системах [5]. Nginx считается очень быстрым HTTP сервером. Nginx используют для ускорения обработки запросов и уменьшения нагрузки на сервер. В силу универсальности и высокой производительности nginx будет использоваться именно этот сервер.

Фреймворк для разработки программного обеспечения Tornado**.** Для разработки программного обеспечения серверной части был использован фреймворк Tornado.

Tornado – расширяемый, неблокирующий веб-сервер и фреймворк, написанный на Python [6].

Основные возможности Tornado:

– высокая производительность относительно других фреймворков, написанных на Python;

– парадигма Модель-Представление-Контроллер;

– кэширование страниц и отдельных фрагментов;

– обработка и перехват различных ошибок;

– ввод и валидация различных формы;

– возможность аутентификация и авторизация;

– возможность подключения сторонних библиотек.

Одним из главных преимуществ этого фреймворка является высокая производительность. Данный фреймворк был создан специально чтобы решить проблему C10K (до 10 тысяч одновременных соединений). Вместе с сервером на nginx можно достичь очень высокой производительности, что является необходимым условием в разрабатываемой информационной системе.

Таблица 2 – Сравнение производительности с другими веб-фреймворками на Python

Сервер	Конфигурация	Запросов в секунду
Tornado	nginx	8213

Tornado	1 однопоточный фронтенд	3353
Django	Apache/mod_wsgi	2223
Web.py	Apache/mod_wsgi	2066
CherryPy	standalone	785

1.4.3 Система управления базами данных и инструмент проектирования базы данных

PostgreSQL – свободная объектно-реляционная система управления базами данных. Большое количество функций, высокая производительность при сложных запросах, MVCC и объектно-реляционная основа являются основными причинами ее популярности.

PostgreSQL поддерживает большинство возможностей, например, транзакции, блокирование на уровне строк, внешние ключи, подзапросы , полнотекстовый поиск, использование индексов.

MVCC предоставляет одновременный доступ к БД множеству агентов на чтение и запись. Это устраняет необходимость каждый раз блокировать чтение-запись, когда кто-то взаимодействует с данными. Таким образом, значительно повышается эффективность управления СУБД и ее производительность [7].

Система PostgreSQL хорошо проверена и надежна. Версия MySQL 13.1 вышла 12 ноября 2020 года и является самой последней версией postgreSQL.

Основные преимущества PostgreSQL:

– высокопроизводительные и надёжные механизмы транзакций и репликации;

– возможность использовать многомерные массивы;

– управление параллельным доступом посредством многоверсионности (MVCC);

– расширяемая система встроенных языков программирования: в стандартной поставке поддерживаются PL/pgSQL, PL/Perl, PL/Python и PL/Tcl;

– наследование;

– возможность индексирования геометрических объектов;

– встроенная поддержка слабоструктурированных данных в формате JSON с возможностью их индексации;

– расширяемость (возможность создавать новые типы данных, типы индексов, языки программирования, модули расширения, подключать любые внешние источники данных).

Aqua Data Studio. Для проектирования базы данных была выбрана Aqua Data Studio. Это интегрированная среда программирования, облегчающая создание запросов к базам данных. Осуществляя поддержку множества операционных систем, решение Aqua Data Studio обеспечивает пользователей богатым спектром возможностей управления базами данных от различных производителей (Oracle, DB2, Microsoft SQL Server, MySQL, Sybase, Informix и PostgreSQL) [8].

К преимуществам можно отнести:

– прямой выход к большинству функциональных возможностей программы с помощью интерфейса программирования приложений;

– эффективная разработка и обслуживание корпоративных реляционных баз данных;

– реализация возможности разработки сценариев, получающих выход к функциональным возможностям программы для построения пользовательских заданий и решений;

– функции случайной генерации данных и статистики;

– встроенные программные средства Entity-Relationship (ER) Modeler обеспечивают наглядное представление зависимости объектов баз данных [8].

1.4.4 Средства реализации тонкого клиента

Язык программирования JavaScript**.** JavaScript – это межплатформенный объектно-ориентированный язык сценариев, который был создан Netscape. Как правило JavaScript не используется для создания самостоятельных приложений, а используется для упрощенного внедрения в другие приложения и продукты, например web-браузеры. JavaScript может подключаться к различным объектам среды, в который он был запущен.

Клиентский JavaScript расширяет ядро языка, добавляя объекты

управления браузером (Navigator или другой аналогичный web-браузер) и Document Object Model/Объектную Модель Документа (DOM). Например, клиентские расширения позволяют приложению размещать элементы на HTML-форме и реагировать на пользовательские события, такие как щелчок мышью, ввод данных в форму или навигация по страницам [9].

jQuery - это библиотека, значительно упрощающая и ускоряющая написание JavaScript-кода. Девиз jQuery «write less, do more» (пиши меньше, делай больше) отражает ее главное предназначение. jQuery содержит команды, которые могут создавать анимации и обрабатывать различные события. Также эта библиотека очень сильно облегчает выбор элементов и создание AJAX запросов.

Данная библиотека совместима с многими популярными браузерами (IE 6.0+, FF 2.0+, Safari 3.0+, Opera 9.0+, Chrome). Это означает, что ненужно заботиться о кроссбраузерной совместимости JavaScript-кода. Для jQuery написано огромное количество плагинов, которые расширяют ее возможности еще больше [10].

Язык гипертекстовой разметки**.** HTML (Hyper Text Markup Language) – «язык гипертекстовой разметки», предназначенный для создания и просмотра Web-страниц в Сети [11].

HTML – это универсальный язык для функциональной классификации различных частей документа в соответствии с их назначением. HTML показывает, какая часть документа является заголовком, а какая – текстом, какую часть текста надо выделить, и где должны располагаться графические изображения [12].

HTML обладает следующими достоинствами:

- гибкость;
- упрощение отладки;
- независимость.

На данный момент актуальной версией HTML является версия 5.

1.4.5 Средства реализации толстого клиента

Операционная система Android**.**

В настоящее время является самой популярной платформой для разработки приложений для мобильных устройств является Android. По широте возможностей платформа Android не уступает операционным системам настольных ПК. Это многоуровневая среда с богатыми функциональными возможностями.

Android как правило представляют как комбинацию трех компонентов:

- свободная операционная система с открытыми исходными кодами;
- среда разработки;
- множество устройств, на которых установлена операционная система Android.

Android состоит из нескольких необходимых и взаимозависимых элементов:

- референс-дизайн аппаратного обеспечения с перечнем требований к мобильным устройствам, чтобы гарантировать совместимость с ПО;
- ядром операционной системы является Linux. Он предоставляет низкоуровневый интерфейс для управления аппаратным обеспечением;
- библиотеки с открытыми исходными кодами, предназначенными для разработки приложений SQLite, WebKit, OpenGL и медиа-менеджер;
- среду исполнения для приложений, включающую виртуальную машину Dalvik и библиотеки ядра;
- набор программных компонентов, обеспечивающих доступ к системным службам на уровне приложений; среди них менеджер окон и менеджер местоположения и т.д.;
- набор компонентов пользовательского интерфейса для размещения и запуска приложений;
- предустановленные приложения, поставляемые в общем программном наборе;
- комплект программ для разработки приложений, включающий инструменты, плагины и справочную документацию.

Для отображении графики Android использует библиотеку OpenGL. Хранение необходимых данных берет на себя популярная база данных с

открытым исходным кодом SQLite.

Стоит обратить внимание, что открытая архитектура Android позволяет исправлять ошибки в пользовательском интерфейсе или дизайне встроенных приложений. Android предоставляет возможность создавать собственные интерфейсы для мобильных телефонов, а также приложения с функционалом и дизайном, максимально отвечающими вашим потребностям [13]. Чаще всего, при разработки приложений для операционной системы Android используется язык Java, т.к. использование данного языка обеспечивает высокую скорость работы создаваемых приложений.

Язык программирования Java.

Java представляет собой высокоуровневый объектно-ориентированный язык программирования. При компиляции, которая выполняется один раз во время сборки приложения, код на Java преобразуется в байт-код. Байт-код интерпретируется виртуальной машиной Java (JVM), которая играет роль транслятора между языком Java и аппаратным обеспечением с операционной системой. Все реализации Java должны эмулировать JVM, чтобы создаваемые приложения могли выполняться на любой системе, включающей виртуальную машину Java.

Основным преимуществом языка Java является переносимости Java-приложений. Это значит, что приложения будут смогут работать на любых аппаратных платформах, т.к. все JVM, способны исполнять один и тот же байт-код.

База данных SQLite**.** Операционная система Android использует встраиваемую базу данных SQLite. Это значит, что SQLite не использует парадигму клиент-сервер. SQLite предоставляет собой библиотеку, с которой программа компонуется. Такой принцип позволяет использовать вызовы функций библиотеки SQLite для обмена данными. Это позволяет уменьшить время отклика и упрощает программу.

SQLite хранит всю базу данных программы в единственном стандартном файле на устройстве, на котором исполняется программа. Перед началом транзакции весь файл с БД блокируется. ACID-функции достигаются за счёт создания файла журнала

Чтение из базы данных может осуществлять несколько процессов

одновременно, но записывать данные может только один, при этом блокируется запись осуществляется, только если к базе данных в этот момент нет других запросов.

В комплекте поставки идёт также функциональная клиентская часть в виде исполняемого файла sqlite3, с помощью которого демонстрируется реализация функций основной библиотеки.

Благодаря архитектуре движка возможно использовать SQLite как на встраиваемых системах, так и на выделенных машинах с гигабайтными массивами данных [14].

1.4.6 Среды разработки

JetBrains PyCharm – коммерческая кроссплатформенная интегрированная среда разработки для языке программирования Python. Существуют несколько версий сред разработки. Community предоставляет ограниченный доступ к функционалу PyCharm. Professional дает доступ ко всему функционалу. Также JetBrains предоставляет возможность получить бесплатную версию в целях обучения. Разрабатывается компанией JetBrains на основе платформы IntelliJ IDEA.

PyCharm представляет собой интеллектуальный редактор для Python, HTML и JavaScript с возможностями анализа кода на лету, предотвращения ошибок в коде и автоматизированными средствами рефакторинга для Python и JavaScript. Также среда имеет встроенную поддержку популярных фреймворков, например Django. Имеется полноценный SQL-редактор с возможностью редактирования полученных результатов запросов.

PyCharm разработан на основе платформы IntelliJ IDEA, написанной на Java. Пользователи могут активировать плагины разработанные JetBrains или другими людьми, тем самым увеличивая функционал среды.

Для разработки серверной части используется среда разработки от JetBrains – IntelliJ IDEA – коммерческая интегрированная среда разработки программного обеспечения на многих языках программирования, в частности Java, JavaScript, Python и др., разработанная компанией JetBrains.

Android Studio – коммерческая кроссплатформенная интегрированная среда разработки для разработки под операционной системой Android.

Находится в свободном доступе. Разрабатывается компанией JetBrains на основе платформы IntelliJ IDEA [15].

Среда поддерживает несколько языков: Java, C++ и набирающий популярность язык Kotlin.

Для разработки мобильных приложений используется среда разработки от JetBrains - IntelliJ IDEA – коммерческая интегрированная среда разработки программного обеспечения для операционной системы Android на многих языках программирования, в частности Java, C++, Kotlin и др., разработанная компанией JetBrains.

1.4.7 Средства моделирования

AnyLogic – программное обеспечение для имитационного моделирования, разработанное российской компанией The AnyLogic Company. Инструмент обладает современным графическим интерфейсом и позволяет использовать язык Java для разработки моделей [16].

AnyLogic включает в себя графический язык моделирования, а также позволяет пользователю расширять созданные модели с помощью языка Java. Также программа предоставляет возможность визуализировать построенную модель для лучшего понимания.

Имитационное моделирование позволит проверить корректность выбранной модели распределения пассажиропотока и оценить улучшения пришедшие с внедрением системы.

Выводы

По результатам исследований, выполненных в настоящей главе, можно сделать выводы, перечисленные ниже.

На рынке программного обеспечения, отсутствует система учета пассажиропотока. Данная система позволит уменьшить нагрузку на водителей общественного транспорта.

Разрабатываемой системой может пользоваться несколько типов пользователей. Водитель общественного транспорта, пассажир и сотрудник контроля маршрутов.

В качестве основы сервера были выбраны сервер приложений nginx, фреймворк tornado и СУБД PostgreSQL. Вместе эти средства обеспечивают

высокую производительность при обработки запросов на сервер.

2 Разработка и исследование математической модели системы контроля общественного транспорта и пассажиропотока

2.1 Математическая модель системы контроля общественного транспорта и пассажиропотока в архитектуре клиент-сервер

Основной целью осуществления моделирования информационной системы является определение изменения основных характеристик системы в зависимости от изменения размера толщины клиента. Анализ полученных оценок характеристик позволит получить выводы о требованиях к системе.

Тонкие и толстые клиенты разрабатываемой системы могут генерировать несколько типов запросов к серверу:

– заявки на получение данных, таких как изображение, HTML-страницы или таблицы стилей;

– заявки на получение информации из базы данных,

– заявки на ввод, изменение или удаление информации в базе данных.

Рассматриваемая модель имитирует потоки заявок. Концептуальная модель информационной системы контроля общественного транспорта и пассажиропотока представлена на рисунке 8.

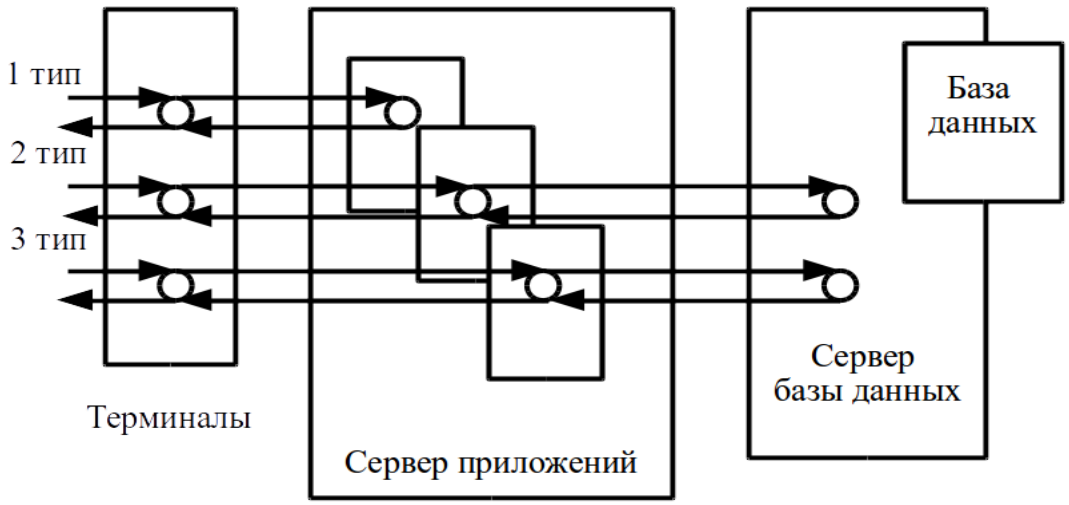

Рисунок 8 – Концептуальная модель информационной системы

Концептуальная модель состоит из терминалов, сервера приложений и сервера базы данных. Терминалы отправляют заявки 3 типов, которые попадают на сервер приложения и обрабатываются. Сервер в свою очередь

при необходимости может обращаться к серверу базы данных для получения информации или изменения данных в базе данных.

Имитационную модель системы можно изобразить в виде стохастической сети Петри. Для моделирования обработки сервером заявок 3 перечисленных выше типов, в модель введены маркеры разной формы <▼ ▲ ◆>. Маркеры, обозначаемые в виде <●> обозначают обработку запроса каналом сервера.

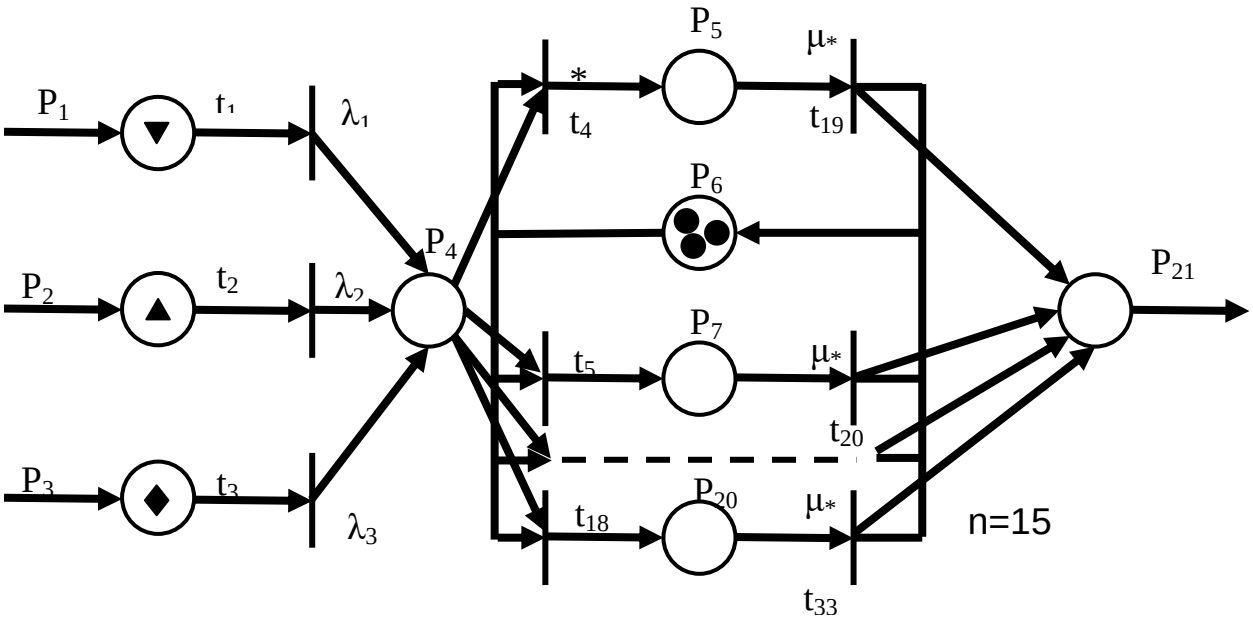

Рисунок 9 – Сеть Петри моделирующая Web-сервера с 15 каналами

На рисунке 9 изображена модель Web-сервера в виде сети Петри. $t_{1\text{-}3}$, являются источниками заявок на сервер. Заявки поступают с интенсивностями λ_1, λ_2, λ_3. Все каналы сервера можно представить в виде двух множеств. M_1 – это подмножество всех загруженных каналов системы и M_2 – это подмножество всех свободных каналов системы. Объединение этих подмножеств – M, является множеством всех каналов в системе. Отсюда следует $M1 \cap M2 = \emptyset$ и $M = (M1 \cup M2)$. Моменты, когда сервер становиться занятым и начинает обработку поступившей заявки имитируются переходами. Переходы $t_{4\text{-}18}$. имитируют обработку заявок сервером приложения.

При поступлении заявки на сервер в τ_j, ищется незанятый канал, с

минимальным номером K_q. Далее поступившая заявка занимает выбранный канал. Переменная $\tau_q^k = \tau_j$. определяет время запуска выбранного канала K_{q*}.

Переходы $t_{19\text{-}33}$ имитируют окончание обслуживания заявок занятыми каналами. Выражение

$$t_k^q = \tau_k^q + \frac{1}{\mu_i}; i = 1,2,3,k\ ,$$

где μ_i – интенсивность обслуживания заявки с раскраской i задаёт время окончания обслуживания заявки с раскраской i каналом K_q.

Систему массового обслуживания можно представить в виде адекватной модели стохастической сети Петри [13]. Таким образом можно провести анализ свойств имитационной модели разрабатываемой информационной системы контроля общественного транспорта и пассажиропотока.

Подтвердим результаты имитации аналитическими методами. Для этого предположим, что поток поступаемых на сервер заявок является однородным и у них отсутствует раскраска. В таком случае, рассматриваемому модель можно преобразовать в стохастическую сеть Петри. Данная сеть будет иметь циркулирующий однородный, ординарный потоком заявок с экспоненциальном распределением.

На рисунке 10 изображена модель системы с однородным потоком заявок.

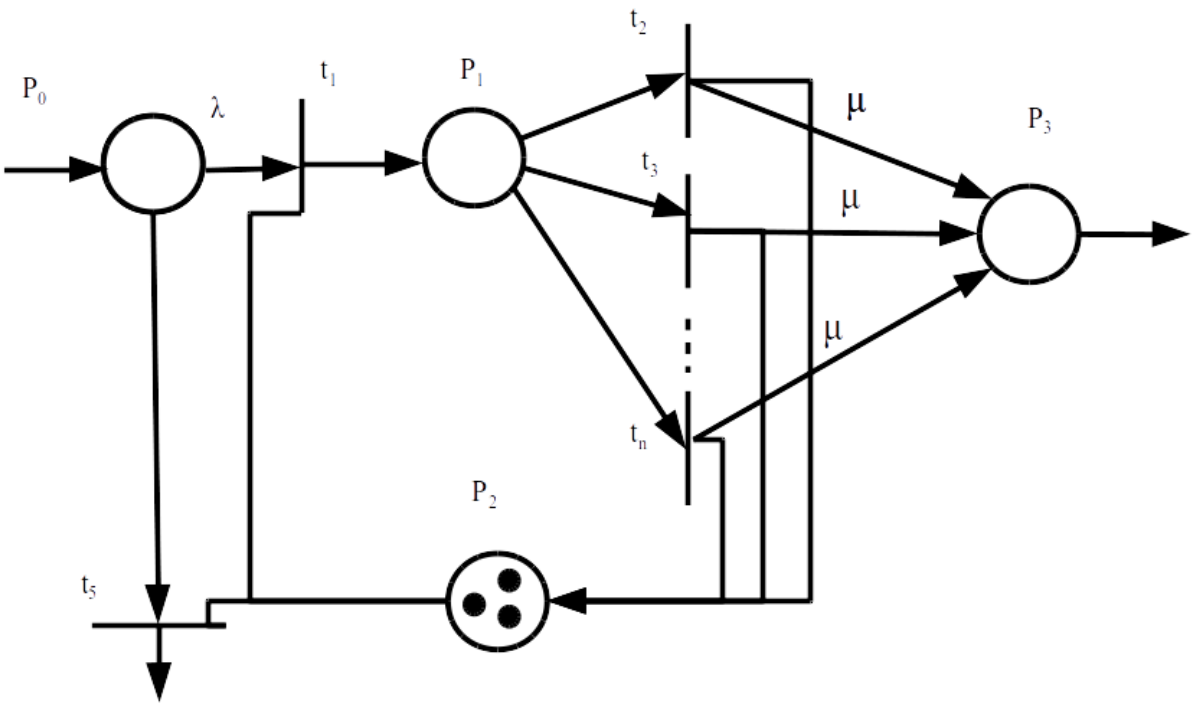

Рисунок 10 – Модель системы с однородным потоком заявок

Для того чтобы преобразовать сеть Петри, нужно построить граф достижимости. На основе графа достижимости обычно исследуются логика функционирования процессов, а также выявляются циклы, находящиеся в модели. Сам граф строиться на основе матрицы. Где матрица это <P, T, I, O, M>. В ней P – это множество вершин; T – это множество переходов; I – это функция входов; O – это функция выходов; M – это матрица маркировки.

После постройки графа достижимости, склеим полученные вершины достижимости, которые связаны отношением перехода модели из одного состояния в другое. В результате проделанных мы получим упрощенный граф достижимости, на котором будет строиться непрерывная цепь Маркова.

Цепи Маркова для модели системы с n каналами обслуживания изображены на рисунке 11.

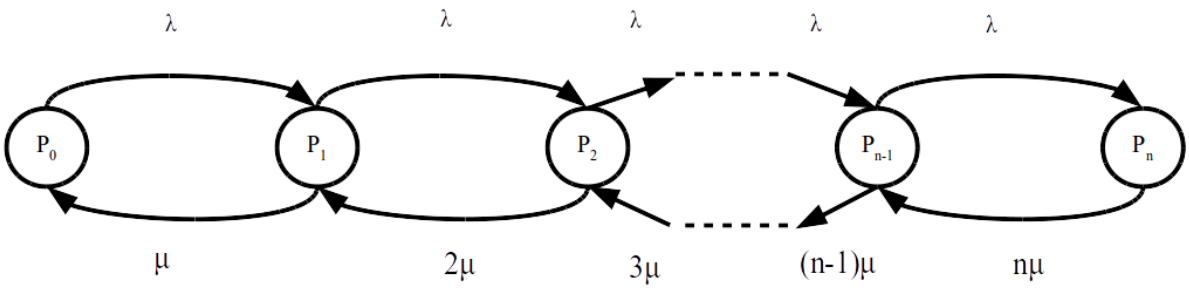

Рисунок 11 – Непрерывная цепь Маркова модели системы с однородным

потоком заявок для n - каналов.

Описать непрерывную цепь Маркова, можно с помощью системы уравнений Колмогорова. Такая система позволит рассчитать основные характеристики системы:

- вероятность простоя, вероятность отказа;
- абсолютную пропускную способность системы;
- среднее число занятых каналов.

Полученные результаты проведенного имитационного моделирования необходимо сравнить с результатами полученными в ходе аналитического расчета. После анализа сравнения необходимо будет откалибровать узлы полученной математической модели.

Результаты имитационного моделирования нужно сравнить с полученными результатами аналитического расчета, и откалибровать узлы полученной математической модели.

Основная идея аналитических методов заключается в расчете характеристик системы, используя математические модели, отражающие свойства реальных систем.

При использовании математического аппарата, который был предложен А.А. Марковым, можно добыть большую часть аналитических результатов, необходимых для анализа системы.

Web-сервер в разрабатываемой информационной системе можно представить как систему массового обслуживания (СМО). Но при этом необходимо принять несколько упрощающих предположений:

- поток заявок в системе является случайным, стационарным, ординарным, однородным, без последствий, без отказов;
- время обслуживания заявок в системе случайное;
- система является многоканальной и разомкнутой.

Большинство математических моделей, описывающих СМО, являются вероятностными. Для того чтобы исследовать изменения вероятностных характеристик в непрерывном времени, можно использовать систему уравнений Колмогорова.

В таком случае примем что, интенсивность заявок на входе системы массового обслуживания будет равна λ, а интенсивность обработки заявок

будет равна μ. Тогда, систему будут описывать следующие характеристики:

- пуассоновский поток заявок на входе системы;
- экспоненциальное распределение времени обслуживания;
- в системе отсутствует очередь заявок.

В таком случае, система уравнения Колмогорова для такой системы будет иметь такой вид:

$$\begin{cases} \frac{dP_0(t)}{dt} = -\lambda P_0(t) + \mu P_1(t); \\ \frac{dP_1(t)}{dt} = -\lambda P_0(t) - (\lambda + \mu) P_1(t) + 2\lambda\mu P_2(t); \\ \dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots. \\ \frac{dP_i(t)}{dt} = \lambda P_{i-1}(t) - (\lambda + i\mu) P_i(t) + (i+1)\lambda\mu P_{i+1}(t); \\ \dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots \\ \frac{dP_n(t)}{dt} = \lambda P_{n-1}(t) + n\mu_n(t); \end{cases},$$

где n – число каналов обслуживания системы; i = 1, 2, ... n-1.

Основным показателем входящего потока требований является его интенсивность λ – среднее число требований, входящих в систему за единицу времени. Обратной величиной является среднее время μ между последовательными моментами поступления требований в обслуживающую систему [17]. В таком случае

$$\rho = \frac{\lambda}{\mu}.$$

Такой режим работы, позволит оценить следующие характеристики системы:

- вероятность простоя;
- вероятность отказа в обслуживании;
- среднее число заявок, обслуженных за заданное время;
- среднее время обслуживания одной заявки;
- среднее время обслуживания заявок одним каналом, поступивших за время Т;
- среднее число занятых каналов;
- система обслуживания является разомкнутой.

В таком случае вероятность простоя (отсутствия заявок в системе) будет иметь вид:

$$P_0 = \frac{1}{\sum_{i=0}^{n-1} \frac{\rho^i}{i!}}.$$

Среднее число заявок, которые поступили за время T считается следующим образом:

$$\underline{N} = \lambda T.$$

При этом среднее время обслуживания одной заявки будет составлять

$$T_c = \frac{1}{\mu};$$

$$T_{\text{обс}} = \frac{\lambda}{\mu} T = \rho T,$$

где $T_{\text{обс}}$ показывает среднее время обслуживания поступающих заявок одним каналом за время T.

Среднее число занятых каналов можно получить следующей формулой:

$$B = \rho(1 - P_n).$$

Предположим, что сервер имеет 15 каналов, тогда n = 15. Среднее число заявок за 1 с примем равным 15. Среднее время обработки одной заявки возьмем равным 0,3 с. Интенсивность обслуживания заявок будет равна μ = 3.

Возьмем интенсивность прибытия заявок в систему равной 15, λ = 15 и интенсивность обслуживания заявок μ = 3. Тогда график переходов процессов, рассчитанный на основе системы уравнений Колмогорова, составленной ранее, представлен на рисунке 12. Исходя из этого графика видно, что вероятность простоя системы составляет P_0= 3,589 * 10^{-3} . При этом видно, что среднее число занятых канало равно 9.

Графики переходных процессов в информационной системе при значениях интенсивностей λ = 15, μ = 3, рассчитанные на основе составленной системы уравнений Колмогорова приведены на рисунке 12. Из результатов следует, что вероятность простоя системы при этом составила P_0 = 3,589 * 10^{-3}, в, среднее число занятых каналов в системе равно 9.

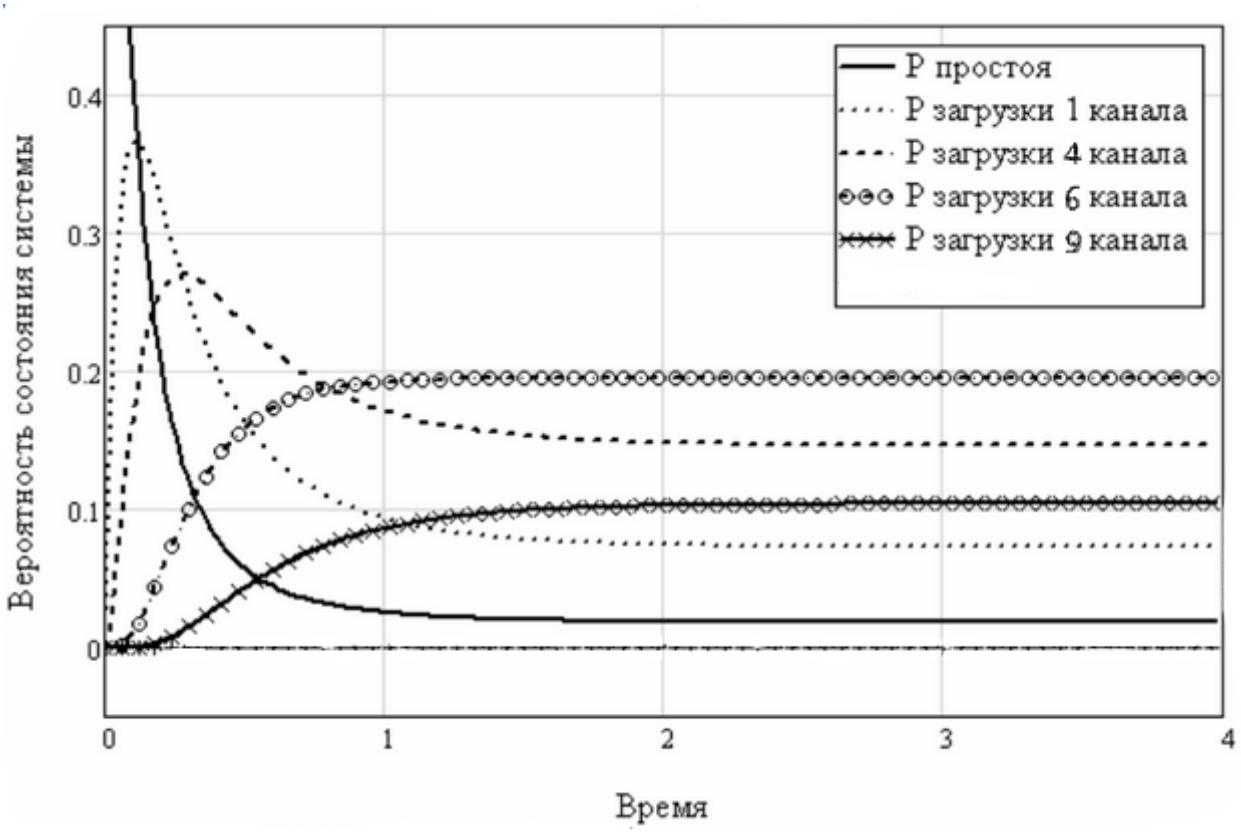

Рисунок 12 – Результаты аналитического расчёта характеристик СМО с 15 каналами

Чтобы получить достоверность результатов имитационного моделирования нужно изучить адекватность, чувствительность, устойчивость полученной модели. Адекватность модели рассчитывается по степени схожести ее с реальным моделируемым объектом или явлением. Для корректного моделирования информационной системы можно используются следующие методы:

– метод проверки преобразований от входа к выходу;

– метод предельных точек;

– метод получения статистически значимых выводов относительно данных имитационного моделирования.

Если имеется возможность использовать имитационные и аналитические методы, то наиболее целесообразным является выбор метода предельных точек. Он позволяет исследовать характеристики адекватности имитационной модели разрабатываемой информационной системы. Данный метод заключается в проведении модельного эксперимента для вариантов исходных данных. Эти исходные данные обеспечивают функционирование имитационной модели в режимах, для которых конечный результат можно

вычислить аналитическими методами. В этом случае адекватность осуществляется в сравнении результатов, которые были получены при имитационном моделировании с результатами полученными при аналитическом моделировании. Сравнение этих результатов приведено в таблице 3.

Таблица 3 – Сравнение результатов аналитического и имитационного моделирования

Данные		Тип моделирования	
Начальные значения	Показатель	Аналитическое	Имитационное
$\lambda = 15$, $\mu = 3$	Среднее время обслуживания заявки	0.3 с	0.36 с
	Загрузка системы	4,999	5,07
	Среднее число заявок	3	3,12
$\lambda = 15$, $\mu = 5$	Среднее время обслуживания заявки	0.2 с	0.2 с
	Загрузка системы	2	2,04
	Среднее число заявок	2.1	2,15

Продолжение таблицы 3

Данные		Тип моделирования	
Начальные значения	Показатель	Аналитическое	Имитационное
λ = 15, μ = 1	Среднее время обслуживания заявки	1 с	1.03
	Загрузка системы	14,621	14,79
	Среднее число заявок	14,7	14,86

Устойчивость полученной модели показывает способность модели сохранять адекватность при исследовании эффективности системы.

На рисунке 13 изображен график зависимости интегрированной интенсивности поступления заявок L от λ для разрабатываемой модели, где λ = 0,1...15 и μ = 0,1...10.

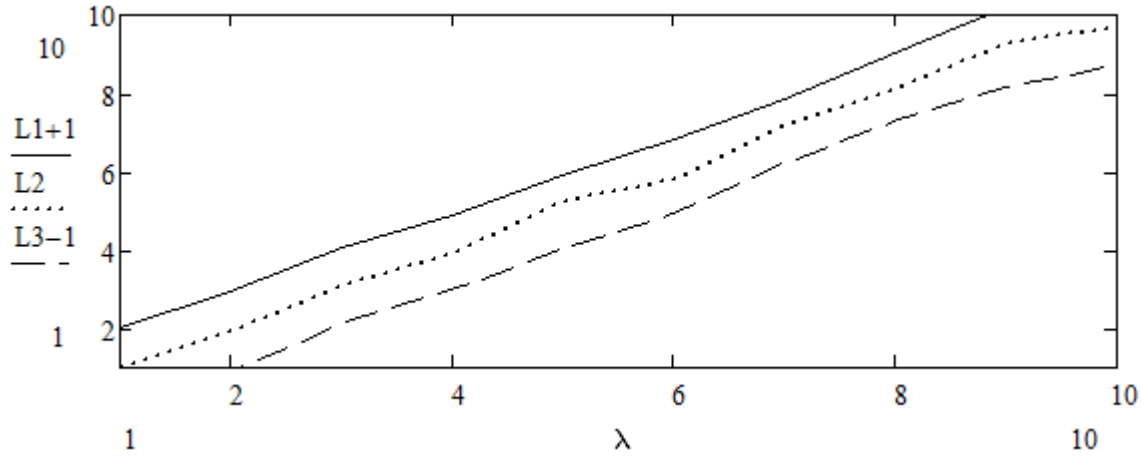

Рисунок 13 – График зависимости интегрированной интенсивности поступления заявок L от λ

На графике L_1 отображена интегрированная интенсивность обслуживания заявок при μ = 1. На графике L_2 и L_3 с μ = 5 и μ = 10 соответственно. Если используется одинаковый масштаб по обеим осям, в идеальной модели графики должны представлять собой линию с углом наклона 45°, т.к. в такой модели L стремиться по своему значению к λ.

Интегрированная интенсивность L поступления заявок рассчитывается следующим образом:

$$L_c = \frac{N_c}{T_c},$$

где N_c – среднее число заявок в узле обслуживания, T_c – среднее время обслуживания заявки.

По графику на рисунке 13 можно сделать об устойчивости имитационной модели.

2.2 Построение и исследование имитационной модели

Была собрана статистика имитирующая нагрузку, сопоставимую с использованием системы 30 пользователями. Число одновременных соединений равно 15, поэтому n = 15. Данный показатель не меняется на протяжении всего исследования.

Интенсивность поступления заявок на обслуживание различаются в зависимости от типа клиента. Например, для тонкого клиента интенсивность составила $\lambda = 13$ и $\lambda = 9$ для толстого клиента. Т.к. толстый клиент позволяет выполнять часть функционала, то уменьшается необходимость обращения к серверу. Как показывает статистика, среднее время обработки одной заявки равно 0,2 с. Средняя интенсивность обслуживания заявок при этом появляется $\mu = 5$.

После анализа собранной статистики работы тонкого клиента были получены следующие результаты. Интенсивность прихода заявок на получение статического содержания страницы $\lambda_1 = 4$, среднее время обслуживания $t_1 = 0.132$ и интенсивность обслуживания была равна $\mu_1 = 7$. Заявка на получение информации из базы данных приходят с интенсивностью $\lambda_2 = 5$, среднее время обслуживания составило $t_2 = 0{,}18$ с и интенсивность обслуживания была равна $\mu_2 = 5{,}5$. Интенсивность поступления в систему заявок на ввод данных в базу данных, исходя из статистики имитации, равнялась $\lambda_3 = 4$, среднее время обслуживания составило $t_3 = 0.172$ с, а интенсивность обслуживания равна $\mu_3 = 5{,}8$.

Анализ взаимодействия сервера с толстым клиентом выявил следующие данные. Интенсивность получения статической информации $\lambda_1 = 1$, заявки на получение информации из базы данных с интенсивностью $\lambda_2 = 5$, интенсивность заявок на модификацию информации $\lambda_3 = 3$. Среднее время обслуживания $t_1 = 0.15$ с, $t_2 = 0.21$ с, $t_3 = 0.25$ с, а интенсивность обслуживания составила $\mu_1 = 7.7$, $\mu_2 = 5$, $\mu_3 = 5$ соответственно для каждого типа заявки.

На рисунке 14 приведена имитационная модель, собранная в системе AnyLogic. Добавлены 3 источника заявок, которые генерируют заявки с указанной интенсивностью. Далее заявка попадает в блок

Chose_chanel_block. Данный блок ищет свободное множество из 5 каналов. Как только находится свободный канал, то найденный канал берет на себя обслуживание заявки и помечается как занятый.

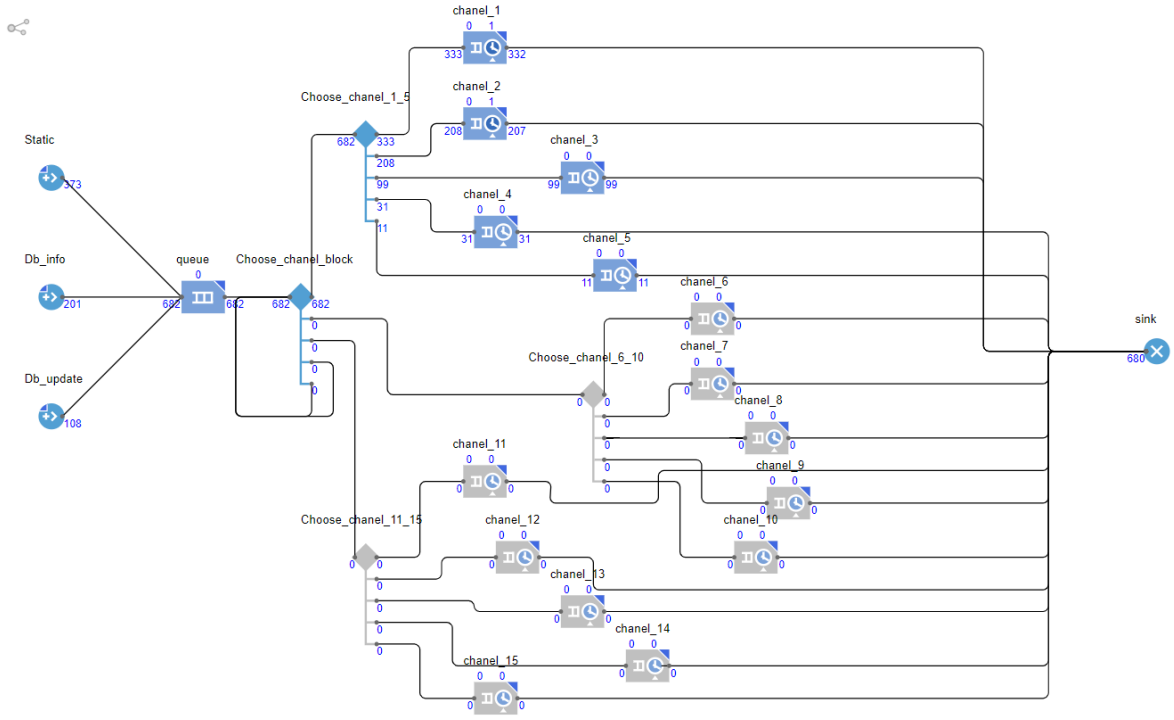

Рисунок 14 – Построенная имитационная модель в AnyLogic

На рисунке 15 представлен график трафика заявок на выходе узла обслуживания с следующими интенсивностями поступления заявок: $\lambda_1 = 4$, $\lambda_2 = 5$, $\lambda_3 = 4$ для тонкого клиента, и интенсивностями обслуживания $\mu_1 = 6$, $\mu_2 = 2.1$, $\mu_3 = 5.8$.

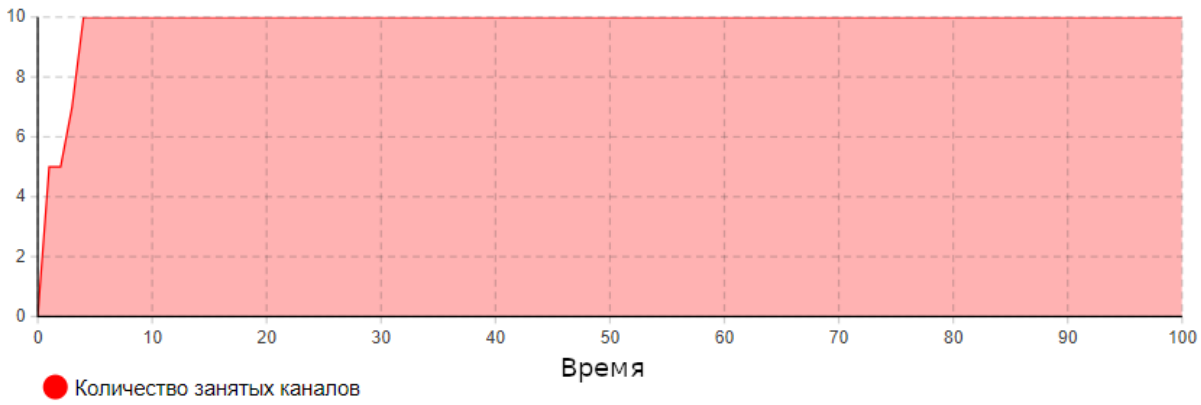

Рисунок 15 – График трафика на выходе узла обслуживания для тонкого клиента

Как показано на рисунке, при заданных характеристиках для тонкого клиента, каналы с 10 по 15 остаются не загруженными. Среднее время обслуживания заявок в системе вне зависимости от раскраски

$T_c = 0,213 \pm 0,024$ с. Среднее число заявок в системе $N_c = 5,217 \pm 0,421$ с. Среднее число одновременно используемых каналов в системе равно 5. Интегрированная интенсивность поступления заявок в систему $L = 24,492 \pm 0,146$.

На рисунке 16 представлен график трафика с интенсивностями поступления заявок $\lambda_1 = 1$, $\lambda_2 = 6$,$\lambda_3 = 3$ для толстого клиента, и интенсивностями обслуживания $\mu_1 = 7,7$, $\mu_2 = 5$, $\mu_3 = 5$.

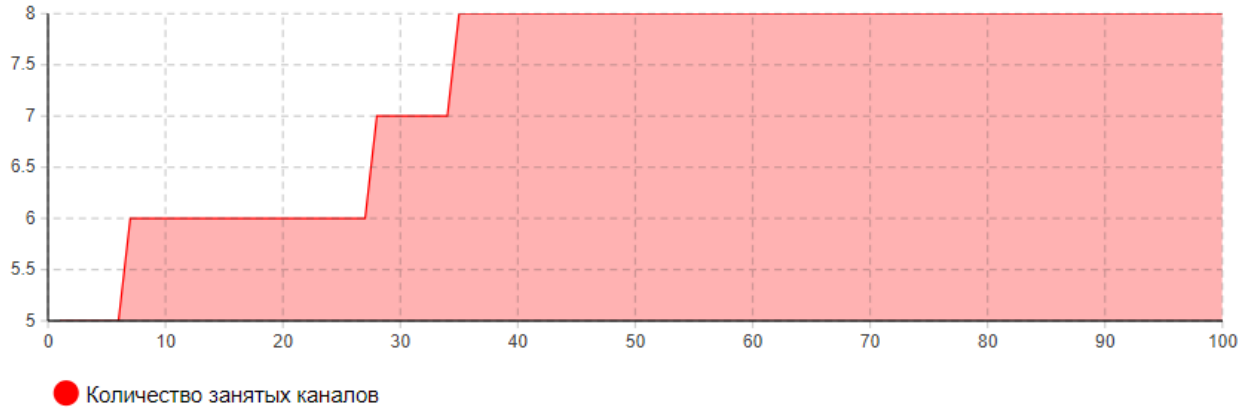

Рисунок 16 – График трафика на выходе узла обслуживания для толстого клиента

Для толстого клиента не загруженными остаются каналы с 8 по 15. Среднее число заявок в системе $Nc = 3,217 \pm 0,418$. Среднее время обслуживания заявок в системе $Tc = 0,253 \pm 0,024$ Интегрированная интенсивность поступления заявок в систему $L = 12,715 \pm 0,239$.

В ходе опытной эксплуатации была выявлена устойчивость функционирования системы. В результате проведения нагрузочного тестирования системы была проведена корректировка распределения нагрузки между клиентом и сервером информационной системы в целях повышения отказоустойчивости системы.

Выводы

После проведенного этапа проектирования, выявлено, что осуществить оценку параметров разрабатываемой информационной системы контроля за общественным транспортом и пассажиропотоком без предварительного проведения имитационного моделирования системы невозможна. Необходимо обеспечить выполнение требований к имитационной модели по

адекватности, устойчивости и непрерывности.

Все заявки в модели разрабатываемой информационной системе можно разделить на 3 типа в зависимости от выполняемых операций при обработке поступающей информации. У каждого типа есть свои интенсивности возникновения и обслуживания заявок. В имитационной модели информационной системы типы заявок могут быть учтены с использованием раскраски для маркеров сети Петри.

Использование толстого клиента в информационной системе контроля общественного транспорта и пассажиропотока уменьшает интенсивность поступления заявок, что, при сохранении интенсивностей обслуживания, обеспечивает меньшую загрузку сервера и повышает отказоустойчивость системы.

3 Разработка информационной системы контроля общественного транспорта и пассажиропотока

3.1 Архитектура информационной системы контроля общественного транспорта и пассажиропотока

В большинстве современных приложений появляются элементы клиент – серверного взаимодействия. Очень популярная двухзвенная архитектура. Двухзвенной она называется из-за распределения трех компонентов между двумя (клиентом и сервером). На рисунке 17 изображена двухзвенная клиент-серверная архитектура.

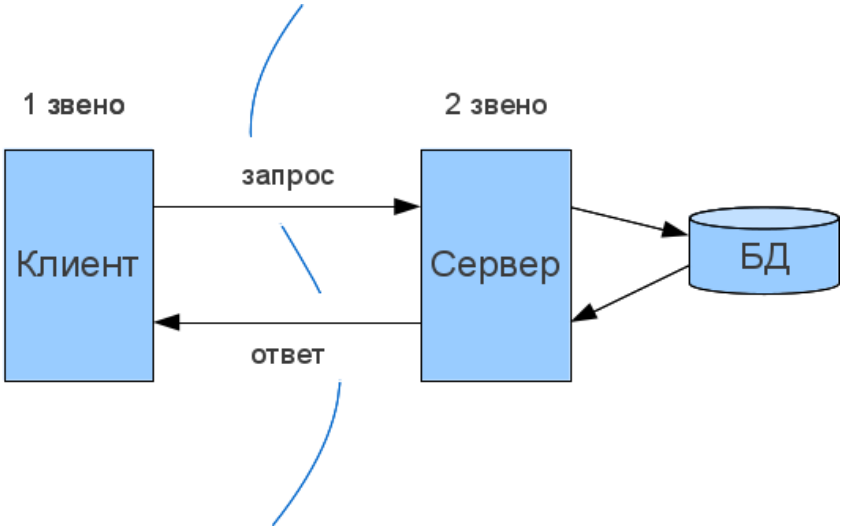

Рисунок 17 – Двухзвенная клиент-серверная архитектура

Двухзвенная архитектура зачастую используется в клиент-серверных приложениях, где сервер отвечает на клиентские запросы напрямую, используя только собственные ресурсы. Это значит что сервер не вызывает сторонние сетевые приложения для выполнения запроса.

На рисунке 18 приведен пример модели клиент-серверного взаимодействия.

Клиент-серверная архитектура используется в множестве сетевых технологий, используемых для доступа к различным сетевым сервисам. Из них можно выделить несколько видов:

- Web-серверы первоначально представляли доступ к текстовым документам по протоколу HTTP, но в данный момент поддерживают расширенные возможности;
- Серверы приложений используются для решения прикладных

задач. Являются промежуточным звеном между клиентом и сервером базы данных. Использование серверов приложений позволяет снизить требования к конфигурации клиентов и упрощает общее управление сетью;

– Серверы баз данных используется для обслуживания и управлением базой данных и для обработки запросов на языке SQL.

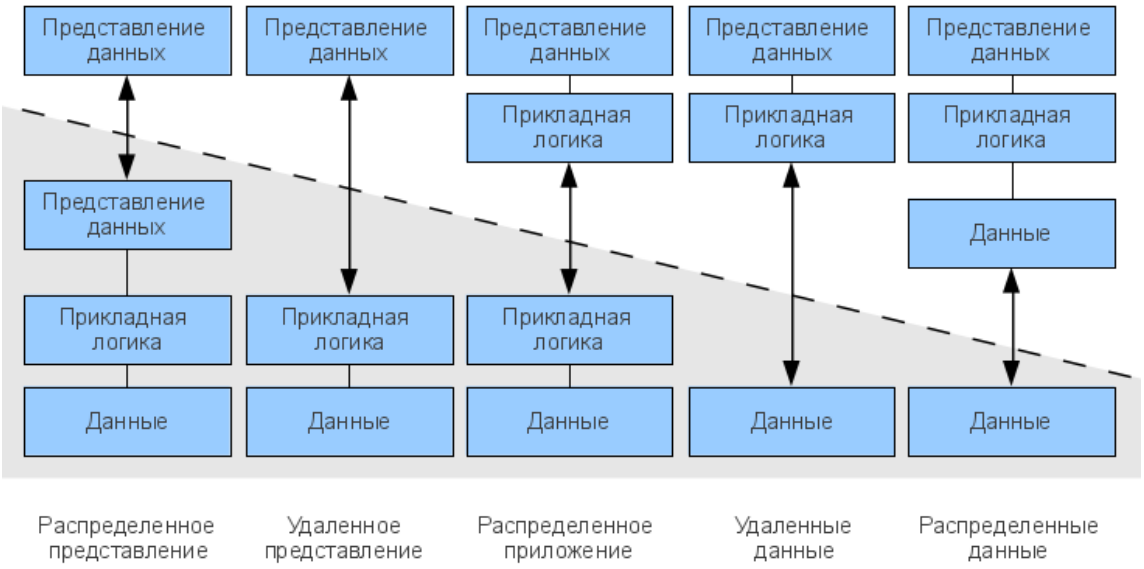

Рисунок 18 – Модели клиент-серверного взаимодействия

К тем или иным сетевым сервисам могут подключаться различные клиенты. Клиенты могут различаться аппаратным обеспечением и программным обеспечением. Все клиенты можно разделить на два типа, «тонкий» и «толстый» клиент.

«Тонкий» клиент – это клиент у которого вычислительные возможности позволяют только запускать сетевые приложения через web-интерфейс. Интерфейс такого приложения реализуется с помощью средств HTML, а вся логика приложения выполняется на стороне сервера. Для функционирования «тонкого» клиента достаточно обеспечить запуск web-браузера на стороне клиента. Через окно web-браузера и будут производиться все действия клиента. Из-за этой причине web-браузер называют «универсальным клиентом».

«Толстый» клиент – это клиент имеющий вычислительные мощности, которые позволяет выполнять часть функций сервера на стороне клиента, уменьшая при этом нагрузку на сервер. Зачастую такие клиенты работают под управлением собственной операционной системы. Как правило «толстый» клиент обращается к серверу только за обновлением базы данных.

Так же такие клиенты могут совмещать графический пользовательский интерфейс и вычислительные мощности клиентского устройства.

Разрабатываемое приложение имеет многоуровневую архитектуру. Сервер с «толстым» и «тонким» клиентами.

Приложение имеет многоуровневую архитектуру клиент сервер с тонким и толстым клиентами. Пользователь может взаимодействовать с сервером с помощью «толстого» или «тонкого» клиента. «Толстый» клиент может работать автономно. Необходимая для этого база данных находится на стороне клиента. Эта база данных обновляется при запуске приложения или при изменении данных на центральной базе данных.

На рисунке 19 представлена диаграмма развертывания информационной системы контроля общественного транспорта и пассажиропотока с толстым и тонким клиентами. Диаграмма состоит из сервера приложений, сервер базы данных. Толстого клиента, тонкого клиента использующих web-браузер. Мобильного приложения реализующего толстый клиент.

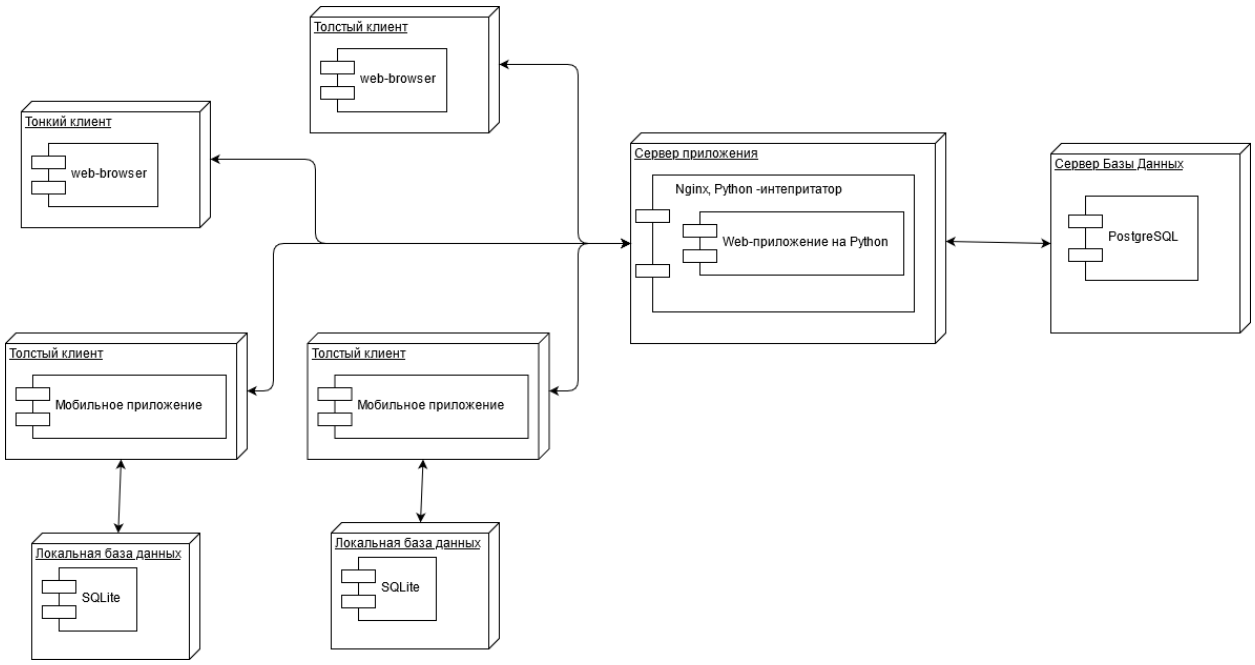

Рисунок 19 – Диаграмма развертывания информационной системы контроля общественного транспорта и пассажиропотока с толстым и тонким клиентами

На диаграмме развертывания информационной системы показано взаимодействия сервера приложения с СУБД, тонкими и толстым клиентами.

3.2 Разработка базы данных системы

База данных – это совокупность данных, организованных в соответствии с концептуальной структурой, описывающей характеристики этих данных и взаимоотношения между ними. Существует большое количество разных типов баз данных. При разработке будет использоваться реляционная база данных [18].

Реляционная база данных – это база данных, основанная на реляционной модели данных. Реляционная модель включает в себя следующие компоненты:

- структурный аспект (составляющая) – данные в базе данных представляют собой набор отношений;
- аспект целостности – отношения отвечают определенным условиям целостности. Реляционная модель данных (РМД) поддерживает декларативные ограничения целостности уровня домена, уровня отношения и уровня базы данных;
- аспект обработки – РМД поддерживает операторы манипулирования отношениями (реляционная алгебра, реляционное исчисление) [19].

3.2.1 Разработка логической модели базы данных

После анализа предметной области была разработана логическая модель базы данных разрабатываемой информационной системы. Для разработки была использована программная среда Aqua Data Studio позволяющая моделировать, создавать и эксплуатировать базу данных.

На рисунке 20 изображена логическая модель базы данных сервера.

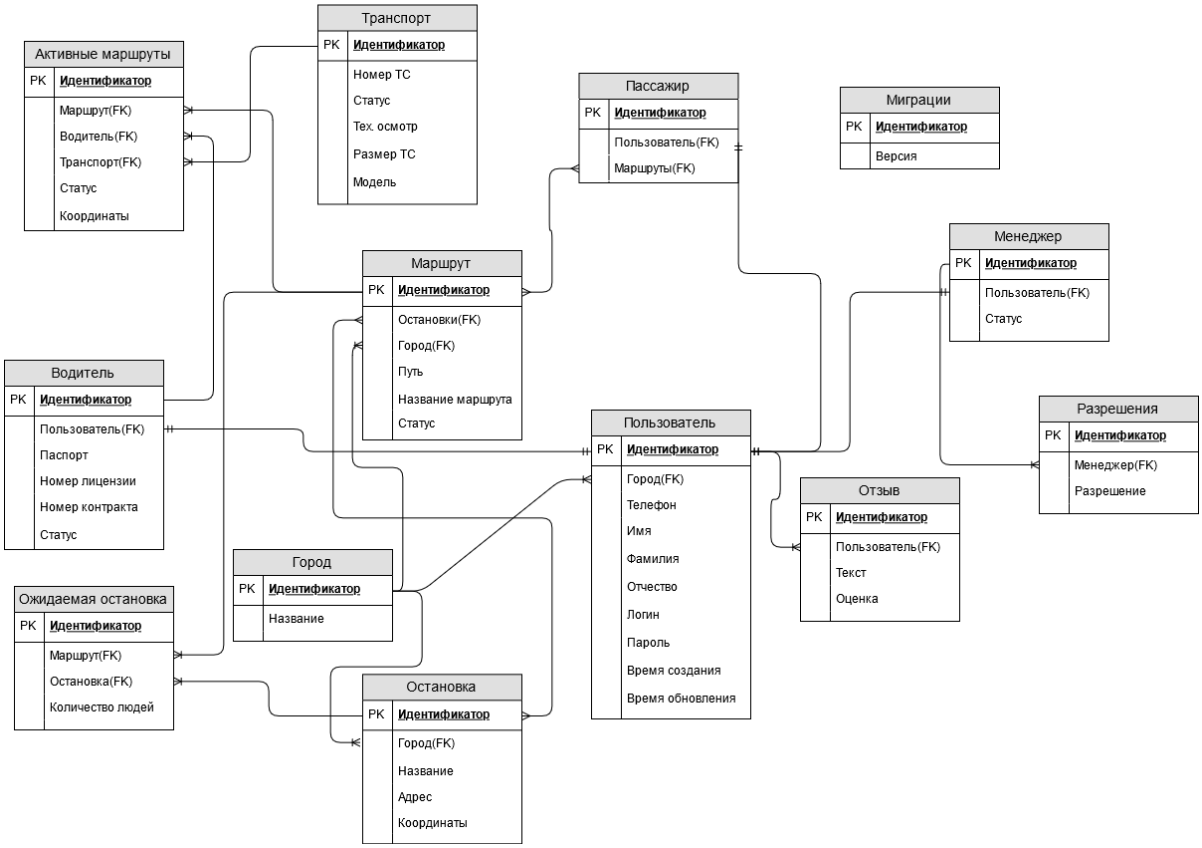

Рисунок 20 – Логическая модель базы данных сервера

База данных мобильного приложения содержит сущности необходимые для автономной работы на мобильном устройстве. Структуры сущностей незначительно отличаются от сущностей базы данных централизованного сервера. На рисунке 21 приведена логическая модель базы данных мобильного приложения.

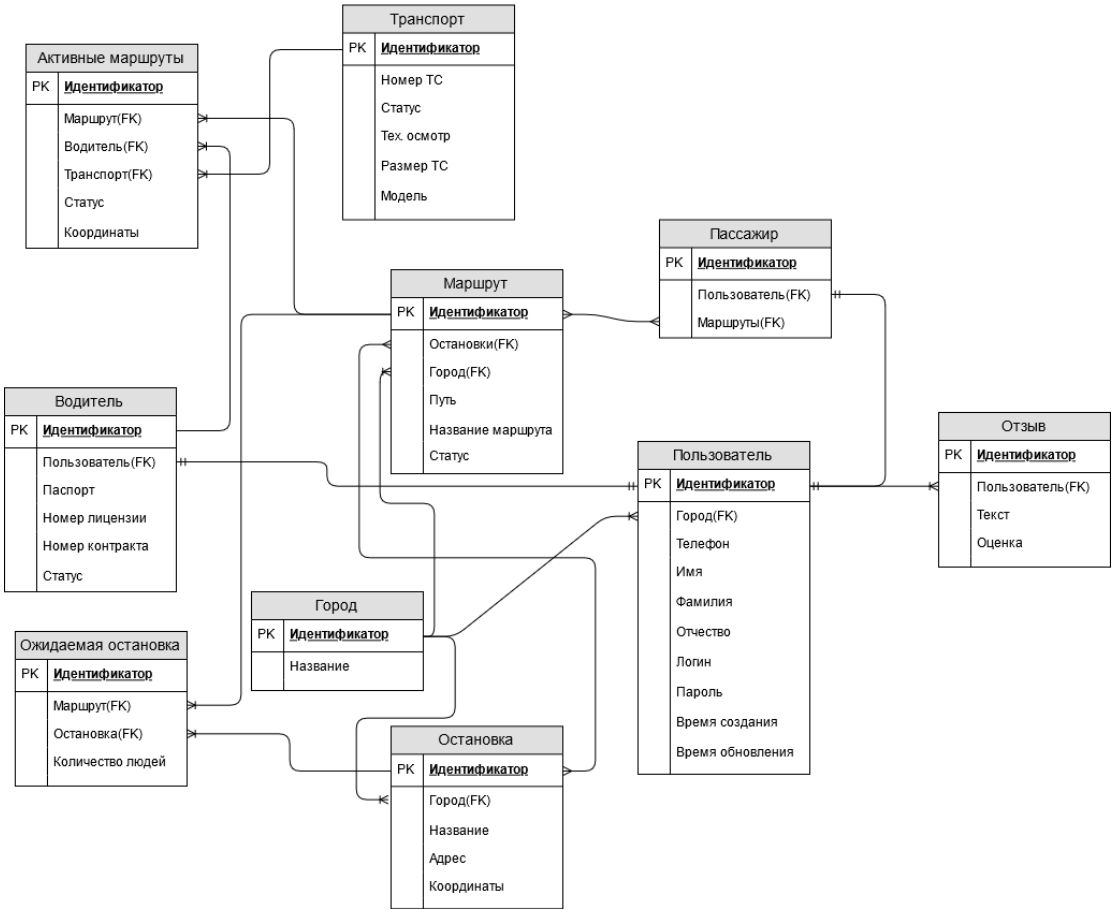

Рисунок 21 – Логическая модель базы данных мобильного приложения

Устранение избыточности модели производится, в большинстве случаев с помощью декомпозиции отношений. Декомпозиция происходит таким образом, чтобы в каждом отношении хранились только первичные факты. В реляционной модели, логическая модель это набор схем отношений с первичными ключами и связями между отношениями, выражающимся в внешних ключах. При проектировании реляционных баз данных обычно выполняется так называемая нормализация.

3.2.2 **Анализ уровня нормализации логической модели базы данных**

Нормализация базы данных. Нормальная форма – свойство отношения в реляционной модели данных, характеризующее его с точки зрения избыточности, которая потенциально может привести к логически ошибочным результатам выборки или изменения данных. Нормальная форма определяется как совокупность требований, которым должно удовлетворять отношение [18].

Нормализацией называется процесс преобразований отношений базы

данных к нормальным формам. Она не осуществляется для увеличения производительности работы или уменьшения физического объёма базы данных. Нормализация предназначена для обеспечения минимальной логической избыточности базы данных и уменьшении противоречивости информации в базе данных.

Процесс преобразования отношений базы данных к виду, отвечающему нормальным формам, называется нормализацией. Нормализация предназначена для приведения структуры базы данных к виду, обеспечивающему минимальную логическую избыточность, и не имеет целью уменьшение или увеличение производительности работы или же уменьшение или увеличение физического объёма базы данных. Конечной целью нормализации является уменьшение потенциальной противоречивости, хранимой в базе данных информации. Общее назначение процесса нормализации заключается в следующем:

- устранение аномалий обновления;
- исключение типов избыточности;
- разработка базы данных, являющимся достаточно «качественным» представлением реального мира;
- упрощение процедуры применения необходимых ограничений целостности.

Устранение избыточности производится, как правило, за счёт декомпозиции отношений таким образом, чтобы в каждом отношении хранились только первичные факты.

Хотя нормализация полезна для баз данных, она не является универсальным способом повышения качества проекта базы данных. Такое происходит из-за разнообразия возможных ошибок. Существует три основных нормальные формы.

Первая нормальная форма (1NF). Переменная отношения находится в первой нормальной форме (1НФ) тогда и только тогда, когда в любом допустимом значении отношения каждый его кортеж содержит только одно значение для каждого из атрибутов. В реляционной модели отношение всегда находится в первой нормальной форме по определению понятия отношение. Что же касается различных таблиц, то они могут не быть правильными

представлениями отношений и, соответственно, могут не находиться в 1НФ.

Вторая нормальная форма (2NF). Переменная отношения находится во второй нормальной форме тогда и только тогда, когда она находится в первой нормальной форме и каждый не ключевой атрибут неприводимо зависит от ее потенциального ключа.

Третья нормальная форма (3NF). Переменная отношения находится в 3NF тогда и только тогда, когда она находится во второй нормальной форме и отсутствуют транзитивные зависимости не ключевых атрибутов от ключевых [20].

Разработанная логическая модель базы данных не содержит транзитивных зависимостей. Таким образом, можно сделать вывод, что модель соответствует третьей нормальной форме.

3.2.3 Разработка физической модели базы данных

При моделировании физической модели базы данных осуществляется переход от логических сущностей к физическим таблицам. При построении физической модели должна учитывается целевая СУБД. В качестве СУБД используется программный продукт PostgreSQL.

Перевод в физическую модель осуществляется согласно следующим правилам:

- объекты становятся таблицами в физической базе данных;
- атрибуты становятся колонками в физической базе данных;
- для каждой колонки выбирается подходящий тип данных;
- уникальные идентификаторы превращаются в колонки, не допускающими значение NULL;
- отношения между сущностями моделируются в виде внешних ключей.

На рисунке 22 изображена физическая модель базы данных сервера приложения.

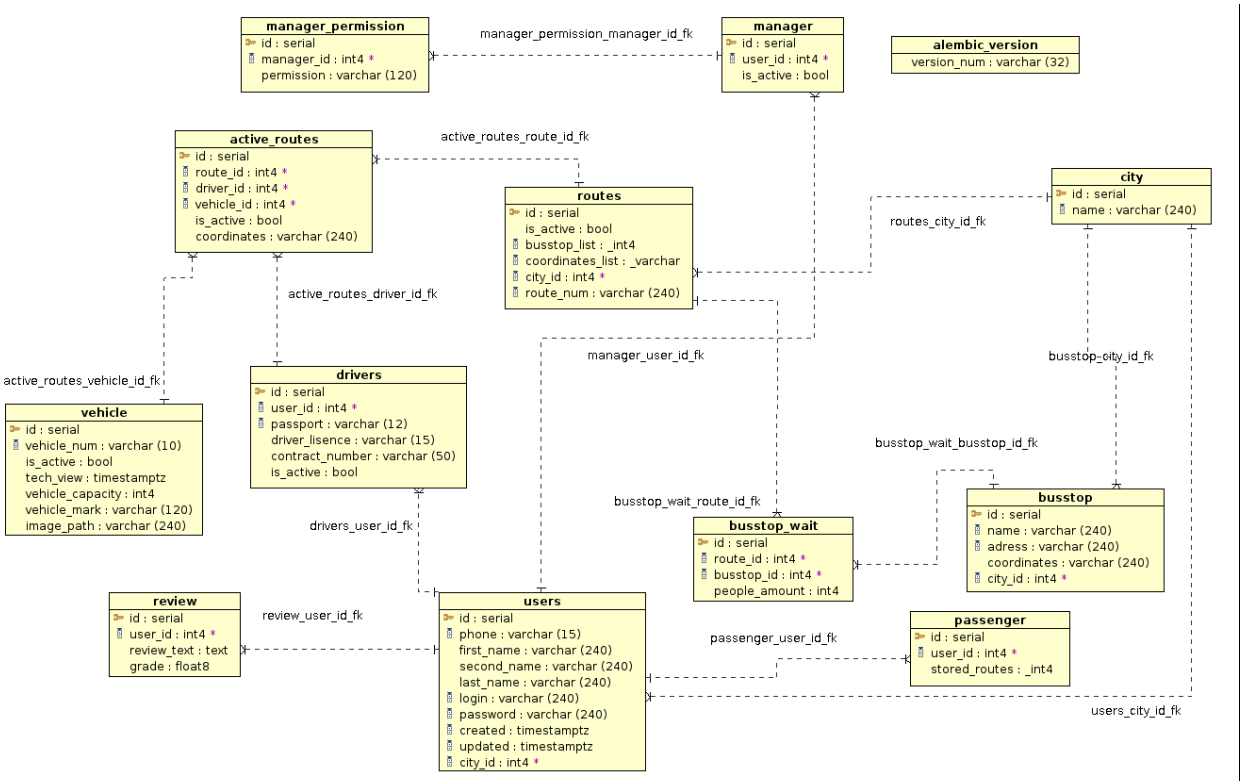

Рисунок 22 – Физическая модель базы данных сервера приложения

База данных состоит из 13 таблиц, которые связаны внешними ключами. В таблице users храниться информация о пользователях, на эту таблицы ссылаются другие более специализированные таблицы, drivers, passenger, manager, которые представляют специфические данные водителя, пассажира и сотрудника контроля маршрутов. На рисунке 23 изображена физическая модель базы данных мобильного приложения. База данных мобильного приложения не сильно отличается от базы расположенной на сервере системы.

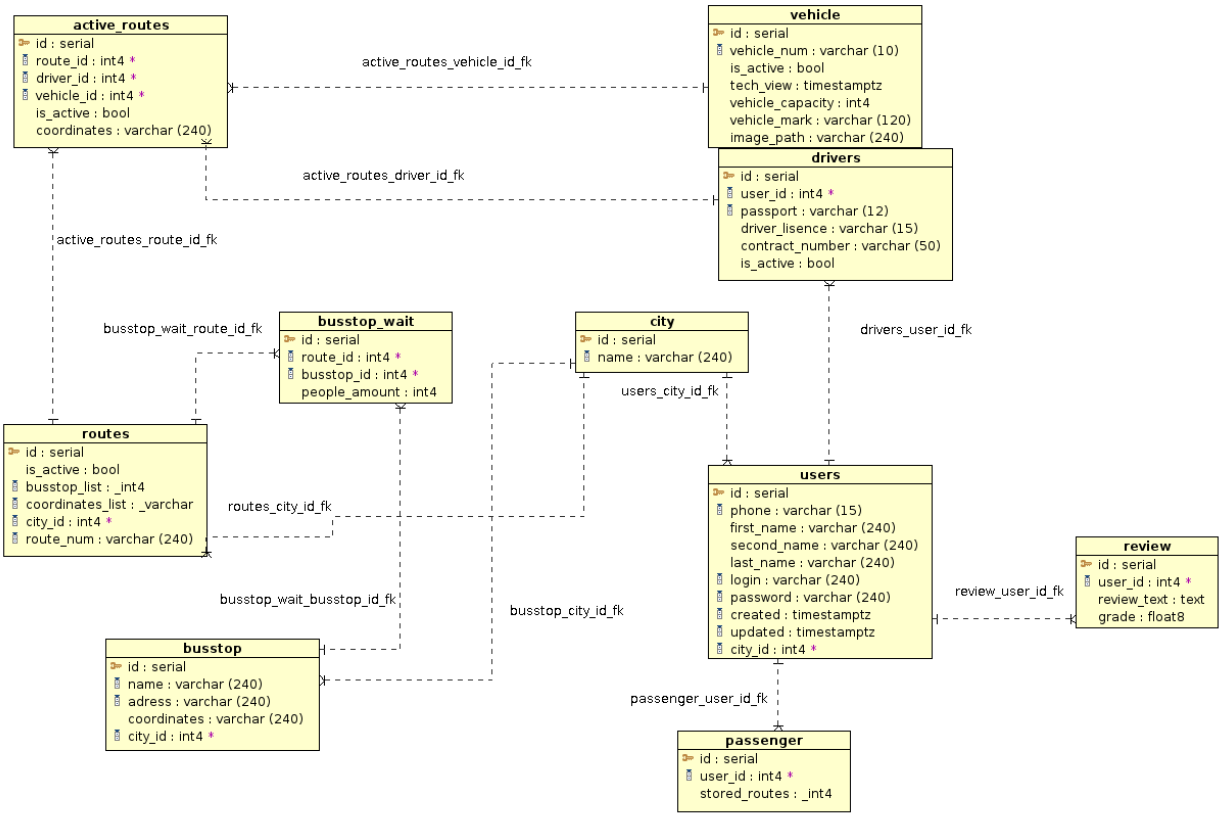

Рисунок 23 – Физическая модель базы данных мобильного приложения

Рисунки 22 и 23 наглядно демонстрируют структуру разработанных баз данных.

3.2.4 Пользовательский интерфейс клиентского приложения

Исходя из диаграммы вариантов использования и анализа требований, был спроектирован понятный графический пользовательский интерфейс. Ниже представлен графический интерфейс некоторых частей информационной системы.

На рисунке 24 изображен графический интерфейс окна авторизации в систему

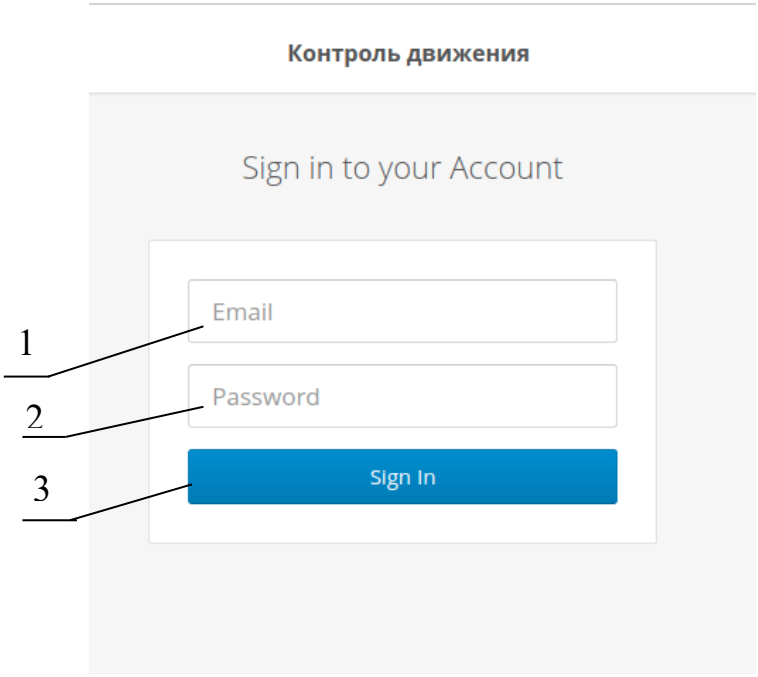

Рисунок 24 – Графический интерфейс окна авторизации

Окно авторизации содержит следующие элементы графического интерфейса:

1 – поле для ввода почты для авторизации;

2 – поле для ввода пароля;

3 – кнопка авторизации.

На рисунке 25 изображена страница заполнения информации о водителе.

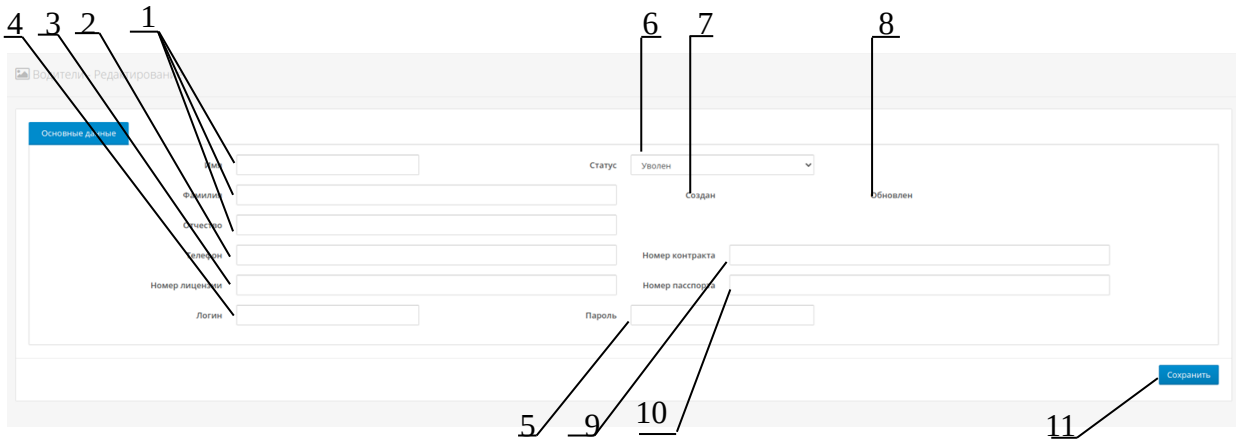

Рисунок 25 – Страница заполнения информации о водителе

Страница заполнения информации о водителе содержит следующие элементы:

1 – поля для ввода ФИО водителя;

2 – поле для ввода телефона;

3 – поле содержащее номер водительской лицензии;

4 – поле с логином пользователя;

5 – поле с паролем пользователя;

6 – поле-список с выбором статуса водителя;

7 – время создания пользователя;

8 – время последнего обновления записи;

9 – номер контракта водителя;

10 – номер паспорта;

11 – кнопка для сохранения информации в базе данных;

На рисунке 26 представлена страница добавления остановки базу данных.

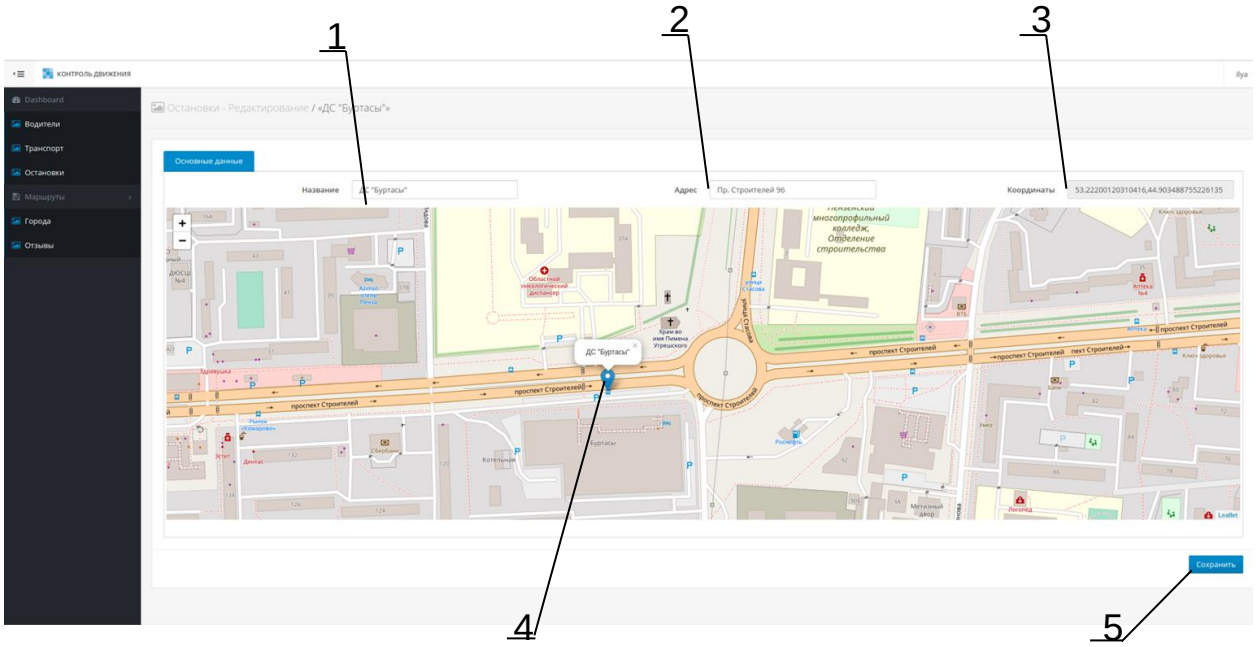

Рисунок 26 – Страница редактирования остановки

Рассмотрим основные элементы интерфейса страницы редактирования остановки:

1 – поле с названием остановки;

2 – поле с адресом остановки;

3 – поле с координатами остановки, заполняется автоматически при выборе точки на карте;

4 – иконка остановки на карте. Она добавляется при нажатии на любое место на карте;

5 – кнопка сохранения информации в базе данных.

На рисунке 27 изображена страница редактирования маршрута.

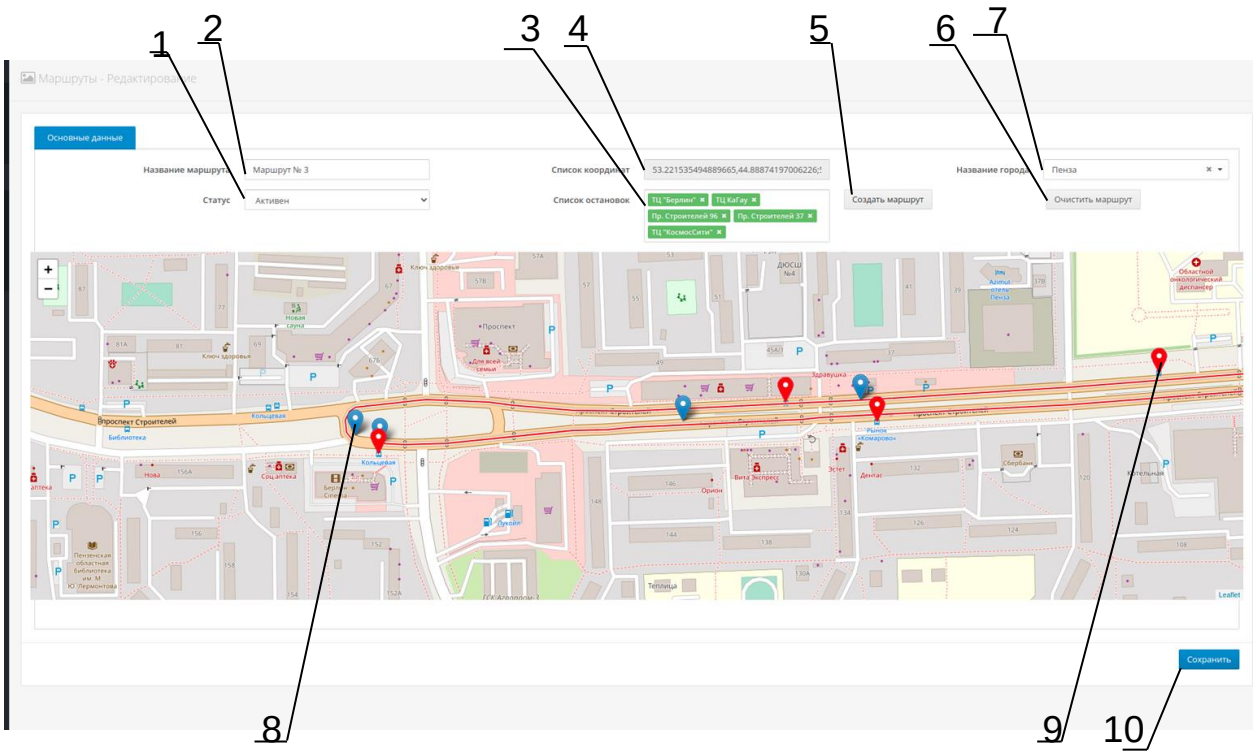

Рисунок 27 – Страница редактирования маршрута

Окно редактирования маршрута содержит следующие элементы:

1 – поле с выбором статуса маршрута;

2 – поле с названием маршрута;

3 – поле с выбором списка остановок;

4 – список координат точек маршрута, в поле нельзя осуществить ввод данных, они заполняются автоматически;

5 – поле с выбором активного города;

6 – кнопка заполнения поля номер 4;

7 – кнопка очищающая весь построенный маршрут на карте;

8 – иконка обозначающая точку в маршруте, появляются при нажатии на карту;

9 – иконка отображающая добавленную остановку в маршрут;

10 – кнопка сохранения информации о маршруте в базе данных.

На рисунке 28 изображено окно мобильного приложения у пользователя водитель.

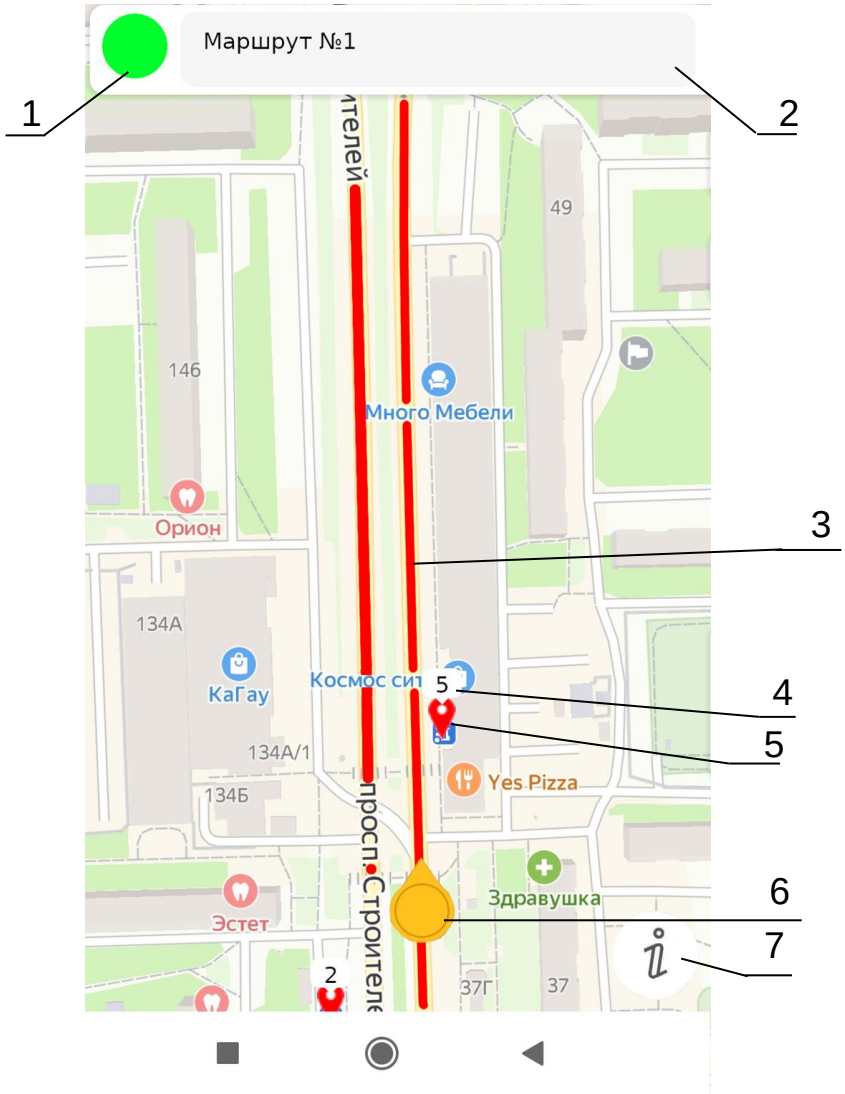

Рисунок 28 – Окно мобильного приложения водителя

Окно мобильного приложения водителя имеет следующие элементы

1 – иконка статуса маршрута;

2 – поле с названием маршрута;

3 – маршрут отображающийся на карте;

4 – количество пассажиров ожидающих данный маршрут на остановке;

5 – метка остановки, занесенная в маршрут;

6 – маркер водителя общественного транспорта;

7 – кнопка вызова меню.

3.3 Разработка программного обеспечения

3.3.1 Разработка программного обеспечения серверной части

Разработка программного обеспечения серверной части велась на основе фреймворка Tornado.

Любое приложение Tornado как правило, состоит из, собственно, класса приложения и классов хэндлеров (обработчиков) запросов При использовании Tornado не приходится работать с HTTP напрямую - разбор заголовков и URL фреймворк берет на себя. От разработчика требуется лишь словарь, состоящий из регулярных выражений и соответствующих им классов-обработчиков запросов (хендлеров). Такая реализация позволяет использовать методы ООП, в частности наследование для упрощения обработки запросов. Также в Tornado присутствует встроенный шаблонизатор, но при необходимости можно подключить любой другой.

Рассмотрим последовательность действий более подробно:

– пользователь осуществляет запрос посредством URL http://www.example.com/busstops;

– веб-сервер ищет подходящий путь URL в списке связанных хендлеров;

– далее находиться подходящий route и вызывается соответствующий ему хендлер;

– далее запрос передается в найденный метод подходящего хендлера;

– происходит обращение в метод get хендлера;

– происходит обращение к классу model для работы с СУБД;

– делается запрос на получение данных из базы данных;

– полученная информация возвращается в метод хендлера;

– хендлер подключает некоторые виджеты;

– сформированное данные вставляется в макет страницы;

– хендлер завершает формирование страницы и выводит результат пользователю.

На рисунке 29 изображена последовательность работы приложения в фреймворке Tornado.

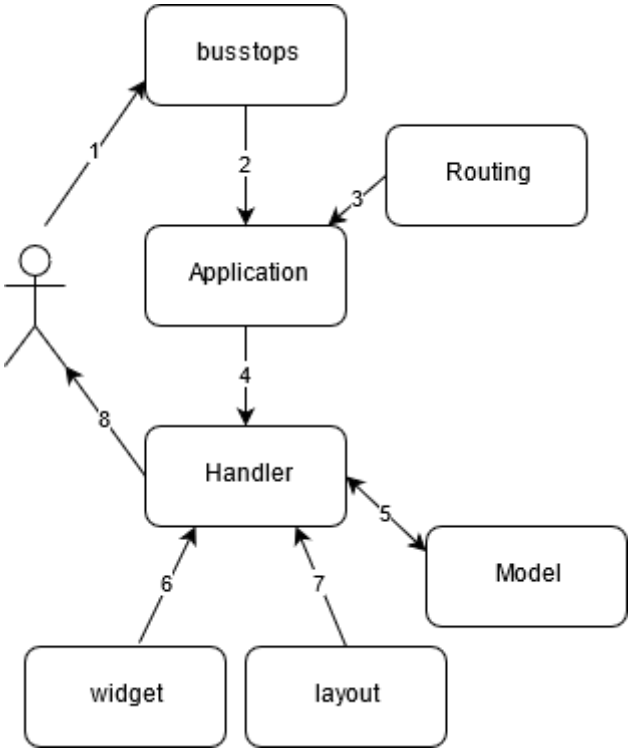

Рисунок 29 – Последовательность работы приложения Tornado

На рисунке 30 представлена структура директории приложения сервера. В модуле Static содержаться все статические ресурсы, такие как картинки, css-файлы, основные js-файлы.

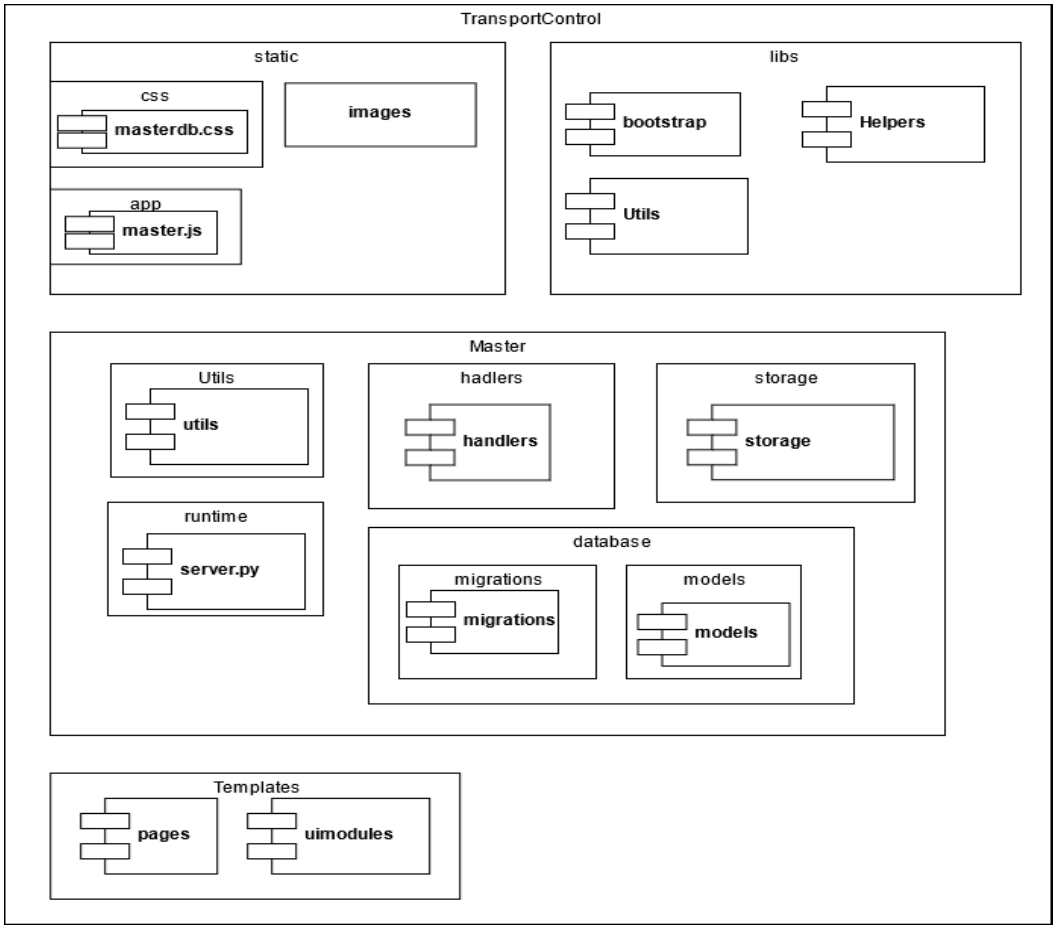

Рисунок 30 – Диаграмма компонентов приложения сервера

Рассмотрим диаграмму последовательности просмотра и редактирования профиля водителя. Данная диаграмма изображена на рисунке 31.

На данной диаграмме показано, что клиент обращается для получения страницы с данными водителя. Этот запрос передаётся в хендлер DriversEditHandler, который отвечает за отображение и редактирование данных водителя. Далее DriversEditHandlers передаёт запрос на получение информации о водителе через метод create_or_update класс AsyncDrivers. Данный класс отвечает за реализацию асинхронных запросов в базу данных для получения или изменении данных водителя. В данном случае у класса AsyncDrivers метод create_or_update, не переопределен, поэтому вызывается такой же метод родительского класса AsyncStorageBase. Этот класс делает запрос в базу данных и возвращает полученный результат. При получении результата из базы данных, хендлер DriversEditHandler добавляет данные в макет страницы от отправляет её обратно пользователю.

Если пользователь хочет изменить данные, то происходит аналогичная последовательность действий. Измененные данные сначала отправляются в хендлер DriversEditHandlers, затем в AsyncStorageBase и уже после этого происходит модификация базы данных. Результат выполнения запроса отправляется обратно хендлеру, при успешном выполнении, хендлер обновляет страницу с новыми данными, при ошибке, выводит на страницу сообщение об ошибке.

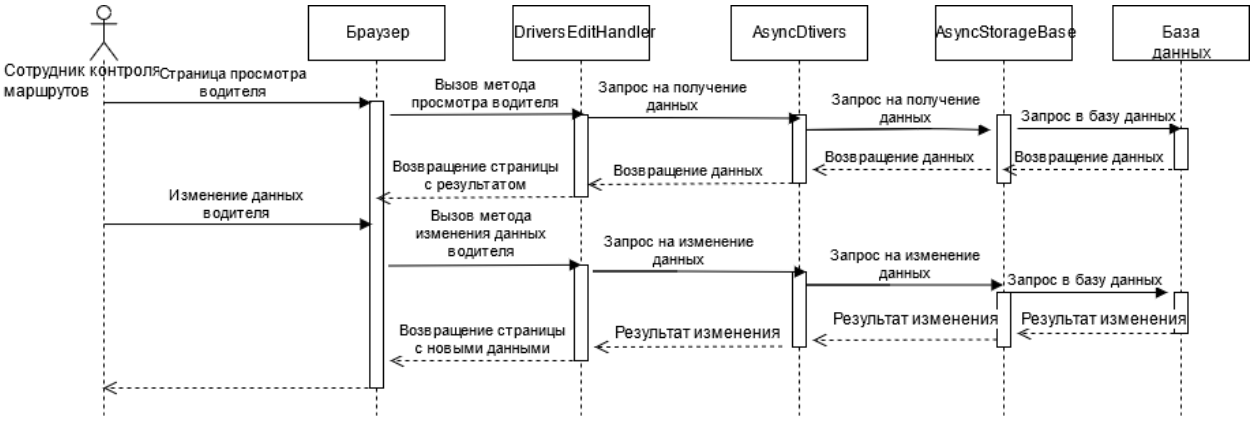

Рисунок 31 – Диаграмма последовательности редактирования информации водителя.

Модуль Storage обеспечивает взаимодействие сервера приложения с базой данных. Класс методы класса StorageBase реализует запросы к базе данных. Класс SyncStorageBase наследуется от класса StorageBase и реализует синхронные запросы к базе данных, переопределяя методы родительского класса. Класс AsyncStorageBase аналогичен классу SyncStorageBase, но он реализует асинхронные запросы к базе данных. Классы DriversMixin, SyncDrivers, AsyncDrivers реализуют обращения к таблице с данными водителей. На рисунке 32 изображена диаграмма классов модуля Storage.

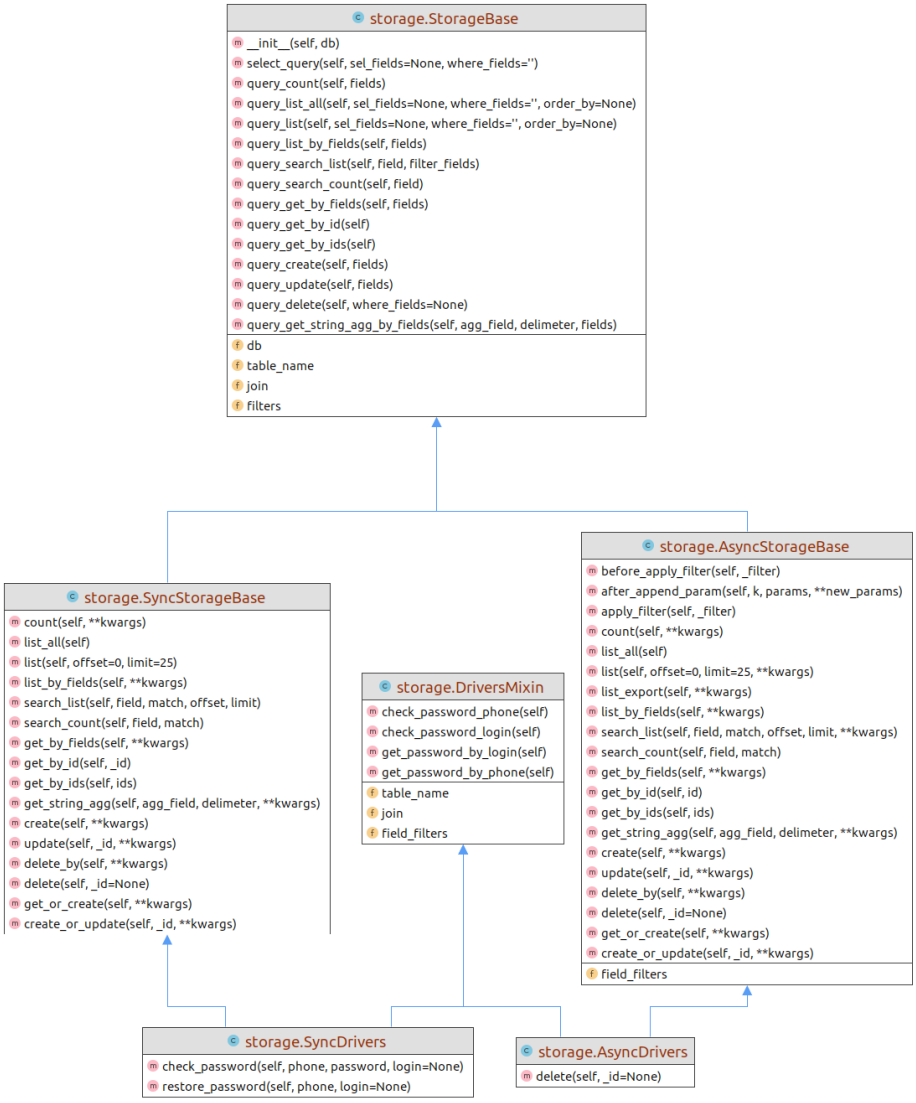

Рисунок 32 – Диаграмма классов модуля Storage

Система содержит множество классов, которые наследуются от AsyncStorageBase или SyncStorageBase. Каждый такой класс аналогичен DriversMixin, SyncDrivers, AsyncDrivers и реализует обращения к соответствующей таблице.

3.3.2 Разработка программного обеспечения тонкого клиента

Во время разработки клиентской части тонкого клиента использовался язык разметки HTML, язык CSS, JavaScript и библиотека jQuery.

Для уменьшения нагрузки на клиента все операции выполняются на стороне сервера. На стороне клиента выполняются минимальные операции используя язык программирования JavaScript. В базовом макете base.html от которого наследуются все остальные макеты, подключаются сразу все css-файлы, которые обеспечивают графическое оформление приложения. Подключаемы js-файлы обеспечивают динамику сайта.

Для удобства использования приложения пользователем используется технология Ajax запросов. Например при выборе определенной остановки приложение отправляет асинхронный запрос на сервер и после его выполнения обновляет карту, нанося на неё выбранную остановку.

Код реализации клиентской и серверной части представлен в приложении А. Использование Ajax запросов и JavaScript позволяет предоставить пользователю бесшовный опыт работы с приложением.

3.3.3 Разработка программного обеспечения мобильного приложения

Программное обеспечение которое запускается на мобильном устройстве, исполняет виртуальная машина Dalvik. Это аналог виртуальной машины Java, но в то же время существенно отличается от неё. Основным отличием является использование архитектуры на основе регистров, структура данных куда помещаются методы основана на регистрах процессора. Благодаря этому операции выполняются быстрее чем у обычных виртуальных машин. Dalvik выполняет байт-код собственного формата. Он преобразует class-файлы в формат .dex, которые оптимизированы для Dalvik VM.

Класс MainActivity – главная активность приложения, наследующаяся от стандартного класса Activity. На ней изображена карта и поисковые поля для выбора нужной остановки.

Классы RoutesActivity, ActiveRoutesActivity, представляют классы

активностей для просмотра маршрутов и активных маршрутов. Эти классы наследуются от ListActivity. Вид данных активностей генерируется программно.

Классы EditPassangerBuilder, EditActiveRoutesBuilder, EditDriversBuilder, являются классами диалоговых окон, предназначенные для редактирования различных записей. Разметка этих классов лежит в xml-файлах.

Класс JournalDb отвечает за обновление и создание базы данных до новых версий.

Класс JournalProvider обеспечивает доступ к таблицам базы данных, позволяя делать запросы, добавлять, удалять или обновлять данные.

3.4 Контроль качества программного обеспечения

Тестирование – способ обеспечения качества программного продукта.

Качество программного продукта характеризуется набором свойств, определяющих, насколько продукт «хорош» с точки зрения заинтересованных сторон. Тестирование является одним из наиболее устоявшихся способов обеспечения качества разработки программного обеспечения и входит в набор эффективных средств современной системы обеспечения качества программного продукта [27].

3.5 Функциональное тестирование программного обеспечения

Функциональное тестирование необходимо, чтобы сделать вывод о корректности работы системы. Функциональные тесты как правило делаются для основных вариантов использования. Функциональное тестирование будет проводиться для двух клиентов. Тонкий клиент представляет собой веб-браузер, а толстый клиент представляет собой мобильное приложение. План функционального тестирования тонкого клиента представлен в таблице 3.

Результаты функционального тестирования в виде снимков экрана приведены в приложении Г.

Таблица 4 – План функционального тестирования программного обеспечения для тонкого клиента

Номер теста	Вариант использования	Описание теста	Ожидаемый результат	Полученный результат
1	Добавление информации о водителе	Открыть страницу добавления информации о водителе, ввести данные, нажать «Сохранить»	Отображение добавленного водителя в списке водителей	Соответствует ожидаемому
2	Просмотр списка маршрутов	Открыть страницу просмотра маршрутов	Корректное отображение списка маршрутов	Соответствует ожидаемому
3	Добавление остановки	Открыть страницу добавления информации о остановки, ввести данные, нажать «Сохранить»	Отображение добавленной с остановки в списке остановок	Соответствует ожидаемому
4	Удаление элемента остановки	Нажать на кнопку удалить на странице списка остановок	Отображение списка остановок без удаленного элемента	Соответствует ожидаемому
5	Добавление активного маршрута	Открыть страницу добавления активного маршрута, ввести данные, нажать «Сохранить»	Новый активный маршрут добавлен	Соответствует ожидаемому
6	Просмотр отзывов	Открыть страницу содержащую информацию об отзыве	Отзыв корректно отображается	Соответствует ожидаемому
7	Добавление нового транспортного средства	Открыть страницу добавления транспортного средства, ввести данные, нажать «Сохранить»	Добавлен новое транспортное средство	Соответствует ожидаемому

Продолжение таблицы 4

Номер теста	Вариант использования	Описание теста	Ожидаемый результат	Полученный результат
8	Изменение статуса	Открыть страницу	Статус обновлен, маршрут	Соответствует ожидаемому

	активного маршрута	редактирования активного маршрута, изменить статус, нажать «Сохранить»	недоступен для использования.	
9	Авторизация	Открыть страницу авторизации, ввести учетные данные, нажать «Sign In»	Отображение стартовой главной страницы информационной системы	Соответствует ожидаемому

Тестирование мобильного клиента системы проводилось с помощью Android Virtual Device, а также на реальных мобильных устройствах под управлением ОС Android 4.1 и выше. План тестирования представлен в таблице 4.

Таблица 5 – План функционального тестирования мобильного приложения

Номер теста	Вариант использования	Описание теста	Ожидаемый результат	Полученный результат
10	Запуск приложения	Запуск приложения на мобильном устройстве	На экране появится главная активность приложения с формой для авторизации	Соответствует ожидаемому
11	Изменение статуса у маршрута	Сделать маршрут активным на активности водителя.	Статус активного маршрута обновился	Соответствует ожидаемому
12	Просмотр списка доступных маршрутов	Открыть список доступных маршрутов	Появится активность со списком доступных маршрутов	Соответствует ожидаемому
17	Авторизация	Открыть активность авторизации, ввести учетные данные, нажать «Войти»	Отображение главной активности приложения	Соответствует ожидаемому

Выводы

По результатам исследований, выполненных в настоящей главе, можно сделать выводы, перечисленные ниже.

Разработана логическая и физическая база данных для сервера информационной системы и мобильного приложения.

Описан пользовательский интерфейс информационной системы.

Проведено функицональное тестирование, доказывающее

работоспособность системы.

Заключение

В диссертационной работе разработана имитационная модель информационной системы контроля общественного транспорта и пассажиропотока. Разработан программный комплекс из клиентского приложения и серверной части.

Была спроектирована физическая и логическая реляционная база данных. Был спроектирован графический интерфейс.

Серверная часть информационной системы была реализована с помощью сервера приложений nginx, фреймворка tornado на языке python. В качестве базы данных использовалась СУБД PostgreSQL. Клиентские приложения реализованы с помощью языков HTML, JavaScript, CSS, Java и СУБД SQLite.

Было произведено сравнение основных характеристик системы при работе с разными нагрузками. Была разработана информационная система с клиентами разной функциональности. Данная система позволяет отслеживать движение общественного транспорта. Добавлять новые маршруты движения, отслеживать загруженность пассажиропотока.

Список использованных источников

1. Ионкин, И.А. Система оптимизации движения общественного транспорта и пассажиропотока / И.А. Ионкин // Информационные технологии в науке и образовании. Проблемы и перспективы: сборник научных статей VII Всероссийской межвузовской научно-практической конференции / под редакцией Л.Р. Фионовой 2020. – Пенза: Изд-во ПГУ, 2020. – С. 401-402.

2. В. Н. Юрьев, В.Н. Волкова Информационные системы в экономике : учебник для студентов, обучающихся по специальностям "Бухгалтерский учет, анализ и аудит", "Финансы кредит" / В. Н. Юрьев, В.Н. Волкова – "Мировая экономика" Санкт-Петербург 2006

3. Крег Л. Применение UML и шаблонов проектирования / Л. Крег – М.: Издательский дом «Вильямс», 2004. – 624 с.

4. Бабич, А.В. Введение в UML : курс лекций / Бабич А.В. — Москва : Интуит НОУ, 2016. — 208 с. — ISBN 978-5-94774-878-9

5. Nginx // Nginx [Электронный ресурс]. URL: https://nginx.org/ru/ (дата обращения 10.12.2020)

6. The Tornado Authors Revision // Introduction [Электронный ресурс]. URL: https://www.tornadoweb.org/en/stable/guide/intro.html (дата обращения 18.10.2020)

7. СУБД PostgreSQL // [Электронный ресурс]. URL: https://www.postgresql.org/about/ (дата обращения 18.10.2020)

8. Aqua-Data-Studio // [Электронный ресурс]. URL: https://axoft.ru/vendors/AquaFold/Aqua-Data-Studio (дата обращения 18.10.2020)

9. Маккоу А. Веб-приложения на JavaScript / А. Маккоу. – СПб.: Питер, 2012. – 288 с.

10. Прохоренок Н. jQuery. Новый стиль программирования на JavaScript. / Прохоренок Н. — М.: Вильямс. – 2010

11. Шафер С. HTML, XHTML и CSS. Библия пользователя, 5-е издание / С. Шафер. – М.: Диалектика, 2010. – 656 с

12. Лабберс П. HTML5 для профессионалов: мощные инструменты для разработки современных веб-приложений / П. Лабберс, Б. Олберс, Ф. Салим. – М.: Вильямс, 2011. – 272 с

13. Афонин, А. Ю. Аналитическое и имитационное моделирование Web-сервера / А. Ю. Афонин, П. П. Макарычев // Новые информационные технологии и системы : тр. IX Междунар. науч.-техн. конф. – Пенза: Изд-во ПГУ, 2010. – Ч. 2. – С. 94–99.

14. Карпова, Т. С. Базы данных: модели, разработка, реализация / Т. С. Карпова. – СПб. : Питер, 2001. – 304 с. – ISBN 5-272-00278-4.

15. Android Studio // Android Studio [Электронный ресурс]. URL: https://ru.wikipedia.org/wiki/Android_Studio (дата обращения 01.10.2020)

16. AnyLogic // AnyLogic [Электронный ресурс]. URL: https://www.anylogic.ru/features/ (дата обращения 02.02.2021) //

17. Баканов М. И., Степанов В. Г. Информационные технологии контроля качества функционирования систем обслуживания в торговле / "Компьютерный аудит" – 2000.

18. Стружкин, Н. П. Базы данных: проектирование : учебник для академического бакалавриата / Н. П. Стружкин, В. В. Годин. — Москва : Издательство Юрайт, 2019. — 477 с.

19. Хорстманн, К. С. Java 2/ Библиотека профессионала, том 1. Основы / Кей С. Хорстманн, Гари Корнелл. – 7-е изд., пер. с англ. – М. : Издательский дом ≪Вильямс≫, 2007. – 896 с. – ISBN 978-5-8459-0970-1

20. Туманов В.Е. Основы проектирования реляционных баз данных / В.Е. Туманов. – М.: Бином, 2011. – 420 с

21. Макнейл П. Веб-дизайн. Идеи, секреты, советы. / Макнейл П. – СПб.: Питер. – 2012

22. Зандстра М. Объекты, шаблоны и методики программирования, 2-е издание / М. Зандстра– М.: Издательский дом «Вильямс», 2010. – 463 с.

23. Крег Л. Применение UML и шаблонов проектирования / Л. Крег – М.: Издательский дом «Вильямс», 2004. – 624 с.

24. Шафер С. HTML, XHTML и CSS. Библия пользователя, 5-е издание / С. Шафер. – М.: Диалектика, 2010. – 656 с

25. Афонин, А. Ю. Система анкетирования на основе Web-технологий /А. Ю. Афонин, П. П. Макарычев // Известия высших учебных заведений.Поволжский регион. Технические науки. – 2010. – № 3. – С. 33–39

26. Берд Дж. Веб-дизайн. Руководство разработчика. / Берд Дж. –

СПб.: Питер. – 2012

27. Котляров В.П. Основы тестирования программного обеспечения / В.П. Котляров, Т.В. Коликова. – М.: Бином. Лаборатория знаний, 2006. – 285 с.

ГЛОССАРИЙ

Приложение А

(справочное)

XML (eXtensible Markup Language) – расширяемый язык разметки. Используется для хранения и передачи данных.

API (программный интерфейс приложения, интерфейс прикладного программирования) – описание способов (набор классов, процедур, функций, структур или констант), которыми одна компьютерная программа может взаимодействовать с другой программой.

HTTP (HyperText Transfer Protocol) – протокол прикладного уровня передачи данных, изначально – в виде гипертекстовых документов в формате HTML, в настоящее время используется для передачи произвольных данных.

MVCC (multiversion concurrency control –управление параллельным доступом посредством многоверсионности) –один из механизмов СУБД для обеспечения параллельного доступа к базам данных, заключающийся в предоставлении каждому пользователю так называемого «снимка» базы, обладающего тем свойством, что вносимые пользователем изменения невидимы другим пользователям до момента фиксации транзакции. Этот способ управления позволяет добиться того, что пишущие транзакции не блокируют читающих, и читающие транзакции не блокируют пишущих.

JSON –текстовый формат обмена данными, основанный на JavaScript. Как и многие другие текстовые форматы, JSON легко читается людьми.

ER-модель (Entity-Relationship model, модель «сущность –связь») – модель данных, позволяющая описывать концептуальные схемы предметной области.

AJAX –подход к построению интерактивных пользовательских интерфейсов веб-приложений, заключающийся в «фоновом» обмене данными браузера с веб-сервером.

DOM (document bbject model) – это независящий от платформы и языка программный интерфейс, позволяющий программам и скриптам получить доступ к содержимому HTML, XHTML и XML-документов, а также изменять содержимое, структуру и оформление таких документов.

HTML (Hyper Text Markup Language) – «язык гипертекстовой разметки», предназначенный для создания и просмотра Web-страниц в Сети

CSS (cascading style sheets) – формальный язык описания внешнего вида документа, написанного с использованием языка разметки.

JVM - Java Virtual Machine виртуальная машина Java — основная часть исполняющей системы Java, так называемой Java Runtime Environment (JRE).

ACID - описывает требования к транзакционной системе (например, к СУБД), обеспечивающие наиболее надёжную и предсказуемую её работу.

атомарность. Консистентность, изолированность, прочность.

Цепь Маркова – последовательность случайных событий с конечным или счётным числом исходов, где вероятность наступления каждого события зависит только от состояния, достигнутого в предыдущем событии.

РМД – Реляционная модель данных.

Приложение Б

ЛИСТИНГ СЕРВЕРНОГО ПРИЛОЖЕНИЯ

(рекомендуемое)

Листинг Б.1 – Исходный код модуля Storage

```
class StorageBase(object):
   table_name = ''
   join = {}
   filters = []

   def __init__(self, db):
       self.db = db

   def select_query(self, sel_fields=None, where_fields=''):
       select_fields = []
       tables = [self.table_name]

       for k, v in self.join.iteritems():
           tables.append('%s join %s on (%s = %s)' % (v.get('direction', ''),
k, v['on'][0], v['on'][1]))
           tmp = []
           if isinstance(v['select_field'], tuple):
               tmp.append(v['select_field'])
           else:
               tmp = v['select_field']
           for sf in tmp:
               select_fields.append('%s.%s as %s' % (k, sf[0], sf[1]))

       sel_fields = sel_fields or ['*']
       if isinstance(sel_fields, list):
           select_fields.append(', '.join(['%s.%s' % (self.table_name, x) for
x in sel_fields]))
       else:
           select_fields = [sel_fields]

       query = ''' select %s from %s ''' % (', '.join(select_fields),
'\r\n'.join(tables))

       where_fields = ' and '.join(filter(None, [where_fields] +
self.filters))
       if where_fields:
           query += ' where %s' % where_fields
       return query

   def query_count(self, fields):

       params = dict(
           sel_fields='count(1) as cnt'
       )
       if fields:
           fields = ' and '.join(map(lambda f: '%s = %%s' % f, fields))
           params.update(dict(where_fields=fields))

       return self.select_query(**params)

   def query_list_all(self, sel_fields=None, where_fields='', order_by=None):
       order_by_clause = order_by and order_by or '%s.id' % self.table_name
       if order_by_clause:
           order_by_clause = ' order by %s' % order_by_clause
       return self.select_query(sel_fields, where_fields) + order_by_clause

   def query_list(self, sel_fields=None, where_fields='', order_by=None):
       return self.query_list_all(sel_fields, where_fields, order_by) + '''
offset %s limit %s'''

   def query_list_by_fields(self, fields):
       fields = ' and '.join(map(lambda f: '%s.%s = %%s' % (self.table_name,
f), fields))
       return self.select_query(where_fields=fields) + ' order by %s.id' %
self.table_name

   def query_search_list(self, field, filter_fields):
```

```
        fields = ' and '.join(filter(None, [' and '.join(map(lambda f: '%s.%s
= %%s' % (self.table_name, f), filter_fields)),
                 '%s.%s ilike %%s' % (self.table_name, field)]))
        query = self.select_query(where_fields=fields)
        return query + ''' offset %s limit %s'''

    def query_search_count(self, field):
        return self.select_query(sel_fields='count(*) as cnt',
where_fields='%s.%s ilike %%s' % (self.table_name, field))

    def query_get_by_fields(self, fields):
        fields = ' and '.join(map(lambda f: '%s.%s = %%s' % (self.table_name,
f), fields))
        return self.select_query(where_fields=fields)

    def query_get_by_id(self):
        return self.select_query(where_fields='%s.id = %%s' % self.table_name)

    def query_get_by_ids(self):
        return self.select_query(where_fields='%s.id = any(%%s)' %
self.table_name)

    def query_create(self, fields):
        values_rep = ', '.join(['%s'] * len(fields))
        fields = ', '.join(map(lambda f: '%s' % f, fields))
        return '''insert into ''' + self.table_name + '''(''' + fields + ''')
values (''' + values_rep + ''')'''

    def query_update(self, fields):
        fields = ', '.join(map(lambda f: '%s = %%s' % f, fields))
        return '''update ''' + self.table_name + ''' set ''' + fields + '''
where id = %s'''

    def query_delete(self, where_fields=None):
        query = ''' delete from ''' + self.table_name
        if where_fields:
            query += ''' where ''' + ' and '.join(map(lambda f: '%s = %%s' %
f, where_fields))
        return query

    def query_get_string_agg_by_fields(self, agg_field, delimeter, fields):
        fields = ' and '.join(map(lambda f: '%s.%s = any(%%s)' %
(self.table_name, f), fields))
        return self.select_query(sel_fields='''string_agg(%s::varchar(240),
'%s') as agg_name''' % (agg_field, delimeter), where_fields=fields)
```

Листинг Б.2 – Исходный код хендлеров

```
class RoutesBaseHandler(EditorBaseHandler):
    TITLE = u'Маршруты'
    ICON = 'fa-picture-o'
    PERMISSION_REQUIRED = 'routes'
    FILTER_COOKIE_NAME = 'routes_filter'
    PAGE_COOKIE_NAME = 'routes_page'


    MENU_GROUPS = OrderedDict([
        ('ROUTES', {
            'title': u'Маршруты',
            'icon': 'fa-save'
        })
    ])

    FIELD_NAMES = _O(
        route_num=u'Имя',
        is_active=u'Статус',
    )
    # Добавить statuses
```

```
    def __init__(self, application, request, **kwargs):
        super(RoutesBaseHandler, self).__init__(application, request,
                                                **kwargs)
        self.PAGE_COOKIE_NAME = 'routes_page'
        self.FILTER_COOKIE_NAME = 'routes_filter'

    @staticmethod
    def is_changed(old_data, new_data, unique=u''):
        unique_changed = False
        changed = False
        if not old_data and new_data:
            return True, True

        for field, value in old_data.iteritems():
            if new_data.get(field, value) != value:
                unique_changed = True if field == unique else False
                changed = True
        return changed, unique_changed

    def render(self, template_name, **kwargs):
        super(RoutesBaseHandler, self).render(
            template_name,
            field_names=self.FIELD_NAMES,
            format_date=format_date,
            **kwargs)

    @authenticated
    def get_filter(self):
        _filter = _O()
        for k in AsyncRoutes.field_filters.keys():
            val = self.get_argument(k, default=None)
            if val:
                eq =
formencode.validators.Bool().to_python(self.get_argument('%s_equal' % k,
False))
                _filter[k] = (val, eq)
        return _filter

@route('/routes/(\d+)', name='page-routes-edit')
class RoutesEditHandler(RoutesBaseHandler):
    HIDDEN = True

    @gen.coroutine
    @authenticated
    def page_render(self, data):
        statuses = [_O(name=u'Активен', id=1), _O(name=u'Не активен', id=0)]
        self.render(
            'pages/routes/route/edit.html',
            data=data,
            statuses=statuses
        )

    @gen.coroutine
    @authenticated
    def get(self, _id):
        _id = long(_id)
        if _id == 0:
            data = _O(
                route_num=u'', second_name=u'', busstop_list=[], status_id=0,
                coordinates=u'', city_id=None
            )
        else:
            data = yield AsyncRoutes(self.master_db).get_by_id(_id)
            data.coordinates = u';'.join(data.coordinates_list)
            data.status_id = int(data.is_active)
            if not data:
                raise HTTPError(404)
        data['id'] = _id
        data.busstop_all = yield AsyncBusstop(self.master_db).get_all_stops()
```

```
        # test = yield AsyncDrivers(self.master_db).get_all_stops()
        if not isinstance(data.busstop_list, list):
            data.busstop_list = []
        yield self.page_render(data)


    @gen.coroutine
    @authenticated
    def post(self, _id):
        _id = long(_id)
        data = _O(errors=[])

        for k in 'status_id route_num city_id coordinates
route_busstops'.split():
            val = self.get_argument(k, default=None) or None
            if k in 'status_id route_num city_id coordinates
route_busstops'.split():
                if not val:
                    data.errors.append(u"The {} must be
filled".format(k.capitalize()))

            if k in ['status_id']:
                data['is_active'] = bool(val)
            else:
                if k == 'route_num':
                    data[k] = self.request.arguments[k][0]
                if k == 'city_id':
                    data[k] = int(self.request.arguments[k][0])
                if k == 'route_busstops':
                    data['busstop_list'] = map(int,
self.request.arguments['route_busstops'])
                if k == 'coordinates':
                    data['coordinates_list'] = val.split(';')
        if not data['errors']:
            del data['errors']
            record, updated = yield
AsyncRoutes(self.master_db).create_or_update(_id, **data)
        else:
            dt = yield AsyncRoutes(self.master_db).get_by_id(_id)
            data['id'] = _id
            yield self.page_render(data)
            return
        self.redirect(''.join([self.reverse_url('page-routes-home'), '?page=',
str(self.get_cookie(self.PAGE_COOKIE_NAME, 1))]))

lass EditorJSONHandler(EditorBaseHandler):
    def set_default_headers(self):
        self.set_header('Content-Type', 'application/json')
        super(EditorJSONHandler, self).set_default_headers()

    def write_error(self, status_code, **kwargs):
        error = dict(status_code=status_code)

        if 'exc_info' in kwargs:
            exception = kwargs['exc_info'][1]
            if isinstance(exception, HTTPError) and exception.reason:
                error['reason'] = exception.reason
        self.write(error)

    def write(self, chunk):


        try:
            pretty_print = bool(int(self.get_argument('_pretty')))
        except (MissingArgumentError, ValueError):
            pretty_print = False

        super(EditorJSONHandler, self).write(better_json_encode(chunk,
pretty_print))
```

```
@route('/~/city/(multisearch|search)', name='page-city-search')
class CitySearchHandler(EditorJSONHandler):
   HIDDEN = True

   @gen.coroutine
   @authenticated
   def get(self, _type):
       match = self.get_argument('match', '')
       page = self.get_argument('page', '')
       offset = (int(page) - 1) * SEARCH_LIMITER

       logging.info('Start data count')
       test_time = datetime.datetime.now()
       data = yield AsyncCity(self.master_db).search_list('name', match,
offset, SEARCH_LIMITER)
       logging.info('Time %s' % str(datetime.datetime.now() - test_time))
       total_cnt = yield AsyncCity(self.master_db).search_count('name',
match)
       data = dict(
           data=data,
           total_count=total_cnt,
           limit=SEARCH_LIMITER
       )
       self.write(data)
```

Листинг Б.3 – Исходный код хендлеров

Приложение В
ЛИСТИНГ КЛИЕНТСКОГО ПРИЛОЖЕНИЯ
(рекомендуемое)

Листинг В.1 – Исходный код файла base.html

```
<!DOCTYPE html>
```

```
<!--[if IE 8]>         <html class="ie8"> <![endif]-->
<!--[if IE 9]>         <html class="ie9 gt-ie8"> <![endif]-->
<!--[if gt IE 9]><!--> <html class="gt-ie8 gt-ie9 not-ie"> <!--<![endif]-->
<head>
   <meta charset="utf-8">
   <meta http-equiv="X-UA-Compatible" content="IE=edge,chrome=1">
   <meta name="viewport" content="width=device-width, initial-scale=1.0,
user-scalable=no, minimum-scale=1.0, maximum-scale=1.0">
   <meta name="author" content="">

   <title>{% block page_title %}Editor{% end %}</title>

   <!-- Open Sans font from Google CDN -->
   <link
href="http://fonts.googleapis.com/css?family=Open+Sans:300italic,400italic,60
0italic,700italic,400,600,700,300&subset=latin" rel="stylesheet"
type="text/css">

   <link href="{{ static_url('assets/stylesheets/bootstrap.min.css') }}"
rel="stylesheet" type="text/css">
  <link href="{{ static_url('assets/stylesheets/pixel-admin.min.css') }}"
rel="stylesheet" type="text/css">
  <link href="{{ static_url('assets/stylesheets/pages.min.css') }}"
rel="stylesheet" type="text/css">
  <link href="{{ static_url('assets/stylesheets/rtl.min.css') }}"
rel="stylesheet" type="text/css">
   <link href="{{ static_url('assets/stylesheets/themes.min.css') }}"
rel="stylesheet" type="text/css">
   <link href="{{ static_url('assets/stylesheets/p-loading.min.css') }}"
rel="stylesheet" type="text/css">
   <link href="{{ static_url('css/libs/colorbox.css') }}" rel="stylesheet"
type="text/css">
   <link href="{{ static_url('js/plugins/jquery-upload-file-
4.0.2/css/uploadfile.css') }}" rel="stylesheet" type="text/css">
   <link href="{{ static_url('js/plugins/jQuery-File-Upload-
9.31.0/css/jquery.fileupload.css') }}" rel="stylesheet" type="text/css">
   <link href="{{ static_url('css/masterdb.css') }}" rel="stylesheet"
type="text/css">

   <!--[if lt IE 9]>
       <script src="assets/javascripts/ie.min.js"></script>
   <![endif]-->
 </head>

 <body class="theme-default {% block page_style %}{% end %}">
   <script>var init = [];</script>

   {% block body %}
   <div id="main-wrapper">
       <!-- Navigation bar -->
       {% include "navbar.html" %}
       {% include "sidebar.html" %}

       <div id="content-wrapper">

           <div class="page-header">
               <h1>{% block page_header %}{% end %}</h1>
           </div> <!-- / .page-header -->

           {% block content %}{% end %}

       </div> <!-- / #content-wrapper -->

       <div id="main-menu-bg"></div>
   </div> <!-- / #main-wrapper -->
   {% end %}

   {% block footer_js %}

   {% end %}
```

```
   {% block extra_footer_js %}
   {% end %}
 </body>
</html>
```

Листинг В.2 – Исходный код файла routes.edit.html

```
{% extends "../../base.html" %}
{% block page_header %}
<link rel="stylesheet"
href="https://unpkg.com/leaflet@1.7.1/dist/leaflet.css"
integrity="sha512-
xodZBNTC5n17Xt2atTPuE1HxjVMSvLVW9ocqUKLsCC5CXdbqCmblAshOMAS6/keqq/sMZMZ19scR4
PsZChSR7A=="
crossorigin=""/>
<script src="{{ static_url('js/libs/leaflet/leaflet.js') }}"></script>
<script src="https://unpkg.com/leaflet-routing-machine@latest/dist/leaflet-
routing-machine.js"></script>
<link rel="stylesheet" href="leaflet-routing-machine-3.2.7/dist/leaflet-
routing-machine.css" />
<script src="leaflet-routing-machine-3.2.7/dist/leaflet-routing-
machine.js"></script>
   <span class="text-light-gray">
       <i class="fa {{icon}} page-header-icon"></i> {{ _(title) }} -
Редактирование </span>
       {% if data.route_num %} / «{{ _(data.route_num) }}»{% end %}
{% end %}

{% block content %}
   <div class="busstops">
       <form id="busstop_add" class="panel form-horizontal" action=""
method="post">
           {% raw xsrf_form_html() %}
           <div class="panel-body">
               {% if 'errors' in data %}
                   {% for error in data.errors %}
                       <div class="alert alert-warning" role="alert">{{ error
}}</div>
                   {% end %}
               {% end %}
               <!-- Tabs -->
               <ul class="nav nav-tabs bs-tabdrop-example">
                   <li class="active"><a href="#bs-tabdrop-tab1" data-
toggle="tab">Основные данные</a></li>
               </ul>
               <div class="tab-content tab-content-bordered clearfix">

                   <div class="tab-pane active" id="bs-tabdrop-tab1">

                       <div class="form-group">
                           <label for="route_num" class="col-sm-2 control-
label">Название маршрута</label>
                           <div class="col-sm-2">
                               <input id="route_num" name="route_num"
type="text" class="form-control"
                                      {% if data.route_num %} value="{{
_(data.route_num) }}" {% end %}>
                           </div>
                           <label for="coordinates" class="col-sm-2 control-
label">Список координат</label>
                           <div class="col-sm-2">
                               <input id="coordinates" name="coordinates"
type="text" class="form-control" readonly
                                      {% if data.coordinates %} value="{{
_(data.coordinates) }}" {% end %}>
                           </div>
                           <label for="city_id" class="col-sm-2 control-
label">Название города</label>
                           <div class="col-sm-2">
```

```
                                <input type="text" id="city_id" name="city_id"
class="form-control jquery-select2"
                                       data-href="/~/city/search" {% if
data['city_id'] %} value="{{ data['city_id'] }}" {% end %}>
                            </div>
                            <input id="waypoints_list" name="waypoints_list"
type="text" class="form-control" style="display: none">
                            <input id="route_busstops_list"
name="route_busstops_list" type="text" class="form-control" style="display:
none">
                        </div>
                        <div class="form-group">
                            <label for="status_id" class="col-sm-2 control-
label">Статус</label>
                            <div class="col-sm-2">
                                <select class="form-control" id="status_id"
name="status_id" >
                                    {% for val in statuses %}
                                        {% if data.status_id == val.id %}
                                            <option selected value="{{
_(val.id) }}">{{ _(val.name) }}</option>
                                        {% else %}
                                            <option value="{{ _(val.id) }}">{{
_(val.name) }}</option>
                                        {% end %}
                                    {% end %}
                                </select>
                            </div>
                            <label for="route_busstops" class="col-sm-2
control-label">Список остановок</label>

                            <div class="col-sm-2 select2-success">
                                <select id="route_busstops"
name="route_busstops" class="route_busstops" multiple="multiple">
                                    {% for busstop in data.busstop_all %}
                                        <option {% if  busstop.id in
data.busstop_list %}selected{% end %} value="{{_(busstop.id)}}" >{{
_(busstop.adress) }}</option>
                                    {% end %}
                                </select>
                            </div>
                            <div class="form-group">
                            <div class="col-sm-2">
                                <button id="create-route" type="button"
name="create-route" class="btn create-route">{{ _("Создать маршрут")
}}</button>
                            </div>
                            <div class="col-sm-2">
                                <button id="clean-route" type="button"
name="clean-route" class="btn clean-route">{{ _("Очистить маршрут")
}}</button>
                            </div>
                        </div>
                        <div class="form-group">
                            <div id="mapid" class="map_show" style="height:
540px;"></div>
                            <!-- Здесь будет карта-->
                        </div>

                        </div>

                </div>
                </div><!-- /.nav-tab -->

            </div><!-- /.panel-body -->

            <div class="panel-footer text-right">
                <div class="form-group">
                    <div class="col-sm-offset-11 col-sm-1">
```

```
                        <button id="save-button-submit" name="save"
type="submit" class="btn btn-primary">{{ _("Сохранить") }}</button>
                    </div>
                </div>
            </div>

        </form>

        <div class="modals_block"></div>

    </div><!-- / products -->

{% end %}
{% block extra_footer_js %}
<script src="{{ static_url('app/route.js') }}"></script>
{% end %}
```

Листинг В.3 – Исходный код файла busstop.edit.html

```
{% extends "../base.html" %}
{% block page_header %}
<link rel="stylesheet"
href="https://unpkg.com/leaflet@1.7.1/dist/leaflet.css"
integrity="sha512-
xodZBNTC5n17Xt2atTPuE1HxjVMSvLVW9ocqUKLsCC5CXdbqCmblAshOMAS6/keqq/sMZMZ19scR4
PsZChSR7A=="
crossorigin=""/>
<script src="{{ static_url('js/libs/leaflet/leaflet.js') }}"></script>
    <span class="text-light-gray">
        <i class="fa {{icon}} page-header-icon"></i> {{ _(title) }} -
Редактирование </span>
        {% if data.name %} / «{{ _(data.name) }}»{% end %}
{% end %}

{% block content %}
    <div class="busstops">
        <form id="busstop_add" class="panel form-horizontal" action=""
method="post">
            {% raw xsrf_form_html() %}
            <div class="panel-body">
                {% if 'errors' in data %}
                    {% for error in data.errors %}
                        <div class="alert alert-warning" role="alert">{{ error
}}</div>
                    {% end %}
                {% end %}
                <!-- Tabs -->
                <ul class="nav nav-tabs bs-tabdrop-example">
                    <li class="active"><a href="#bs-tabdrop-tab1" data-
toggle="tab">Основные данные</a></li>
                </ul>
                <div class="tab-content tab-content-bordered clearfix">

                    <div class="tab-pane active" id="bs-tabdrop-tab1">

                        <div class="form-group">
                            <label for="name" class="col-sm-2 control-
label">Название</label>
                            <div class="col-sm-2">
                                <input id="name" name="name" type="text"
class="form-control"
                                        {% if data.name %}value="{{
_(data.name) }}"{% end %}>
                            </div>
                            <label for="adress" class="col-sm-2 control-
label">Адрес</label>
                            <div class="col-sm-2">
                                <input id="adress" name="adress" type="text"
class="form-control"
```

```
                                        {% if data.adress %}value="{{
_(data.adress) }}"{% end %}>
                            </div>
                            <label for="coordinates" class="col-sm-2 control-
label">Координаты</label>
                            <div class="col-sm-2">
                                <input id="coordinates" name="coordinates"
type="text" class="form-control"
                                        {% if data.coordinates %} value="{{
_(data.coordinates) }}" readonly {%else%} readonly{% end %}>
                            </div>
                        </div>
                        <div class="form-group">
                            <div id="mapid" class="map_show" style="height:
540px;"></div>
                            <!-- Здесь будет карта-->
                        </div>

                        </div>
                </div><!-- /.nav-tab -->

            </div><!-- /.panel-body -->

            <div class="panel-footer text-right">
                <div class="form-group">
                    <div class="col-sm-offset-11 col-sm-1">
                        <button id="save-button-submit" name="save"
type="submit" class="btn btn-primary">{{ _("Сохранить") }}</button>
                    </div>
                </div>
            </div>

        </form>

        <div class="modals_block"></div>

    </div><!-- / products -->

{% end %}
{% block extra_footer_js %}
<script src="{{ static_url('app/busstop.js') }}"></script>
{% end %}
```

Листинг В.4 – Исходный код файла route.js

```
$(".route_busstops").select2();
$('.waypoints_list').hide();
$(function() {
    var mymap = L.map('mapid').setView([53.22507, 44.88923], 13);
    var marker_list = [];
    var waypoints_list = [];
    var old_select = [];
    var busstop_list_marker = [];
    var routes_list = [];
    var route_var;
    var route_text = '';
    var markers_list = [];
    L.tileLayer('https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png').addTo(my
map);
    var myRedIcon = L.icon({iconUrl: '/static/img/marker-icon-red.png',
iconAnchor:   [12, 41],});

    mymap.on('click', function(e) {
        onMapClick(e);
    });
    if ($('#coordinates').val()){
        drawRoute($('#coordinates').val());
        drawBusstop($('#route_busstops').val());
    }
```

```
    $(".route_busstops").on('change', function (e){
        // old_select.push(e.added.id);
        if (e.added){
            // debugger;
            old_select.push(e.added.id);
            getBusstop(e.added.id);
            $("#route_busstops_list").val($("#route_busstops_list").val() + '
' + e.added.id);
        }
        else{
            var index = old_select.indexOf(e.removed.id);
            old_select.splice(index, 1);
            for (var i=0; i< busstop_list_marker.length; i++){
                if (busstop_list_marker[i]['id'] == e.removed.id){
                    busstop_list_marker[i]['marker'].closePopup();
                    mymap.removeLayer(busstop_list_marker[i]['marker']);
                    break;
                }
            }
            busstop_list_marker.splice(i, 1);
            // test_popup.closePopup();
            // mymap.removeLayer(test_popup);
        }

    })

    function getBusstop(bus_id){
        var time_now;
        var time_end;
        time_now = Date.now();
        $.ajax({
                url: '/~/busstop/get_coord',
                type: 'post',
                data: {'id': bus_id}
            }).done(function (response) {
                if (response.success){
                    // debugger;
                    time_end = Date.now();
                    console.log('Get busstop time');
                    console.log((time_end - time_now)/1000);
                    var data_bus = response.busstop;
                    var b_lat, b_lng;
                    var latlng_list = data_bus.coordinates.split(',');
                    b_lat = latlng_list[0];
                    b_lng = latlng_list[1];
                    var bus_marker = L.marker([b_lat, b_lng], {icon:
myRedIcon});
                    bus_marker.addTo(mymap).bindPopup()
                    bus_marker.on('click', function(e){
                        bus_marker.setIcon(myRedIcon);
                    });
                    busstop_list_marker.push({'id': bus_id, 'marker':
bus_marker});
                }

            });
    }

    function drawRoute(coord_str){
        var tmp_list = [];
        var crod_list = [];
        tmp_list = coord_str.split(';');
        for (var i=0; i < tmp_list.length; i++){
            crod_list.push(tmp_list[i].split(','))
        }
        waypoints_list = crod_list;
        route_var = createRoute(waypoints_list);
        $('.leaflet-routing-container').remove();
    };
```

```
    function drawBusstop(busstop_list){
        for (var i=0; i < busstop_list.length; i++){
            getBusstop(busstop_list[i])
        }
    }

    function onMapClick(e) {
        // debugger;
        // Проверить координату
        // mymap.closePopup();
        var new_marker = L.marker([e.latlng.lat, e.latlng.lng]);
        new_marker.addTo(mymap).bindPopup()
        marker_list.push(new_marker);
        if (marker_list.length > 1){
            route_text += ';' + String(e.latlng.lat) + ',' +
String(e.latlng.lng);
            if (route_var){
                // route_var.removeRoutingControl();
                // route_var.remove();
            }
            // debugger;
            waypoints_list = convertMarkers(marker_list);
            route_var = createRoute(waypoints_list);
            routes_list.push(route_var);

        }else{
            route_text += String(e.latlng.lat) + ',' + e.latlng.lng;
        };
        $('.leaflet-routing-container').remove();
    };
    $('.create-route').on('click', function(){
        saveRoute();
    })
    $('.clean-route').on('click', function(){
        // debugger;
        cleanRoute();
    })

    function onClickMarker(e){
       var marker_tmp =  L.marker([e.latlng.lat, e.latlng.lng]);
       var ind = marker_list.indexOf(marker_tmp);
       if (ind >= 0){
           marker_list[ind].closePopup();
           mymap.removeLayer(marker_list[ind]);
           marker_list.slice(ind);
       }
    };

    function saveRoute(){
        $('#waypoints_list').val(String(waypoints_list));
        $('#coordinates').val(route_text);
    };

    function cleanRoute(){
        // debugger;
        for (var i=0;i< marker_list.length; i++){
            marker_list[i].closePopup();
            mymap.removeLayer(marker_list[i]);
        }
        route_var.remove();
        route_var = null;
        waypoints_list = [];
        marker_list = [];
        createRoute(waypoints_list);
        route_text = '';
        $('#waypoints_list').val(String(waypoints_list));
        $('#coordinates').val(route_text);
    };

    function createRoute(coordinates_list){
```

```
        // Берем координаты со страницы
        if (route_var){
            route_var.setWaypoints(coordinates_list);
            route = route_var;
        }
        else{
            var route = L.Routing.control({
                waypoints: coordinates_list,
                display: "None"
            }).addTo(mymap);
        }
        return route;

    };

    function fillCoordStr(){
        var coordinates_str = '';
        for (var i=0; i< marker_list.length; i++){
            coordinates_str += String(marker_list[0]._latlng.lat) + ',' +
String(marker_list[0]._latlng.lng) + ';'
        }
        return coordinates_str;
    }

    function convertMarkers(arr){
        var res_array = [];
       for (var i=0; i < arr.length; i++){
          res_array.push([arr[i]._latlng.lat, arr[i]._latlng.lng]);
        }
       return res_array;
    };

});
```

Приложение Г

РЕЗУЛЬТАТЫ ФУНКЦИОНАЛЬНОГО ТЕСТИРОВАНИЯ

(рекомендуемое)

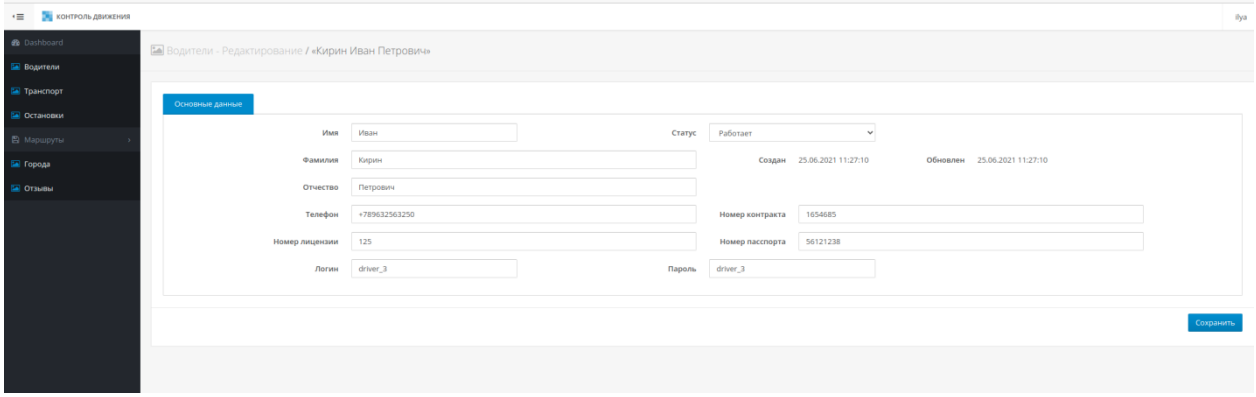

Рисунок Г.1 – Результат выполнения теста №1

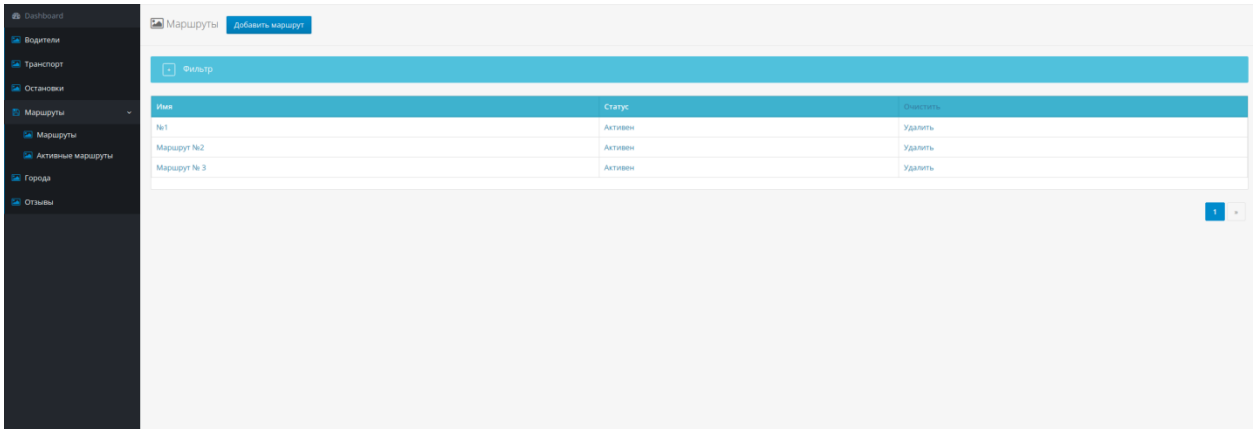

Рисунок Г.2 – Результат выполнения теста №2

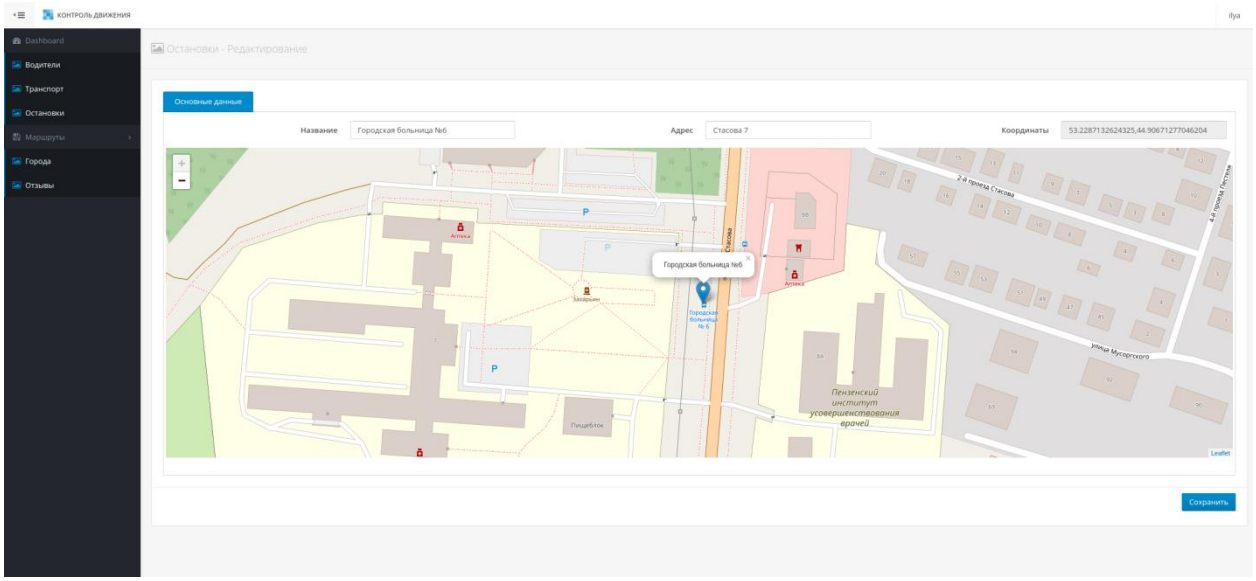

Рисунок Г.3 – Результат выполнения теста №3

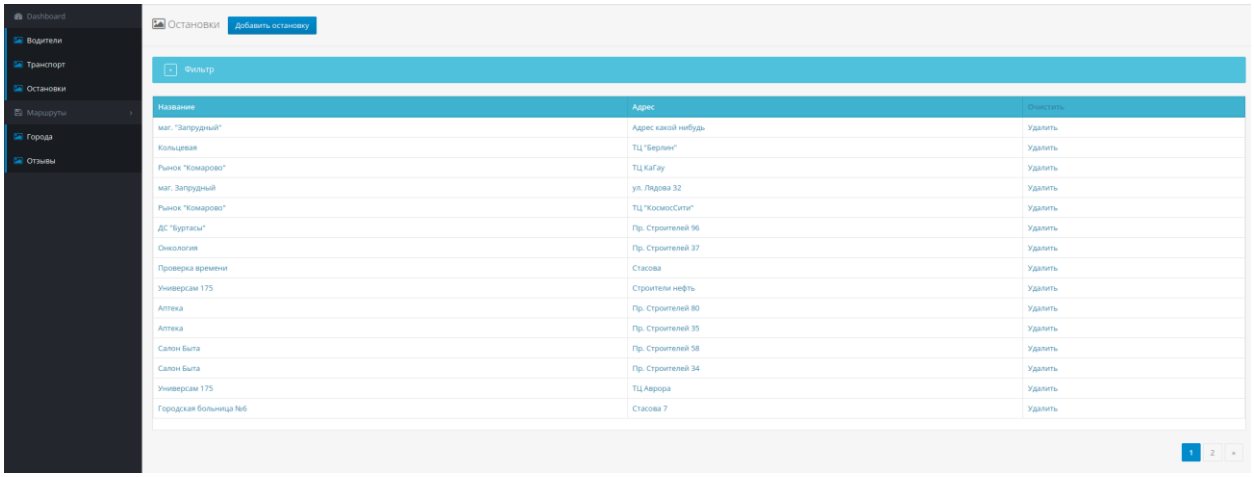

Рисунок Г.4 – Результат выполнения теста №3

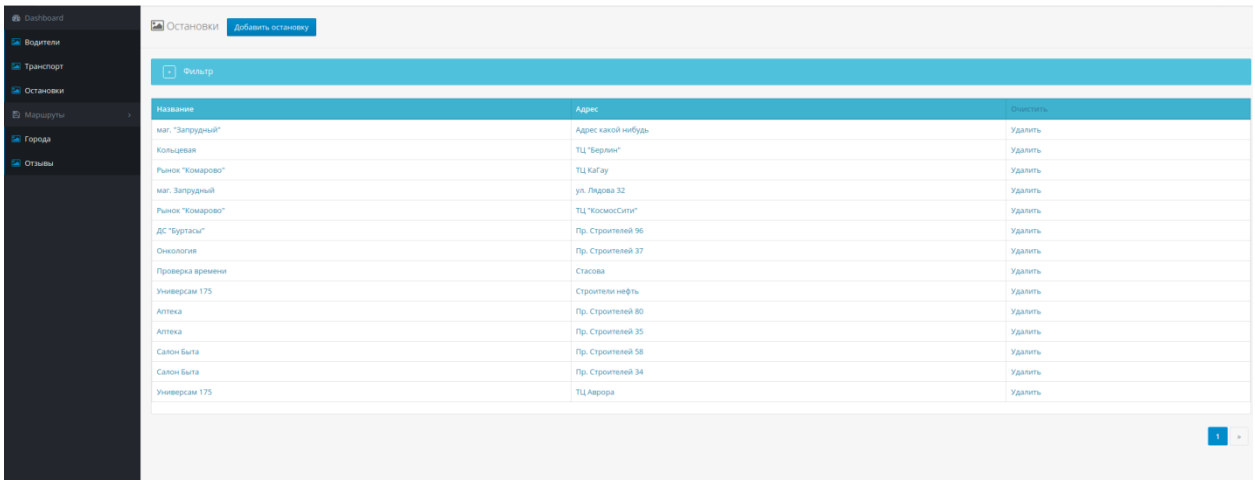

Рисунок Г.5 – Результат выполнения теста №4

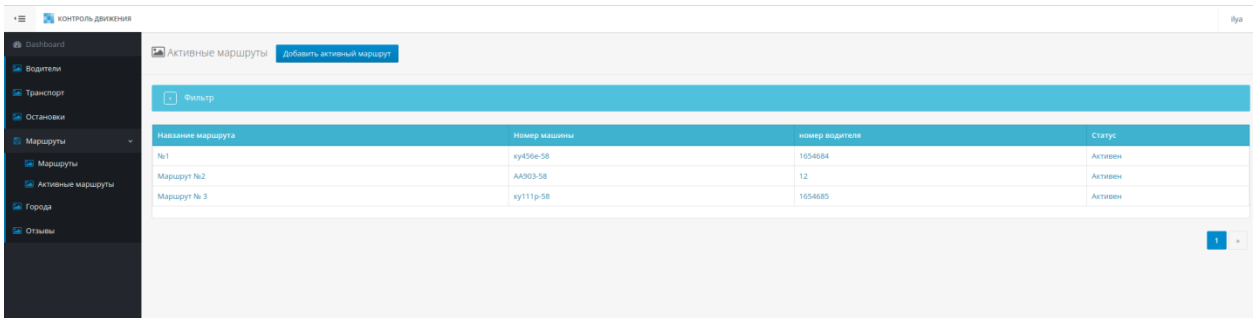

Рисунок Г.6 – Результат выполнения теста №5

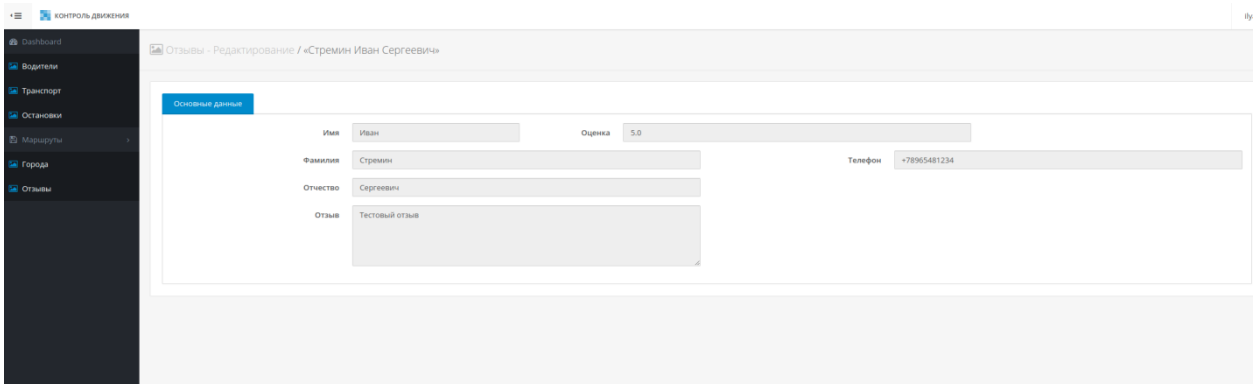

Рисунок Г.7 – Результат выполнения теста №6

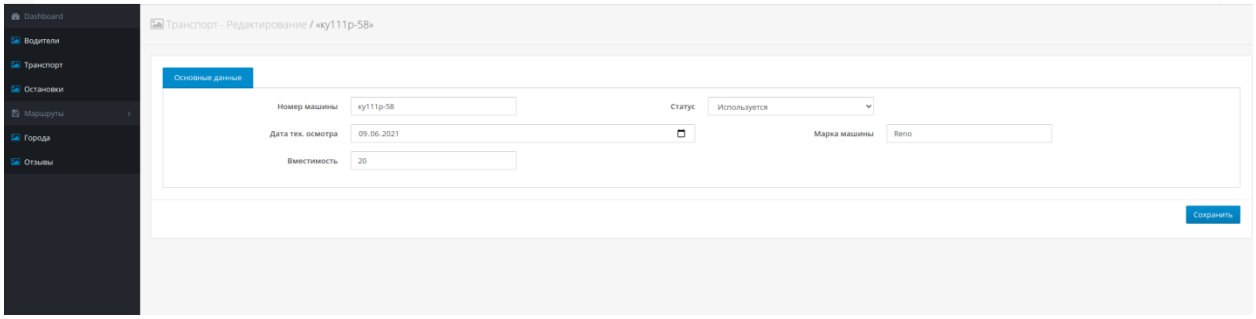

Рисунок Г.8 – Результат выполнения теста №7

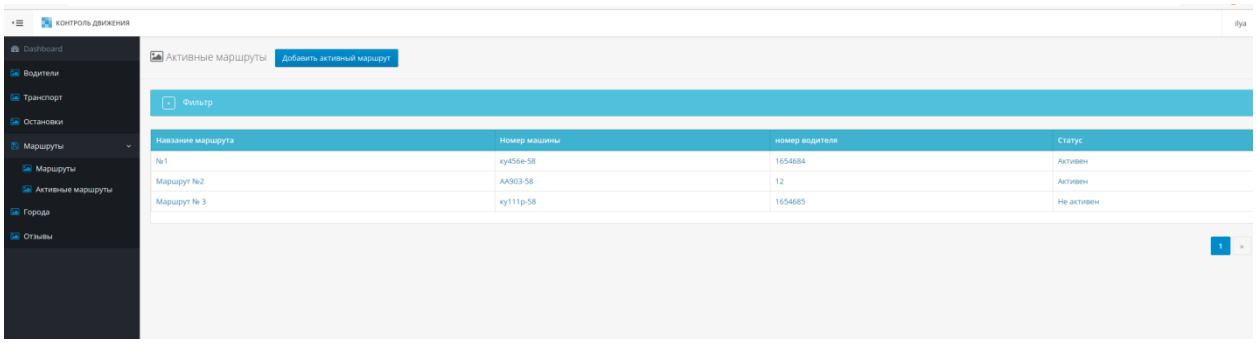

Рисунок Г.9 – Результат выполнения теста №8

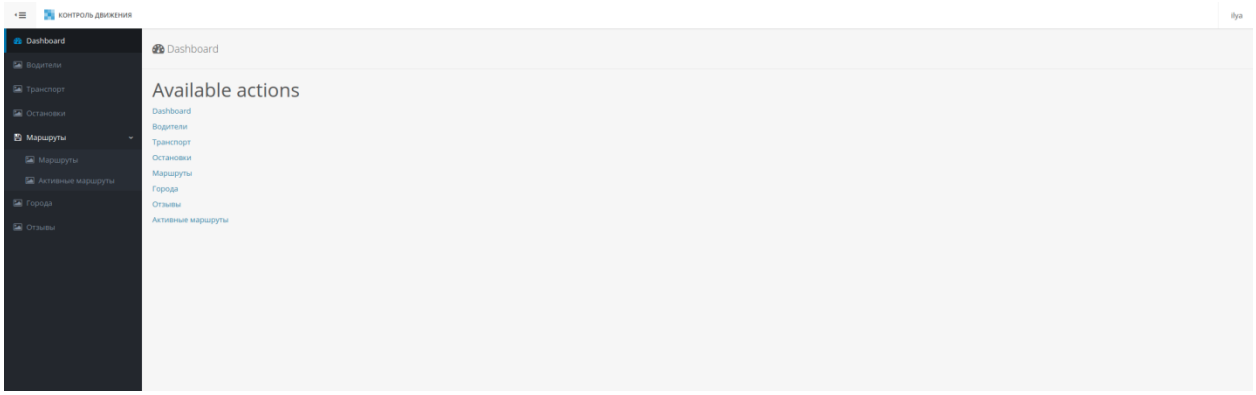

Рисунок Г.10 – Результат выполнения теста №9