

Минобрнауки России
Юго-Западный государственный университет

Кафедра программной инженерии

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА
ПО ПРОГРАММЕ БАКАЛАВРИАТА

09.03.04 Программная инженерия

(код, наименование ОПОП ВО: направление подготовки, направленность (профиль))

«Разработка программно-информационных систем»

**Бизнес-проект «Кроссплатформенная программная система поиска
попутчиков для автомобильных поездок». Разработка серверной части**
(название темы)

Дипломный проект

(вид ВКР: дипломная работа или дипломный проект)

Автор ВКР

М.Е. Шеховцов

(подпись, дата)

(инициалы, фамилия)

Группа ПО-016

Руководитель ВКР

Е. П. Кочура

(подпись, дата)

(инициалы, фамилия)

Нормоконтроль

А. А. Чаплыгин

(подпись, дата)

(инициалы, фамилия)

ВКР допущена к защите:

Заведующий кафедрой

А. В. Малышев

(подпись, дата)

(инициалы, фамилия)

Курск 2024 г.

Минобрнауки России
Юго-Западный государственный университет

Кафедра программной инженерии

УТВЕРЖДАЮ:

Заведующий кафедрой

(подпись, инициалы, фамилия)

«_____» 20____ г.

**ЗАДАНИЕ НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ
РАБОТУ ПО ПРОГРАММЕ БАКАЛАВРИАТА**

Студента Шеховцова М.Е., шифр 19-06-0284, группа ПО-016

1. Тема «Бизнес-проект «Кроссплатформенная программная система поиска попутчиков для автомобильных поездок». Разработка серверной части» утверждена приказом ректора ЮЗГУ от «04» апреля 2024 г. № 1620-с.

2. Срок предоставления работы к защите «11» июня 2024 г.

3. Исходные данные для создания программной системы:

3.1. Перечень решаемых задач:

- 1) провести анализ предметной области;
- 2) разработать концептуальную модель программно-информационной системы;
- 3) спроектировать и реализовать серверную часть программной системы;
- 4) провести тестирование серверной части программной системы.

3.2. Входные данные и требуемые результаты для программы:

1) Входными данными для программной системы являются: пользовательские данные, сведения об автомобилях, фотографии пользовательских учетных записей, данные о поездках, сведения о попутчиках, пользовательские сообщения.

2) Выходными данными для программной системы являются: данные о пользовательских аккаунтах, сформированные поездки, пользовательские чаты, номера телефона попутчиков и водителей, автомобили с привязкой к конкретному пользователю.

4. Содержание работы (по разделам):

4.1. Введение

4.1. Анализ предметной области

4.2. Техническое задание: основание для разработки, назначение разработки, требования к программной системе, требования к оформлению документации.

4.3. Технический проект: общие сведения о программной системе, проект данных программной системы, проектирование архитектуры серверной части программной системы, создания сервиса для автоматизации отслеживания активности поездки, создание контроллеров для отправки и принятия данных по http протоколу.

4.4. Рабочий проект: спецификация компонентов и классов программной системы, тестирование серверной части программной системы, сборка компонентов программной системы.

4.5. Заключение

4.6. Список использованных источников

5. Перечень графического материала:

Лист 1. Сведения о ВКРБ

Лист 2. Цель и задачи разработки

Лист 3. Диаграммы вариантов использования

Лист 4. ER - Диаграмма базы данных

Лист 5. Диаграмма развертывания программы

Лист 6. Диаграммы классов моделей и контроллеров

Лист 7. Диаграмма классов SQL запросов

Лист 8. Заключение

Руководитель ВКР

(подпись, дата)

Е. П. Кочура

(инициалы, фамилия)

Задание принял к исполнению

(подпись, дата)

М.Е. Шеховцов

(инициалы, фамилия)

РЕФЕРАТ

Объем работы равен 126 страницам. Работа содержит 18 иллюстраций, 50 таблиц, 50 библиографических источников и 8 листов графического материала. Количество приложений – 2. Графический материал представлен в приложении А. Фрагменты исходного кода представлены в приложении Б.

Перечень ключевых слов: кроссплатформенная система, поиск попутчиков, автомобильные поездки, серверная часть, информатизация, автоматизация, информационные технологии, автоматический сервис, классы, база данных, компонент, модуль, сущность, метод, попутчик, пользователь, водитель, маршруты, транспорт.

Объектом разработки является кроссплатформенная программная система для поиска попутчиков для автомобильных поездок.

Целью выпускной квалификационной работы является создание удобного и функционального сервиса для поиска попутчиков, что способствует экономии ресурсов, снижению транспортных расходов и уменьшению нагрузки на окружающую среду.

В процессе разработки системы были выделены основные сущности, такие как пользователи, маршруты и поездки, и созданы соответствующие классы и методы модулей, обеспечивающие взаимодействие с данными сущностями и корректную работу системы. Были разработаны и реализованы функциональные компоненты, такие как регистрация пользователей, создание и поиск маршрутов, бронирование мест в поездках, а также мессенджер.

Для реализации серверной части системы использовались технологии C#, ASP.net MVC и Microsoft SQL для создания надежной и производительной базы данных. Обеспечена поддержка различных платформ и устройств благодаря адаптивному дизайну и оптимизации запросов к серверу.

ABSTRACT

The volume of work is 126 pages. The work contains 18 illustrations, 50 tables, 50 bibliographic sources and 8 sheets of graphic material. The number of applications is 2. The graphic material is presented in annex A. The layout of the site, including the connection of components, is presented in annex B. The list of keywords: cross-platform system, travel companion search, car trips, backend, informatization, automation, information technology, automatic service, classes, database, component, module, entity, method, travel companion, user, driver, routes, transport.

The object of the development is a cross-platform software system for finding travel companions for car trips.

The purpose of the final qualification work is to create a convenient and functional service for finding travel companions, which helps to save resources, reduce transport costs and reduce the burden on the environment.

During the development of the system, the main entities such as users, routes and trips were identified, and appropriate classes and methods of modules were created to ensure interaction with these entities and the correct operation of the system. Functional components have been developed and implemented, such as user registration, route creation and search, travel reservations, and messenger.

C# technologies were used to implement the server part of the system, ASP.net MVC and Microsoft SQL to create a reliable and productive database. Support for various platforms and devices is provided thanks to adaptive design and optimization

СОДЕРЖАНИЕ

ВВЕДЕНИЕ	12
РЕЗЮМЕ СТАРТАП-ПРОЕКТА	15
1 Анализ предметной области	18
1.1 Общие принципы и понятия поиска попутчиков для поездки	18
1.2 Приложения для поиска попутчиков, их классификация	20
1.3 Исследование аналогов	22
1.3.1 BlablaCar	22
1.3.2 Едем.рф	23
1.3.3 Попутка.рус	24
1.4 Пользовательское исследование	25
2 Техническое задание	28
2.1 Основание для разработки	28
2.2 Назначение разработки	28
2.3 Требования к программной системе	28
2.3.1 Требования к данным программной системы	28
2.3.2 Функциональные требования к программной системе	29
2.3.2.1 Вариант использования «Регистрация пользователя»	32
2.3.2.2 Вариант использования «Регистрация водителя»	33
2.3.2.3 Вариант использования «Авторизация»	33
2.3.2.4 Вариант использования «Изменение пользовательских данных»	33
2.3.2.5 Вариант использования «Получение информации об автомобилях»	34
2.3.2.6 Вариант использования «Добавление нового автомобиля»	34
2.3.2.7 Вариант использования «Изменение данных об автомобиле»	35
2.3.2.8 Вариант использования «Удаление автомобиля»	35
2.3.2.9 Вариант использования «Получение фото профиля»	36
2.3.2.10 Вариант использования «Смена фото профиля»	36

2.3.2.11 Вариант использования «Получение информации о статусе аккаунта	36
2.3.2.12 Вариант использования «Создание поездки»	37
2.3.2.13 Вариант использования «Удаление поездки»	37
2.3.2.14 Вариант использования «Получение списка поездок по заданным критериям поиска»	38
2.3.2.15 Вариант использования «Бронирование поездки»	38
2.3.2.16 Вариант использования «Отмена брони»	38
2.3.2.17 Вариант использования «Получения списка всех поездок для водителя»	39
2.3.2.18 Вариант использования «Получения списка всех поездок для водителя»	39
2.3.2.19 Вариант использования «Получения списка всех поездок для попутчика»	40
2.3.2.20 Вариант использования «Создание чата»	40
2.3.2.21 Вариант использования «Получение списка чатов»	41
2.3.2.22 Вариант использования «Получение сообщений по чату»	41
2.3.2.23 Вариант использования «Отправка сообщения»	41
2.3.2.24 Вариант использования «Получение отзывов»	42
2.3.2.25 Вариант использования «Удаление отзыва»	42
2.3.3 Нефункциональные требования к программной системе	43
2.3.3.1 Требования к надежности	43
2.3.3.2 Требования к безопасности	43
2.3.3.3 Требования к программному обеспечению	44
2.3.3.4 Требования к аппаратному обеспечению	44
2.4 Требования к оформлению документации	46
3 Технический проект	47
3.1 Общая характеристика организации решения задачи	47
3.2 Обоснование выбора технологий проектирования	48
3.2.1 Язык программирования C#	48
3.2.2 Фреймворк ASP.net	49

3.2.3 Расширение T-SQL	51
3.2.4 Протокол связи HTTP	52
3.3 Проект данных программной системы	54
3.4 Проектирование архитектуры программной системы	58
3.4.1 Компоненты программной системы	58
3.4.2 Архитектура программной системы	61
3.4.3 Архитектура программных классов моделей и контроллеров	63
3.4.4 Архитектура программных классов SQL запросов	65
4 Рабочий проект	68
4.1 Классы, используемые при разработке серверной части программной системы	68
4.1.1 Спецификация классов SQL запросов	68
4.1.2 Описание моделей программной системы	77
4.1.3 Описание контроллеров	81
4.2 Тестирование серверной части программной системы	88
4.2.1 Инструмент тестирования Postman	88
4.2.2 Тестирование URI запросов	89
ЗАКЛЮЧЕНИЕ	98
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ	99
ПРИЛОЖЕНИЕ А Представление графического материала	105
ПРИЛОЖЕНИЕ Б Фрагменты исходного кода программы	114
На отдельных листах (CD-RW в прикрепленном конверте)	126
Сведения о ВКРБ (Графический материал / Сведения о ВКРБ.png)	Лист 1
Цель и задачи разработки (Графический материал / Цель и задачи разработки.png)	Лист 2
Диаграммы вариантов использования (Графический материал / Диаграммы вариантов использования.png)	Лист 3
ER - Диаграмма базы данных (Графический материал / ER - Диаграмма базы данных.png)	Лист 4
Диаграмма развертывания программы (Графический материал / Диаграмма развертывания программы.png)	Лист 5

Диаграммы классов моделей и контроллеров (Графический материал / Диаграммы классов моделей и контроллеров.png)	Лист 6
Диаграмма классов SQL запросов (Графический материал / Диаграмма классов SQL запросов.png)	Лист 7
Заключение (Графический материал / Заключение.png)	Лист 8

ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ

ASP.NET – технология для разработки серверной части от компании Microsoft.

БД – база данных.

JSON – текстовый формат обмена данными.

MVC – шаблон проектирования, разделяющий приложение на три взаимосвязанные компоненты: модель, представление и контроллер.

ПО – программное обеспечение.

Сервер – компьютер или программа, предоставляющие сервисы другим компьютерам или программам в сети.

T-SQL – расширение языка SQL, разработанное компанией Microsoft для работы с базами данных.

ТЗ – техническое задание.

ТП – технический проект.

UML – язык графического описания для объектного моделирования в области разработки программного обеспечения.

URI – унифицированный идентификатор ресурса, строка символов, определяющая идентификатор ресурса в интернете.

ВВЕДЕНИЕ

Сфера приложений для подбора попутчиков набирает популярность и становится важной частью современной транспортной экосистемы. С каждым годом все больше людей предпочитают использовать онлайн-сервисы для поиска попутчиков, что приводит к снижению затрат на поездки и уменьшению нагрузки на дороги. Благодаря развитию информационных технологий, процесс поиска и бронирования поездок стал удобным и доступным для всех пользователей.

Пользователи таких приложений получают возможность быстро найти попутчиков, ознакомиться с их профилями и выбрать оптимальный маршрут и удобное время отправления. Водители, в свою очередь, могут заполнить свободные места в своем автомобиле, сократив расходы на топливо и автомобильные расходы. Это создает взаимовыгодные условия для обеих сторон.

Одним из ключевых факторов успешного функционирования приложений для подбора попутчиков является удобство использования и функциональность. Современные приложения предлагают пользователям встроенный мессенджер для быстрого и удобного общения, возможность создавать и управлять аккаунтами, а также инструменты для планирования маршрута и выбора попутчиков. Эти функции значительно упрощают процесс поиска и бронирования поездок.

В условиях высокой конкуренции на рынке транспортных приложений, разработчики стремятся обеспечить кроссплатформенную доступность своих сервисов. Это позволяет пользователям независимо от используемого устройства получать доступ ко всем функциям приложения и наслаждаться комфортом использования.

Особое внимание уделяется удобству интерфейса и качеству предоставляемых сервисов. Современные приложения используют интуитивно понятные интерфейсы и предлагают пользователям разнообразные функции для удобства планирования поездок, такие как автоматическое определение оптимального маршрута и расчет времени в пути.

Однако, несмотря на все преимущества, пользователи могут столкнуться с некоторыми трудностями. Например, в пиковые часы спрос на поездки может превышать предложение, что приводит к задержкам и увеличению стоимости. Кроме того, важно учитывать географические особенности и наличие инфраструктуры для удобного доступа к месту встречи с водителем.

В целом, мобильные приложения для подбора попутчиков представляют собой инновационное решение, способствующее развитию устойчивого и эффективного транспорта. Благодаря им пользователи могут экономить время и деньги, а также вносить свой вклад в снижение выбросов CO₂ и улучшение экологической ситуации.

Цель настоящей работы – разработка серверной части кроссплатформенного приложения для поиска попутчиков. Для достижения поставленной цели необходимо решить следующие задачи:

- провести анализ предметной области;
- разработать концептуальную модель программно-информационной системы;
- спроектировать и реализовать серверную часть программной системы;
- провести тестирование серверной части программной системы.

Структура и объем работы. Отчет состоит из введения, 4 разделов основной части, заключения, списка использованных источников, 2 приложений. Текст выпускной квалификационной работы равен 126 страницам.

Во введении сформулирована цель работы, поставлены задачи разработки, описана структура работы, приведено краткое содержание каждого из разделов.

В резюме стартап-проекта содержится основная информация о стартап-проекте: название, цели и стратегия, уникальность продукта, результаты, риски и перспективы проекта

В первом разделе на стадии описания технической характеристики предметной области приводится сбор информации об аналогах, а так же про-

изводится пользовательское исследование, на основе которого осуществляется разработка программной системы.

Во втором разделе на стадии технического задания приводятся требования к серверной части.

В третьем разделе на стадии технического проектирования представлены проектные решения для разрабатываемого приложения.

В четвертом разделе приводится список классов и их методов, использованных при разработке API, производится тестирование разработанного решения.

В заключении излагаются основные результаты работы, полученные в ходе разработки.

В приложении А представлен графический материал. В приложении Б представлены фрагменты исходного кода.

РЕЗЮМЕ СТАРТАП-ПРОЕКТА

Название: «Бизнес-проект «Кроссплатформенная программная система поиска попутчиков для автомобильных поездок»». Кроссплатформенное приложение для поиска попутчиков AutoStop – это программа, где все пользователи делятся на попутчиков и водителей. Водитель может создавать поездки и размещать их внутри приложения, в свою очередь, попутчики могут находить поездки и бронировать их. В приложении можно выстраивать коммуникацию между пользователями для обсуждения планов поездки через чаты.

Были проанализированы современные аналоги и выявлены основные проблемы, с которыми сталкиваются пользователи приложений поиска попутчиков для автомобильных поездок:

1. Высокая стоимость сервисного сбора и услуг, которые предоставляют приложения для поиска попутчиков. Основная часть приложений данной сферы, запрашивает сервисный сбор, который может достигать 30% от цены, которую указал водитель. Это побуждает пользователей действовать в обход системы, снимая бронь с поездки. Эти действия создают сложности для комфорtnого использования подобных сервисов.

2. Отсутствие desktop версий под операционные системы Mac os и Windows. Большинство сервисов имеет web сайты для бронирования поездок, однако этот подход создает уязвимость для неопытных пользователей, которые ошибочно могут зайти на сайт мошенников.

3. Некоторые сервисы имеют проблемы с оптимизацией и плохо адаптированы под слабые устройства.

4. Низкая пользовательская база. Так как практически все приложения для поиска попутчиков требуют сервисный сбор, база пользователей не растет. Для приложения, специализирующегося на автомобильных поездках, очень важно иметь активных пользователей. В противном случае, использование сервиса становится невозможным.

Кроссплатформенная программная система поиска попутчиков для автомобильных поездок AutoStop призвана решить эти проблемы, и стать удобным сервисом для взаимодействия попутчиков и водителей.

Актуальность данного сервиса связана с растущей потребностью в удобных и экономичных способах организации совместных автомобильных поездок. На сегодняшний день на рынке отсутствуют кроссплатформенные решения, которые бы полностью удовлетворяли потребности пользователей. Существующие приложения часто обременены высокими сервисными сборами и имеют ограниченные функциональные возможности. AutoStop решает эти проблемы, предлагая доступный и удобный сервис для водителей и попутчиков, который поддерживает популярные операционные системы и устройства.

Отличительные особенности программной системы по сравнению с существующими решениями и технологиями:

1. На данном этапе весь функционал приложения предоставляется бесплатно. Это решение направлено на привлечение потенциальных пользователей и быстрое расширение базы активных пользователей для обеспечения успешного функционирования сервиса в будущем.

2. AutoStop поддерживает популярные операционные системы и устройства, включая Mac OS, Windows, и Android. Это обеспечивает пользователям максимальную гибкость и удобство при использовании сервиса, позволяя им бронировать поездки и общаться с другими участниками независимо от используемого устройства.

В разработанном приложении реализованы следующие возможности:

Регистрация пользователей при помощи номера телефона под аккаунтом водителя или попутчика, изменения данных профиля, добавление автомобилей под учетной записью водителя, смена фотографии профиля, создание поездки под аккаунтом водителя, удаление созданной поездки, просмотр архива пользовательских поездок, поиск поездки, бронирование активных поездок, отмена бронирования, встроенный мессенджер позволяет поддерживать связь между водителем и попутчиком.

Идея создания проекта «AutoStop» возникла в январе 2023 года, однако разработка началась позже в связи с принятием решений, касательно концепции работы приложения. С июля 2024 года планируется расширение функционала приложения и развертывание на общедоступном сервере для тестирования работы приложения под нагрузкой, на этом этапе принимать участие в тестировании смогут пользователи по всей стране, с октября 2024 планируется полноценное оказание услуг. Оказание услуг не зависит от региона Российской Федерации.

Для защиты исключительных прав и правовой охраны необходимо в дальнейшем осуществить регистрацию программного средства.

Самыми значительными и высоковероятными рисками для проекта «AutoStop» являются приложения аналоги, у которых есть схожий функционал и большая аудитория пользователей. Перспективой дальнейшей разработки программной системы является наращивание аудитории, увеличение количества поездок, размещенных на сервере, а также расширение функционала посредством обновлений приложения.

Бизнес-цели:

1. Увеличить число зарегистрированных пользователей: 5000 пользователей через год. Для достижения этой цели можно использовать различные методы привлечения пользователей, например, рекламные кампании в социальных сетях, поисковую оптимизацию и т.д.
2. Увеличить количество поддерживаемых платформ до максимума, добавив в поддержку устройства на базе платформы IOS.
3. Добавить систему отзывов и рейтингов. Для повышения надежности и уверенности в успешном исходе каждой поездки.
4. Обеспечить надежную и бесперебойную работу серверной части. Необходимо провести исследование и расчеты рисков разных вариантов хостинга.
5. Выйти на стабильную прибыль через 2 года. Для достижения этой цели можно использовать различные методы монетизации системы, например, рекламу, введение комиссии за дальние поездки, и т.д.

1 Анализ предметной области

1.1 Общие принципы и понятия поиска попутчиков для поездки

Поиск попутчиков для поездки — это концепция, направленная на оптимизацию транспортных ресурсов путем совместного использования транспортных средств несколькими людьми, имеющими схожие маршруты и временные рамки. Эта практика получила новый импульс благодаря развитию цифровых технологий и увеличению осведомленности об экологических проблемах.

Принципы поиска попутчиков:

- Экономическая выгода. Совместные поездки позволяют участникам делить расходы, что делает путешествия более доступными. В условиях постоянно растущих цен на топливо и обслуживание автомобилей возможность снизить затраты становится важным преимуществом.
- Экологическая устойчивость. Совместные поездки способствуют снижению количества автомобилей на дорогах, что уменьшает выбросы вредных веществ в атмосферу. Это важный аспект в контексте глобальной борьбы с изменением климата.
- Социальные аспекты. Поиск попутчиков способствует укреплению социальных связей и взаимодействию между людьми. Социальные контакты, установленные в процессе совместных поездок, могут перерасти в крепкие дружеские отношения.
- Гибкость и адаптивность. Совместные поездки предоставляют участникам возможность выбирать наиболее удобные для них маршруты и время отправления, делая процесс путешествия более гибким и адаптивным к потребностям каждого.

Понятия, связанные с поиском попутчиков:

- Совместное использование транспорта. Основная концепция, подразумевающая использование одного транспортного средства несколькими людьми, что позволяет оптимизировать затраты и уменьшить нагрузку на дорожную инфраструктуру.

- Маршруты и расписание. Важным аспектом является согласование маршрутов и времени отправления между участниками. Это требует координации и планирования для обеспечения максимального удобства для всех попутчиков.

- Экономия на расходах. Деление расходов на топливо и обслуживание транспортного средства является значительным стимулом для участия в совместных поездках.

- Социальные взаимодействия. В процессе совместных поездок люди имеют возможность общаться, делиться опытом и устанавливать новые социальные связи.

Преимущества поиска попутчиков:

- Снижение нагрузки на дороги. Меньшее количество автомобилей на дорогах способствует уменьшению заторов и улучшению дорожной ситуации.

- Улучшение экологической обстановки. Уменьшение выбросов вредных веществ способствует улучшению качества воздуха и снижению негативного воздействия на окружающую среду.

- Повышение социальной активности. Поиск попутчиков способствует развитию социальных контактов и взаимодействий, что может быть полезно для личностного роста и создания новых дружеских связей.

Вызовы и трудности

- Координация участников. Согласование маршрутов и времени отправления может быть сложным процессом, особенно если участники имеют разные графики и предпочтения. Это требует гибкости и готовности к компромиссам.

- Непредвиденные обстоятельства. Изменения планов одного из участников могут повлиять на весь маршрут, требуя перестройки планов и поиска новых попутчиков.

- Доверие и безопасность. Одним из ключевых факторов успешного поиска попутчиков является доверие между участниками. Важно, чтобы все

участники чувствовали себя комфортно и безопасно в процессе совместных поездок.

1.2 Приложения для поиска попутчиков, их классификация

Приложения для поиска попутчиков – это цифровые платформы, предназначенные для объединения людей, которые хотят совершить совместные поездки. Эти приложения служат посредниками между водителями и пассажирами, предоставляя удобный интерфейс для поиска, бронирования и планирования поездок.

Основная цель приложений для поиска попутчиков – упростить процесс организации совместных поездок, обеспечивая комфорт и экономичность для всех участников. Водители, имеющие свободные места в автомобиле, могут размещать информацию о предстоящих поездках, а пассажиры, нуждающиеся в транспорте, могут искать и бронировать подходящие поездки.

Основные функции приложений для поиска попутчиков:

- Регистрация и создание профиля. Пользователи регистрируются в приложении, создают свои профили, указывая личную информацию и предпочтения по поездкам. Это позволяет другим участникам ознакомиться с профилем пользователя перед поездкой.

- Поиск и бронирование поездок. Водители размещают объявления с информацией о маршруте, времени отправления и количестве свободных мест. Пассажиры могут искать подходящие поездки, используя фильтры по различным параметрам, таким как местоположение, дата и время отправления, цена и предпочтения по уровню комфорта.

- Коммуникация между пользователями. Встроенные мессенджеры позволяют водителям и пассажирам уточнять детали поездки, договариваться о времени и месте встречи, а также обмениваться необходимой информацией.

- Оплата поездок. Приложения могут предлагать различные способы оплаты, включая предоплату через электронные платежные системы или

оплату наличными по факту поездки. Это обеспечивает гибкость и удобство для пользователей.

Преимущества приложений для поиска попутчиков:

- Быстрота поиска. В системах приложений уже есть размещенные поездки, которые попутчики могут забронировать в любую секунду, что ускоряет процесс поиска и бронирования.

- Удобство. Приложения предоставляют пользователям удобный интерфейс для поиска и бронирования поездок, что экономит время и усилия.

- Гибкость. Пользователи могут выбирать маршруты и время отправления, которые лучше всего соответствуют их потребностям и предпочтениям.

Модели взаимодействия. Приложения для поиска попутчиков могут использовать различные модели взаимодействия между водителями и пассажирами:

- Долгосрочные поездки. Пользователи могут находить попутчиков для регулярных и длительных поездок, таких как поездки на работу или учебу.

- Краткосрочные поездки. Приложения также позволяют находить попутчиков для разовых поездок на короткие расстояния, например, для поездок на мероприятия или в соседние города.

- Специальные маршруты. Некоторые приложения предлагают специализированные маршруты, такие как трансферы до аэропортов или курортов, что позволяет пользователям планировать свои поездки заранее.

Разновидности приложений. Существует несколько разновидностей приложений для поиска попутчиков в зависимости от их специализации и модели взаимодействия с пользователями:

- Городские поездки. Приложения, ориентированные на совместные поездки в пределах города, помогают разгрузить транспортную инфраструктуру и снизить заторы.

- Междугородные поездки. Эти приложения фокусируются на длительных поездках между городами, предоставляя пользователям возможность экономить на транспорте.

- Тематика путешествий. Некоторые приложения специализируются на поездках для определенных целей, таких как путешествия, туризм или деловые поездки.

Приложения для поиска попутчиков играют важную роль в современной транспортной экосистеме, предлагая удобные, экономичные и гибкие решения для совместных поездок. Они способствуют улучшению качества жизни пользователей, снижению транспортных расходов и оптимизации использования транспортных ресурсов.

1.3 Исследование аналогов

1.3.1 BlablaCar

BlablaCar – Приложение позиционирует себя как инструмент для поиска попутчиков. Является лидером на рынке среди приложений в данной сфере. Позволяет связываться с попутчиками через личные сообщения внутри приложения, а также через средства мобильной связи, после подтверждения поездки.

Основные плюсы данного аналога:

- Популярный сервис для поиска попутчиков. Является приложением лидером в данной сфере и практически не имеет конкурентов.
 - Богатый функционал приложения.
 - Продуманный дизайн.
 - Имеет встроенный мессенджер для обсуждения планов поездки.
 - Большая база попутчиков и водителей.
 - Возможность бронирования поездки заранее.
 - Позволяет искать попутчиков по системе «Буст», когда водитель может взять попутчика, проездом.
 - Имеет систему рейтинга.
 - Понятный сайт.
 - Имеет приложение на ОС андроид.
 - Встроенная интеграция с поиском автобусов.

- Возможность ставить точки на карте, для уточнения места встречи попутчика и водителя.

- Имеет поддержку пользователя.

Минусами сервиса BlaBlaCar являются следующие критерии:

- Возможны проблемы при использовании приложения на слабых android устройствах.

- Сервис платный. При использовании приложения, может взыматься плата при поездках между населенными пунктами на расстояние в более чем 100км.

- Не имеет desktop версию на Windows и Mac OS.

1.3.2 Едем.рф

Еще одним приложением для поиска попутчиков, является сервис Едем.рф. Данный сервис тоже пользуется большой популярностью и во многом схож с главным конкурентом – BlablaCar. Особенностью данного приложения является реализации грузоперевозок межгород. Приложение позволяет связываться с попутчиками 2 способами:

- Через внутренний чат приложения.
- Через средства мобильной связи в формате звонка.

Плюсы данного приложения:

- Популярный сервис для поиска попутчиков.
- Продуманный дизайн приложения.
- Имеет встроенный мессенджер для обсуждения планов поездки.
- Большая база попутчиков и водителей.
- Возможность бронирования поездки заранее.
- Имеет систему рейтинга.
- Встроенная интеграция с поиском автобусов.
- Возможность ставить точки на карте, для уточнения точки встречи попутчика и водителя.

- Имеет поддержку пользователя.
- Система отзывов и рейтингов.

Минусами сервиса едем.рф являются следующие критерии:

- Приложение имеет некоторые баги.
- Сервис платный. При использовании приложения взымается плата для водителей, при создании объявления о поездке.
- Не имеет desktop версию на Windows и Mac

1.3.3 Попутка.рус

Последним аналогом в данной предметной области является приложение Попутка.рус. Данное приложения не пользуется большой популярностью. Так же приложение имеет возможность публикации объявлений с предоставлением контактных данных.

Приложение позволяет поддерживать диалог между попутчиком и водителем 3 способами:

- Телефонная связь.
- Внутренний мессенджер внутри приложения.
- Сообщения в формате sms на телефон.

Приложение имеет следующие плюсы:

- Есть возможность создавать объявления, в том числе и для осуществления сделки купли/продажи.
- Функционал для связи водителя и попутчиков.
- Простое приложение, которое имеет малый вес и без проблем может работать даже на слабых устройствах.
- Есть возможность авторизации и создания аккаунта с привязкой только к номеру телефона.
- В процессе поиска водителя позволяет выбрать категорию поездки.

Минусы приложения Попутка.рус:

- Маленькая база пользователей. Найти попутчика проблематично.
- Функционал данного приложения в плане поиска попутчиков уступает аналогам.
- Менее привлекательный интерфейс.

1.4 Пользовательское исследование

В данном разделе описаны варианты использования приложения для поиска попутчиков, в зависимости от требований пользователей и различных ситуаций. Был проведен опрос, в котором пользователи подробно описали свой вариант использования приложения для достижения определенных целей, в виду особенностей и требованиям к системе.

Первый вариант использования приложения:

Мужчина среднего возраста планирует дальнюю поездку на своем автомобиле. Ему необходимо найти попутчиков для того, чтобы поездка на была слишком накладной. Мужчина до этого не использовал приложение, и планирует набрать небольшую базу попутчиков. Автомобиль: renault logan.

Он заходит в play market и скачивает приложение, проходит простую регистрацию, в ходе которой указывает свой номер телефона и личную информацию. Далее в самом приложении мужчина находит раздел размещения поездки. Система понимает, что у данного водителя еще не было осуществлено поездок и предлагает пройти дополнительную регистрацию для водителей, в ходе которой мужчина регистрирует свой автомобиль, указывает опыт вождения и дополнительную информацию, необходимую для осуществления поездки.

После прохождения всех этапов регистрации, мужчина размещает поездку через вышеупомянутый раздел. В качестве информации, он указывает точку, откуда он собирается начать свой путь, конечную точку маршрута, дату и время поездки. А также, указывает автомобиль, на котором собирается осуществить поездку.

Так как у данного водителя небольшой опыт использования приложения, водитель указывает цену поездки чуть ниже рыночной. Через некоторое время, водителю удается найти несколько попутчиков, которые звонят на его контактный номер телефона и обсуждают планы проведения поездки.

Попутчики и водитель встречаются в указанной точке в соответствии со временем начала поездки. После успешного проведения поездки, водитель получает денежные средства от попутчиков.

Второй вариант использования приложения:

Молодому парню требуется передать посылку для матери, которая находится в другом городе. У него нет необходимости в передачи посылки из рук в руки. Он решает воспользоваться приложением AutoStop. Для осуществления передачи.

Парень заходит в приложение и осуществляет поиск водителя через соответствующий раздел. В качестве информации, он указывает город отправления, конченую точку маршрута и дату начала поездки. В результате чего находит несколько водителей. Далее парень осуществляет выборку водителей.

Для осуществления передачи посылки парню не требуется бронировать место в автомобиле. Вместо этого он решает вести диалог через сообщения внутри приложения с водителем. В ходе диалога, водитель договаривается с парнем о передачи посылки за соответствующую плату. И обменивается контактными данными для осуществления передачи. Далее они встречаются на месте начальной точки маршрута, и парень передает посылку водителю. Водитель, в свою очередь, по окончанию поездки вручает посылку матери молодого человека и получает за это плату.

Третий вариант использования приложения:

Супружеская пара почтенного возраста планирует добраться из точки А в точку Б. Для них использования приложения вызывает множество трудностей в силу возраста. Вместо бронирования мест, для осуществления поездки со своего телефона, они просят свою дочь забронировать им места.

Для бронирования поездки, дочка входит в приложение и осуществляя подбор водителей в соответствующем разделе. Она тщательно просматривает водителей в соответствии с указанными критериями. Одним из таких является комфортный автомобиль.

После недолгих поисков, она находит водителя Михаила с достаточно большим стажем вождения на Volvo XC90. Понимая, что автомобиль может обеспечить комфортную поездку её родителям, она указывает в качестве брони 2 места. И пишет водителю о том, что в качестве попутчиков, будут её родители. На что Михаил соглашается. После чего, у девушки появляется возможность позвонить Михаилу и обсудить тонкости поездки. В ходе обсуждения узнается, что у водителя полностью заполнен багажник и в салоне автомобиля будут ехать еще 3 человека. Такой вариант, не устраивает девушку, и она отменяет бронь на данную поездку.

В процессе поиска, девушка находит водителя с достаточным стажем и комфортным автомобилем. В чате с водителем она обсуждает все тонкости и в итоге они договариваются о поездке.

Четвертый вариант использования приложения:

Женщина среднего возраста планирует деловую поездку в другой город и решает воспользоваться приложением для поиска попутчиков на своем компьютере с операционной системой Windows. Ей важно найти надежного водителя и комфортный автомобиль для поездки.

Женщина запускает приложение AutoStop, которое уже установлено на её компьютере. Она проходит стандартную процедуру регистрации, указав свою личную информацию и номер телефона. После успешной регистрации она переходит в раздел поиска поездок.

В интерфейсе поиска она указывает точку отправления, конечный пункт назначения, дату поездки. Система отображает список доступных водителей, которые соответствуют её запросу.

После тщательного изучения она выбирает водителя с комфорtnым автомобилем, Audi A6. Женщина отправляет запрос на бронирование места, указав, что она путешествует одна и у нее будет небольшой багаж.

Женщина обсуждает детали поездки через встроенный чат в приложении. Они договариваются о месте встречи.

2 Техническое задание

2.1 Основание для разработки

Основанием для разработки является задание на выпускную квалификационную работу бакалавра «Бизнес-проект "Кроссплатформенная программная система поиска попутчиков для автомобильных поездок". Разработка серверной части».

2.2 Назначение разработки

Функциональное назначение разрабатываемой кроссплатформенной программной системы заключается в предоставлении пользователям функциональных возможностей для поиска и создания автомобильных поездок.

Предполагается, что разрабатываемая программа система будет использоваться широким кругом лиц, заинтересованных в соместных поездах по России.

2.3 Требования к программной системе

2.3.1 Требования к данным программной системы

Входными данными для серверной части программной системы являются:

- данные пользователя при регистрации;
- запрос регистрации пользователя как водителя;
- данные пользователя для входа в систему;
- запрос на добавление автомобиля;
- запрос на удаление автомобиля;
- запрос на изменение пользовательских данных;
- запрос на изменение данных об автомобилях;
- фото профиля;
- данные о планируемой поездке;
- запрос на удаление поездки;

- запрос на поиск активных поездок;
- запрос на добавление попутчика к поездке;
- запрос на удаление пользователя из поездки;
- запрос на создание чата;
- запрос на просмотр активных чатов;
- текст сообщения;
- запрос на просмотр сообщений по чату;
- текст комментария;
- запрос просмотр комментариев по профилю пользователя.

Выходными данными для серверной части программной системы являются:

- данные профиля для попутчика;
- данные профиля для водителя;
- информация об автомобилях;
- данные об активных поездках для попутчика;
- данные об активных поездках для водителя;
- информацию об архивных поездках;
- фото профиля;
- список активных чатов;
- список сообщений по чату;
- список доступных поездок по заданным критериям поиска;
- отзывы и рейтинг пользователя;

2.3.2 Функциональные требования к программной системе

В разрабатываемой программной системе для пользовательской части должны быть реализованы следующие функции:

1. Регистрация пользователя под аккаунтом попутчика.
2. Регистрация пользователя под аккаунтом водителя.
3. Получение информации о пользовательских данных.
4. Редактирование сведений о пользователе.
5. Получение информации об автомобилях водителя.

6. Добавление автомобиля если пользователь является водителем.
7. Редактирование сведений об автомобиле.
8. Удаление автомобиля.
9. Отправка фотографии на пользовательскую часть программной системы.
10. Смена фотографии пользователя.
11. Получение информации о пользовательских поездках.
12. Создание поездки под аккаунтом водителя.
13. Удаление поездки под аккаунтом водителя.
14. Добавление в архив не актуальных поездок.
15. Поиск поездки по заданным критериям.
16. Бронирование поездки.
17. Отмена брони.
18. Создание чата с водителем под аккаунтном попутчика.
19. Получение информации о пользовательских чатах.
20. Получение сообщений по указанному чату.
21. Добавление нового сообщения.
22. Получение информации об отзывах и оценках.
23. Добавление отзыва и оценки.
24. Удаление отзыва.

На рисунках 2.1-2.3 представлены функциональные требования к системе в виде диаграммы прецедентов нотации UML.

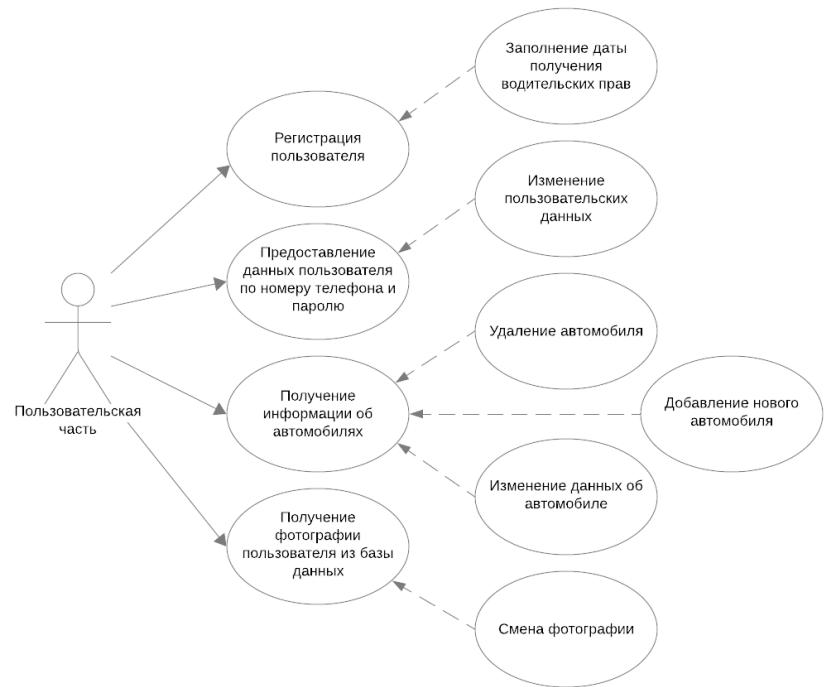


Рисунок 2.1 – Диаграмма вариантов использования, часть 1



Рисунок 2.2 – Диаграмма вариантов использования, часть 2

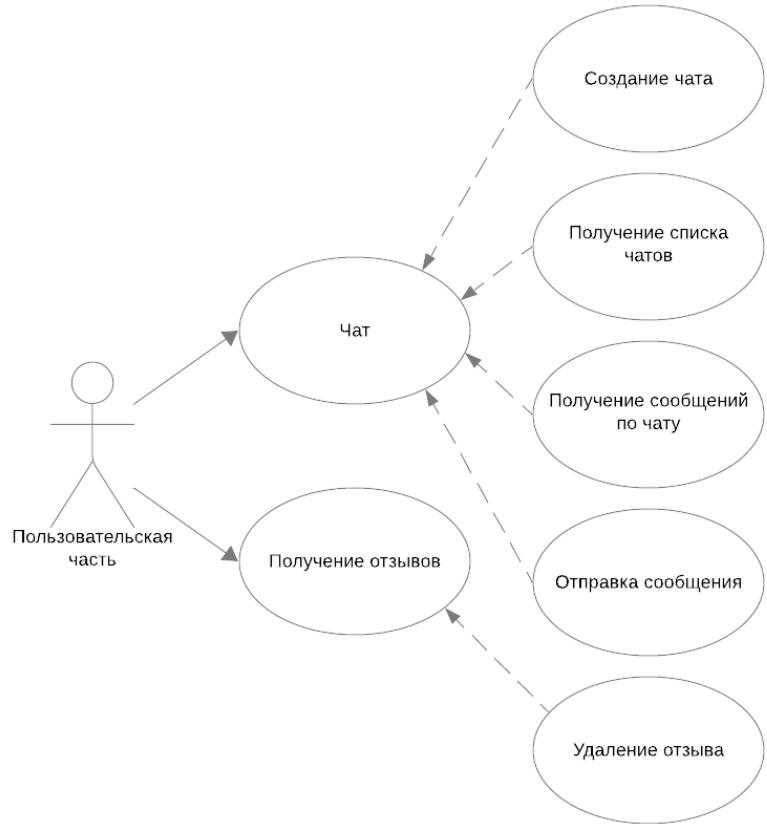


Рисунок 2.3 – Диаграмма вариантов использования, часть 3

2.3.2.1 Вариант использования «Регистрация пользователя»

Заинтересованные лица и их требования: пользователи программной системы, которые хотят зарегистрировать аккаунт, с целью получения возможности использовать функционал приложения для подбора попутчиков. Предусловие: на пользовательском устройстве открыто приложение AutoStop. Постусловие: пользователь регистрирует аккаунт как попутчик. Основной успешный сценарий:

1. Со стороны пользовательской части на сервер отправляется URI запрос с пользовательскими данными в формате json файла.
2. Сервер отслеживает запрос, проверяя возможность регистрации под переданным номером телефона.
3. Сервер запускает скрипт на добавление пользователя в базу данных.
4. Сервер отправляет на frontend статус OK.

2.3.2.2 Вариант использования «Регистрация водителя»

Заинтересованные лица и их требования: пользователи программной системы, которые хотят расширить возможности своей учетной записи, с целью создания поездок в используемом приложении. Предусловие: пользователь уже осуществил базовую регистрацию аккаунта. Постусловие: пользователь регистрирует аккаунт как водитель. Основной успешный сценарий:

1. Со стороны пользовательской части на сервер отправляется URI запрос с номером телефона пользователя и датой получения водительского удостоверения.
2. Сервер отслеживает запрос, и дополняет информацию по аккаунту пользователя.
3. Сервер отправляет на frontend статус OK.

2.3.2.3 Вариант использования «Авторизация»

Заинтересованные лица и их требования: пользователи программной системы, которые хотят войти в систему подбора попутчиков для автомобильных поездок под своей учетной записью. Предусловие: пользователь прошел регистрацию и имеет аккаунт. Постусловие: пользователь входит в систему. Основной успешный сценарий:

1. Со стороны пользовательской части на сервер отправляется URI запрос с номером телефона пользователя и паролем в формате json файла.
2. Сервер отслеживает запрос, и находит пользователя по переданным данным.
3. Сервер отправляет на frontend json файл с пользовательскими данными.

2.3.2.4 Вариант использования «Изменение пользовательских данных»

Заинтересованные лица и их требования: пользователи программной системы, которые хотят изменить личные данные в системе подбора попут-

чиков для автомобильных поездок. Предусловие: пользователь осуществил вход под своей учетной записью. Постусловие: пользователь изменяет данные аккаунта. Основной успешный сценарий:

1. Со стороны пользовательской части на сервер отправляется URI запрос с номером телефона пользователя и измененными данными в формате json файла.
2. Сервер отслеживает запрос, и находит пользователя по переданным данным.
3. Сервер запускает скрипт на изменение в БД.
4. Сервер отправляет на frontend статус OK.

2.3.2.5 Вариант использования «Получение информации об автомобилях»

Заинтересованные лица и их требования: пользователи программной системы, которые хотят получить список всех автомобилей, доступных под используемым профилем. Предусловие: пользователь осуществил вход под своей учетной записью и является водителем в системе. Постусловие: пользователь получает список машин. Основной успешный сценарий:

1. Со стороны пользовательской части на сервер отправляется URI запрос с номером телефона пользователя.
2. Сервер отслеживает запрос, и находит все автомобили зарегистрированные на данного пользователя.
3. Сервер отправляет на frontend список всех найденных машин.

2.3.2.6 Вариант использования «Добавление нового автомобиля»

Заинтересованные лица и их требования: пользователи программной системы, которые хотят добавить автомобиль, для дальнейшего использования в поездках. Предусловие: пользователь осуществил вход под своей учетной записью и является водителем в системе. Постусловие: пользователь добавляет автомобиль в свой профиль. Основной успешный сценарий:

1. Со стороны пользовательской части на сервер отправляется URI запрос с номером телефона пользователя и данными об автомобиле в формате json файла.
2. Сервер отслеживает запрос, и добавляет сведения об автомобиле указанному пользователю.
3. Сервер отправляет на frontend сообщение со статусом OK.

2.3.2.7 Вариант использования «Изменение данных об автомобиле»

Заинтересованные лица и их требования: пользователи программной системы, которые изменить указанные данные о машине в профиле. Предусловие: пользователь осуществил вход под своей учетной записью и является водителем в системе. У пользователя есть хотя бы 1 автомобиль в профиле. Постусловие: пользователь изменяет данные о машине. Основной успешный сценарий:

1. Со стороны пользовательской части на сервер отправляется URI запрос с ГРЗ автомобиля и обновленными данными в формате json файла.
2. Сервер отслеживает запрос, и обновляет данные, на новые.
3. Сервер отправляет на frontend сообщение со статусом OK.

2.3.2.8 Вариант использования «Удаление автомобиля»

Заинтересованные лица и их требования: пользователи программной системы, которые хотят удалить существующий автомобиль из профиля. Предусловие: пользователь осуществил вход под своей учетной записью и является водителем в системе. У пользователя есть хотя бы 1 автомобиль в профиле. Постусловие: пользователь удаляет данные о машине. Основной успешный сценарий:

1. Со стороны пользовательской части на сервер отправляется URI запрос с ГРЗ автомобиля который необходимо удалить.
2. Сервер отслеживает запрос, и находит указанную машину.
3. Сервер запускает скрипт на удаление автомобиля из БД.

4. Сервер отправляет на frontend сообщение со статусом OK.

2.3.2.9 Вариант использования «Получение фото профиля»

Заинтересованные лица и их требования: пользователи программной системы, которые осуществляют вход под своей учетной записью. Предуслугование: пользователь прошел базовую регистрацию и имеет аккаунт в системе. Постусловие: пользователь получает фотографию своего профиля. Основной успешный сценарий:

1. Со стороны пользовательской части на сервер отправляется URI запрос с номером телефона пользователя.
2. Сервер отслеживает запрос, и находит фото профиля в базе данных.
3. Сервер отправляет на frontend изображение.

2.3.2.10 Вариант использования «Смена фото профиля»

Заинтересованные лица и их требования: пользователи программной системы, которые хотят заменить фотографию профиля на новую. Предуслугование: пользователь прошел базовую регистрацию и имеет аккаунт в системе. Постусловие: пользователь изменяет фотографию своего профиля. Основной успешный сценарий:

1. Со стороны пользовательской части на сервер отправляется URI запрос с номером телефона пользователя и фотографией в формате json файла.
2. Сервер отслеживает запрос, и находит пользователя с указанным номером телефона.
3. Сервер запускает скрипт на изменение фотографии в БД.
4. Сервер отправляет на frontend сообщение со статусом OK.

2.3.2.11 Вариант использования «Получение информации о статусе аккаунта

Заинтересованные лица и их требования: пользователи программной системы, которые хотят создать новую поездку в системе. Предуслугование: пользователь вошел под своим аккаунтом в приложение. Постусловие: пользова-

тель получает возможность создать поездку в системе. Основной успешный сценарий:

1. Со стороны пользовательской части на сервер отправляется URI запрос с номером телефона пользователя.
2. Сервер отслеживает запрос, и находит запись с датой получения водительского удостоверения.
3. Сервер отправляет на frontend json файл с датой получения прав.

2.3.2.12 Вариант использования «Создание поездки»

Заинтересованные лица и их требования: пользователи программной системы, которые хотят создать новую поездку в системе. Предусловие: пользователь вошел под своим аккаунтом в приложение и прошел проверку на наличие водительского удостоверения. Постусловие: пользователь создает поездку в системе для подбора попутчиков. Основной успешный сценарий:

1. Со стороны пользовательской части на сервер отправляется URI запрос с данными о поездке, которую необходимо создать, в формате json файла.
2. Сервер отслеживает запрос, и добавляет данные о поездке в БД.
3. Сервер отправляет на frontend сообщение со статусом OK.

2.3.2.13 Вариант использования «Удаление поездки»

Заинтересованные лица и их требования: пользователи программной системы, у которых есть необходимость удалении существующей поездки. Предусловие: пользователь вошел под своим аккаунтом в приложение. В системе есть хотя бы одна поездка созданная до данным профилем. Постусловие: пользователь удаляет поездку в системе для подбора попутчиков. Основной успешный сценарий:

1. Со стороны пользовательской части на сервер отправляется URI запрос уникальным идентификатором удаляемой поездки.
2. Сервер отслеживает запрос, и удаляет поездку из системы.
3. Сервер отправляет на frontend сообщение со статусом OK.

2.3.2.14 Вариант использования «Получение списка поездок по заданным критериям поиска»

Заинтересованные лица и их требования: пользователи программной системы, у которых есть необходимость в поиске подходящей поездки. Предусловие: пользователь вошел под своим аккаунтом в приложение. Постусловие: попутчик получает список поездок по заданным условиям поиска. Основной успешный сценарий:

1. Со стороны пользовательской части на сервер отправляется URI запрос с данными, на основе которых необходимо осуществить выборку из существующих поездок в системе.
2. Сервер отслеживает запрос, и находит все поездки, удовлетворяющие условиям поиска.
3. Сервер отправляет на frontend список всех найденных поездок в формате json файла.

2.3.2.15 Вариант использования «Бронирование поездки»

Заинтересованные лица и их требования: пользователи программной системы, целью которых является бронирование поездки в системе для подбора попутчиков. Предусловие: пользователь вошел под своим аккаунтом в приложение. В системе имеются активные поездки. Постусловие: попутчик бронирует поездку. Основной успешный сценарий:

1. Со стороны пользовательской части на сервер отправляется URI запрос с пользовательскими данными и информацией о бронируемой поездке.
2. Сервер отслеживает запрос, и записывает пользователя в поездку.
3. Сервер отправляет на frontend ответ со статусом OK.

2.3.2.16 Вариант использования «Отмена брони»

Заинтересованные лица и их требования: пользователи программной системы, которые хотят отменить бронирование поездки. Предусловие: пользователь вошел под своим аккаунтом в приложение. У пользователя есть

бронь. Постусловие: попутчик отменяет бронь. Основной успешный сценарий:

1. Со стороны пользовательской части на сервер отправляется URI запрос с пользовательскими данными и информацией о бронируемой поездке.
2. Сервер отслеживает запрос, и удаляет из базы данных информацию о брони.
3. Сервер отправляет на frontend ответ со статусом OK.

2.3.2.17 Вариант использования «Получения списка всех поездок для водителя»

Заинтересованные лица и их требования: пользователи программной системы, получить список всех созданных поездок данным водителем. Предусловие: пользователь вошел под своим аккаунтом в приложение. У пользователя есть хотя бы одна созданная поездка. Постусловие: водитель получает список всех созданных поездок. Основной успешный сценарий:

1. Со стороны пользовательской части на сервер отправляется URI запрос с номером телефона водителя.
2. Сервер отслеживает запрос, и находит пользовательские поездки.
3. Сервер отправляет на frontend список всех созданных поездок со статусом «водитель» в формате json файла.

2.3.2.18 Вариант использования «Получения списка всех поездок для водителя»

Заинтересованные лица и их требования: пользователи программной системы, получить список всех созданных поездок данным водителем. Предусловие: пользователь вошел под своим аккаунтом в приложение. У пользователя есть хотя бы одна созданная поездка. Постусловие: водитель получает список всех созданных поездок. Основной успешный сценарий:

1. Со стороны пользовательской части на сервер отправляется URI запрос с номером телефона водителя.
2. Сервер отслеживает запрос, и находит пользовательские поездки.

3. Сервер отправляет на frontend список всех созданных поездок со статусом «водитель» в формате json файла.

2.3.2.19 Вариант использования «Получения списка всех поездок для попутчика»

Заинтересованные лица и их требования: пользователи программной системы, получить список всех забронированных поездок. Предусловие: пользователь вошел под своим аккаунтом в приложение. У пользователя есть хотя бы одна забронированная поездка. Постусловие: попутчик получает список всех созданных поездок. Основной успешный сценарий:

1. Со стороны пользовательской части на сервер отправляется URI запрос с номером телефона попутчика.
2. Сервер отслеживает запрос, и находит пользовательские поездки.
3. Сервер отправляет на frontend список всех забронированных поездок в формате json файла.

2.3.2.20 Вариант использования «Создание чата»

Заинтересованные лица и их требования: пользователи программной системы, которые хотят связаться с водителем для обсуждения планов поездки. Предусловие: пользователь вошел под своим аккаунтом в приложение и осуществил поиск поездок. Постусловие: попутчик создает чат с водителем. Основной успешный сценарий:

1. Со стороны пользовательской части на сервер отправляется URI запрос с номерами телефона попутчика и водителя.
2. Сервер отслеживает запрос, и проверяет наличие чата.
3. Сервер создает новый чат.
4. Сервер отправляет на frontend информацию о новом созданном чате в формате json файла.

2.3.2.21 Вариант использования «Получение списка чатов»

Заинтересованные лица и их требования: пользователи программной системы, которые хотят получить полную информацию по имеющимся чатам. Предусловие: пользователь вошел под своим аккаунтом в приложение и имеет хотя бы 1 чат. Постусловие: пользователь получает полный список чатов. Основной успешный сценарий:

1. Со стороны пользовательской части на сервер отправляется URI запрос с номерами телефона пользователя.
2. Сервер отслеживает запрос, и находит информацию по чатам.
3. Сервер отправляет на frontend список всех чатов в формате json файла.

2.3.2.22 Вариант использования «Получение сообщений по чату»

Заинтересованные лица и их требования: пользователи программной системы, которые хотят получить все сообщения по выбранному чату. Предусловие: пользователь вошел под своим аккаунтом в приложение и имеет хотя бы 1 чат с сообщениями. Постусловие: пользователь получает полный список сообщений. Основной успешный сценарий:

1. Со стороны пользовательской части на сервер отправляется URI запрос с уникальным идентификатором чата.
2. Сервер отслеживает запрос, и находит сообщения по указанному чату.
3. Сервер отправляет на frontend список всех сообщений в формате json файла.

2.3.2.23 Вариант использования «Отправка сообщения»

Заинтересованные лица и их требования: пользователи программной системы, которые хотят установить связь с собеседником через отправку сообщений. Предусловие: пользователь вошел под своим аккаунтом в приложе-

ние и имеет хотя бы 1 чат. Постусловие: пользователь отправляет сообщение собеседнику. Основной успешный сценарий:

1. Со стороны пользовательской части на сервер отправляется URI запрос с уникальным идентификатором чата и текстом сообщения в формате json файла.
2. Сервер отслеживает запрос, и добавляет сообщение в БД.
3. Сервер отправляет на frontend ответ со статусом OK.

2.3.2.24 Вариант использования «Получение отзывов»

Заинтересованные лица и их требования: пользователи программной системы, которые хотят просмотреть список всех отзывов по профилю. Предусловие: пользователь имеет хотя бы 1 отзыв. Постусловие: пользователь получает список всех отзывов. Основной успешный сценарий:

1. Со стороны пользовательской части на сервер отправляется URI запрос с номером телефона пользователя.
2. Сервер отслеживает запрос, и производит поиск всех отзывов.
3. Сервер отправляет на frontend список всех отзывов.

2.3.2.25 Вариант использования «Удаление отзыва»

Заинтересованные лица и их требования: пользователи программной системы, которые хотят удалить оставленный отзыв. Предусловие: пользователь имеет хотя бы 1 отзыв. Постусловие: пользователь удаляет свой отзыв. Основной успешный сценарий:

1. Со стороны пользовательской части на сервер отправляется URI запрос с уникальным идентификатором отзыва.
2. Сервер отслеживает запрос, и производит удаление отзыва.
3. Сервер отправляет на frontend ответ со статусом OK.

2.3.3 Нефункциональные требования к программной системе

2.3.3.1 Требования к надежности

В процессе работы серверной части программной системы могут произойти следующие аварийные ситуации:

- Потеря доступа к сети Интернет. Возможные причины включают проблемы с сетью, сбои маршрутизаторов или изменения в конфигурации сети.
- Аварийное отключение электропитания. Это может произойти из-за технических проблем в data-центре, аварий на электросетях или природных катаклизмов.
- Сбой операционной системы сервера. Операционная система может выйти из строя по причине аппаратных или программных сбоев, ошибок обновлений или атак вредоносного ПО.

Для минимизации вероятности возникновения аварийных событий серверные компоненты программной системы должны быть размещены на выделенных серверах в data-центрах хостинг-провайдеров. Операционная система должна получать регулярные накопительные обновления.

2.3.3.2 Требования к безопасности

Для защиты системы от несанкционированного доступа каждый запрос к базе данных должен проходить через специальный набор правил. Пользователь, не имеющий доступа к определенным данным должен не иметь возможности взаимодействовать с ними.

Для доступа к приложению пользователю необходимо пройти авторизацию в системе.

Для возможности авторизации в системе, пользователь должен пройти регистрацию.

Функции доступные водительскому акаунту не должны быть доступны для попутчика.

2.3.3.3 Требования к программному обеспечению

Для реализации серверной части программной системы должен быть использован язык высокого уровня C#.

В качестве системы управления базой данных (СУБД) должна быть использована система Microsoft SQL Server, разработанная компанией Microsoft.

Для реализации модуля чатов и сообщений необходим асинхронный подход, для ускорения работы системы и оптимизации используемых ресурсов сервера.

За обеспечение целостности данных должен отвечать автоматический сервис, который запускается на сервере каждые 10 минут. С целью добавления неактуальных поездок в архив. Это снизит нагрузку при поиске данных, связанных с пользовательскими поездками.

2.3.3.4 Требования к аппаратному обеспечению

Для обеспечения стабильной и эффективной работы написанной программной реализации, сервер должен соответствовать следующим требованиям к аппаратному обеспечению:

Процессор (CPU)

- Тип: Многоядерный процессор с поддержкой технологии многопоточности.
- Рекомендуемые модели: Intel Xeon серии E5 или выше, AMD EPYC или Ryzen 7 и выше.
- Частота: Минимум 2.4 ГГц на ядро.
- Количество ядер: Минимум 4 физических ядра.

Оперативная память (RAM)

- Объем: Минимум 16 ГБ, рекомендуется 32 ГБ и выше в зависимости от нагрузки и количества одновременно работающих приложений.

- Тип: DDR4 с частотой не ниже 2400 МГц.

Накопитель (SSD/HDD)

- Тип: SSD для основной системы и данных, HDD можно использовать для хранения архивов и резервных копий.

- Объем: Минимум 500 ГБ для SSD, рекомендуется 1 ТБ и более.

- Скорость чтения/записи: Не менее 500 МБ/с для SSD.

Сетевое оборудование

- Сетевая карта: Гигабитная Ethernet-карта (1 Gbps).

- Дополнительно: Поддержка резервирования сетевых подключений для обеспечения отказоустойчивости.

Источник бесперебойного питания (ИБП)

- Мощность: Достаточная для поддержания работы сервера в течение минимум 30 минут при отключении электропитания.

- Функции: Возможность автоматического корректного завершения работы сервера при длительном отключении питания.

Система охлаждения

- Тип: Эффективная система охлаждения для поддержания стабильной температуры процессора и других компонентов.

- Дополнительно: Наличие нескольких вентиляторов и/или жидкостной системы охлаждения для серверов с высокими нагрузками.

Корпус и питание

- Корпус: Серверный корпус форм-фактора Tower или Rack, обеспечивающий хорошую вентиляцию.

- Блок питания: Сертифицированный блок питания с мощностью не менее 750 Вт, с запасом мощности для дополнительного оборудования.

Дополнительные требования

- Резервирование: Возможность установки резервных блоков питания и дисков для обеспечения отказоустойчивости.

- Мониторинг: Встроенные системы мониторинга состояния компонентов и температуры.

2.4 Требования к оформлению документации

Требования к стадиям разработки программ и программной документации для вычислительных машин, комплексов и систем независимо от их назначения и области применения, этапам и содержанию работ устанавливаются ГОСТ 19.102–77. Программная документация должна включать в себя:

- Анализ предметной области.
- Техническое задание.
- Технический проект.
- Рабочий проект.

3 Технический проект

3.1 Общая характеристика организации решения задачи

Полное наименование системы: кроссплатформенная программная система подбора попутчиков для автомобильных поездок.

Разрабатываемая программа система представляет собой платформу, позволяющую пользователям находить попутчиков для созданных автомобильных поездок. Данная система построена на клиент-серверной архитектуре.

Сервер – backend-модуль[21][11][31], обрабатывающий запросы клиентов. Сервер в данной программной системе один.

Функции, выполняемые на стороне сервера:

- хранение, защита и доступ к данным;
- обработка поступающих клиентских запросов;
- поддержание целостности и актуальности данных;
- формирование и отправка ответов клиентам.

Одним из основных компонентов системы является база данных. В ней хранится практически вся информация, необходимая для работы системы, от данных пользователей до информации о предлагаемых и запрашиваемых поездках, а так же сведения об автомобилях, мессенджер, отзывы и рейтинг.

Задачи разработки включают:

- проектирование и создание базы данных;
- разработку фонового сервиса для поддержания актуальности данных о поездках;
- проектирование и создание моделей данных;
- разработку контроллеров для обработки запросов;
- реализацию функционала для взаимодействия с базой данных.

3.2 Обоснование выбора технологий проектирования

Используемые для создания программно-информационной системы языки и технологии отвечают современным практикам разработки, позволяют достичь высокой производительности и отказоустойчивости программы.

3.2.1 Язык программирования C#

C#[1][2][3] является одним из наиболее популярных и мощных языков программирования, который был разработан корпорацией Microsoft в рамках платформы .NET[5][6][7][8]. Одним из основных преимуществ C# является его современный синтаксис, который сочетает в себе простоту и выразительность, что делает его удобным для изучения и использования как начинающими, так и опытными программистами. Этот язык позволяет разработчикам писать чистый, понятный и поддерживаемый код, что особенно важно при создании крупных и сложных приложений.

Еще одной значимой чертой C# является его поддержка объектно-ориентированного программирования (ООП)[22][23][24]. Это позволяет разработчикам создавать приложения, основываясь на принципах инкапсуляции, наследования и полиморфизма, что способствует созданию гибких и масштабируемых программных решений. ООП также облегчает повторное использование кода, что снижает затраты на разработку и поддержку программного обеспечения.

Богатый набор библиотек и инструментов, доступных для C#, существенно упрощает разработку программ. Эти библиотеки покрывают широкий спектр задач — от работы с базами данных и сетевыми взаимодействиями до создания пользовательских интерфейсов и работы с файлами. Благодаря этому разработчики могут сосредоточиться на решении бизнес-задач, не тратя много времени на создание базовой функциональности с нуля.

Среда разработки Visual Studio[9][10][13][14], в которой часто ведется разработка на C#, предлагает мощные средства для написания, отладки и тестирования кода. Интегрированные средства автоматизации, рефакторинга и

анализа кода помогают разработчикам повышать продуктивность и качество конечного продукта. Это позволяет быстро находить и исправлять ошибки, а также оптимизировать код для повышения его производительности.

C# также обладает высокой безопасностью типов, что предотвращает многие распространенные ошибки, такие как доступ к неинициализированным переменным или некорректное приведение типов. Управление памятью в C# осуществляется с помощью сборщика мусора, что упрощает процесс управления ресурсами и снижает вероятность утечек памяти.

Кроме того, сообщество разработчиков C# активно и постоянно расстет, предоставляя богатый выбор ресурсов для обучения и обмена опытом. Это включает в себя форумы, блоги, видеоуроки и другие формы поддержки, которые помогают разработчикам быстро находить ответы на возникающие вопросы и постоянно совершенствовать свои навыки.

Таким образом, выбор C# в качестве языка программирования предоставляет разработчикам мощные инструменты для создания современных, эффективных и надежных приложений, что делает его отличным выбором для самых разнообразных проектов.

3.2.2 Фреймворк ASP.net

ASP.NET[15][16][17][18] — это мощная и гибкая платформа для разработки веб-приложений, созданная корпорацией Microsoft. Один из основных факторов, делающих ASP.NET привлекательным выбором для разработчиков, — это его интеграция с экосистемой .NET. Это позволяет использовать все возможности языка программирования C#, а также широкий набор библиотек и инструментов, что существенно ускоряет процесс разработки и упрощает создание высокопроизводительных и масштабируемых веб-приложений.

ASP.NET предоставляет разработчикам множество возможностей для создания динамических веб-страниц и приложений. Он поддерживает модель программирования, которая включает в себя серверные элементы управления, такие как формы и сетки данных, что позволяет создавать интерак-

тивные и отзывчивые пользовательские интерфейсы с минимальными усилиями. Это особенно важно для создания сложных веб-приложений, которые требуют высокого уровня интерактивности и пользовательского опыта.

Одним из ключевых преимуществ ASP.NET является его высокая производительность. Платформа оптимизирована для обработки больших объемов запросов и может эффективно управлять ресурсами, что позволяет создавать быстрые и отзывчивые веб-приложения. ASP.NET также поддерживает асинхронное программирование, что улучшает масштабируемость приложений и снижает нагрузку на серверы.

Безопасность является еще одной важной чертой ASP.NET. Платформа предоставляет встроенные механизмы защиты от распространенных угроз веб-безопасности, таких как межсайтовый скрипting (XSS)[19][20], SQL-инъекции[25][26] и другие виды атак. Это позволяет разработчикам сосредоточиться на функциональности приложений, не беспокоясь о внедрении сложных мер безопасности.

ASP.NET также поддерживает модульность и повторное использование кода. Это достигается благодаря компонентам, таким как контроллеры и представления в ASP.NET MVC[27][28][29][32], которые могут быть легко переработаны и использованы в различных частях приложения. Это способствует более эффективной разработке и снижает затраты на поддержку кода.

Кроме того, сообщество разработчиков ASP.NET активно и постоянно растет. Это обеспечивает обширную базу знаний и ресурсов, доступных для обучения и обмена опытом. Форумы, блоги, документация и видеоуроки помогают разработчикам быстро находить решения для возникающих проблем и улучшать свои навыки.

Таким образом, выбор ASP.NET в качестве платформы для разработки веб-приложений предоставляет разработчикам мощные инструменты и возможности для создания надежных, безопасных и высокопроизводительных веб-приложений. Платформа сочетает в себе современные технологии и подходы, что делает её отличным выбором для самых разнообразных проектов, от небольших сайтов до крупных корпоративных приложений.

3.2.3 Расширение T-SQL

T-SQL (Transact-SQL) — это расширение стандартного языка SQL (Structured Query Language), разработанное корпорацией Microsoft для управления и манипулирования данными в реляционных базах данных, таких как Microsoft SQL Server. Одной из ключевых причин выбора T-SQL[30][33] является его тесная интеграция с Microsoft SQL Server[34][35], что делает его мощным инструментом для разработки сложных запросов, процедур и скриптов, необходимых для управления данными и обеспечения их целостности.

Одним из значимых преимуществ T-SQL является его расширенные возможности по сравнению с обычным SQL. T-SQL включает в себя дополнительные конструкции, такие как циклы, условные операторы и встроенные функции, что позволяет разработчикам создавать более сложные и функциональные запросы. Это особенно важно для выполнения сложных бизнес-логик и операций с данными, которые требуют более гибких и мощных инструментов.

T-SQL также поддерживает создание хранимых процедур, триггеров и пользовательских функций. Хранимые процедуры позволяют выполнять сложные операции над данными, объединяя несколько SQL-запросов в один логический блок, что упрощает управление кодом и повышает его повторное использование. Триггеры позволяют автоматически выполнять определенные действия при изменении данных в таблице, что помогает поддерживать целостность данных и автоматизировать рутинные задачи.

Еще одной важной чертой T-SQL является его мощные средства для обработки и анализа данных. T-SQL предоставляет широкий набор функций для работы с текстом, датами, числами и другими типами данных, что облегчает выполнение различных операций над данными. Кроме того, T-SQL поддерживает создание временных таблиц и курсоров, что позволяет разработчикам эффективно управлять временными данными и выполнять сложные операции над ними.

Производительность также является важным аспектом T-SQL. Разработчики могут оптимизировать запросы с помощью индексов, оптимизатора запросов и других инструментов, что позволяет значительно повысить скорость выполнения операций над данными. Это особенно важно для работы с большими объемами данных, где производительность имеет критическое значение.

T-SQL также обеспечивает высокую степень безопасности данных. Разработчики могут использовать механизмы управления правами доступа, чтобы контролировать, кто и какие операции может выполнять над данными. Это помогает защитить данные от несанкционированного доступа и обеспечивать их конфиденциальность и целостность.

Кроме того, сообщество разработчиков T-SQL активно и постоянно расшаряет, что обеспечивает широкий доступ к ресурсам для обучения и обмена опытом. Форумы, блоги, документация и видеоуроки помогают разработчикам быстро находить ответы на возникающие вопросы и совершенствовать свои навыки.

Таким образом, выбор T-SQL в качестве языка для работы с базами данных предоставляет разработчикам мощные и гибкие инструменты для управления данными, выполнения сложных операций и обеспечения их целостности и безопасности. Это делает T-SQL отличным выбором для разработки и поддержки надежных и эффективных решений для управления данными в корпоративных и других приложениях.

3.2.4 Протокол связи HTTP

HTTP (Hypertext Transfer Protocol) — это основополагающий протокол для передачи данных в интернете, который служит базой для обмена информацией между веб-серверами и клиентами, такими как браузеры. Протокол был разработан в начале 1990-х годов в Европейской организации по ядерным исследованиям (CERN) под руководством Тима Бернерса-Ли и Роберта Кайо, и с тех пор HTTP[36][37] стал стандартом для передачи гипертекстовых документов в сети.

Одним из ключевых аспектов HTTP является его простота и универсальность. HTTP использует текстовый формат для передачи данных, что делает его легко читаемым и понятным как для машин, так и для людей. Запросы и ответы в HTTP формируются в виде строк, что упрощает процесс их анализа и отладки. Клиент (обычно браузер) отправляет запрос серверу, который обрабатывает его и возвращает ответ, содержащий запрашиваемые данные, статус выполнения запроса и метаинформацию.

HTTP — это протокол уровня приложения, который работает поверх других протоколов, таких как TCP/IP[38][39]. Это позволяет ему обеспечивать надежную и последовательную доставку данных, а также корректировать ошибки передачи. HTTP поддерживает широкий спектр методов, каждый из которых предназначен для выполнения определенных действий. Наиболее распространенные из них — GET, POST, PUT и DELETE. Метод GET используется для запроса данных с сервера, POST — для отправки данных на сервер, PUT — для обновления существующих данных, а DELETE — для удаления данных.

Преимущество HTTP в его гибкости и способности адаптироваться под различные нужды. Например, современные веб-приложения используют HTTP для взаимодействия с API, что позволяет обмениваться данными между различными системами и сервисами. HTTP также поддерживает работу с мультимедийным контентом, таким как изображения, видео и аудио, что делает его идеальным для создания разнообразных веб-приложений.

Безопасность передачи данных является важным аспектом HTTP. Для обеспечения конфиденциальности и целостности данных используется протокол HTTPS, который представляет собой HTTP, работающий поверх SSL/TLS. HTTPS обеспечивает шифрование данных, передаваемых между клиентом и сервером, что защищает их от перехвата и изменения злоумышленниками. В наши дни использование HTTPS стало стандартом для большинства веб-сайтов, особенно тех, которые обрабатывают конфиденциальную информацию, такую как личные данные или данные кредитных карт.

Кроме того, HTTP поддерживает кэширование, что позволяет сохранять копии веб-страниц и других ресурсов для ускорения их загрузки при повторных обращениях. Это существенно улучшает пользовательский опыт, так как сокращает время ожидания и уменьшает нагрузку на серверы. Также HTTP поддерживает возможность сжатия данных, что снижает объем передаваемых данных и ускоряет их доставку.

HTTP продолжает развиваться, предлагая новые возможности и улучшения, которые позволяют разрабатывать более эффективные, безопасные и производительные веб-приложения. Независимо от сложности и масштабов проекта, HTTP остается фундаментальным инструментом для передачи данных в интернете.

3.3 Проект данных программной системы

На основании анализа предметной области из технического задания была разработана база данных.

База данных включает в себя следующие таблицы:

1. Users - пользователи системы.
2. Cars - пользовательские автомобили.
3. Passengers - пассажиры забронировавшие поездку.
4. Travels - поездки созданные водителем.
5. DriverTravel - таблица связи, необходимая для привязки водителя и поездки.
6. Chats - пользовательские чаты.
7. Messages - сообщения в чате.
8. Comments - комментарии под профилем.

На рисунке 3.1 изображена ER-диаграмма базы данных.

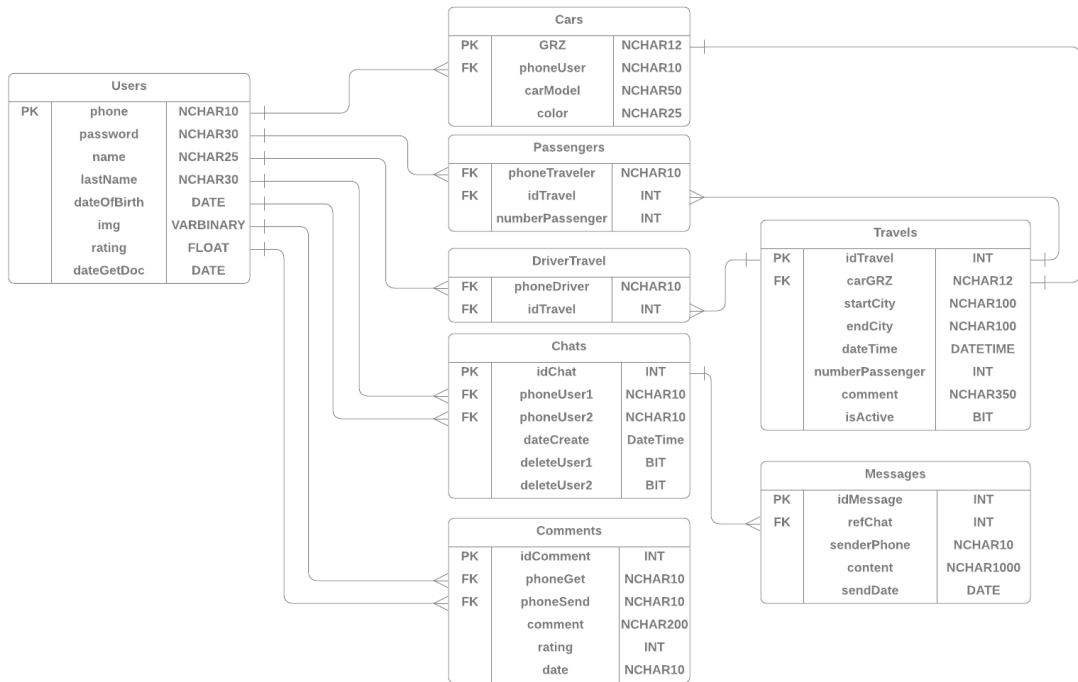


Рисунок 3.1 – ER-диаграмма базы данных

В таблицах 3.1-3.8 представлены атрибуты таблиц базы данных.

Таблица 3.1 – Атрибуты таблицы Users

Название атрибута	Тип данных	Описание
phone	NVARCHAR(10)	Номер телефона пользователя
password	NVARCHAR(30)	Пароль от учетной записи
name	NVARCHAR(25)	Имя пользователя
lastName	NVARCHAR(30)	Фамилия пользователя
dateOfBirth	DATE	Дата рождения
img	VARBINARY(MAX)	Фотография пользователя
rating	FLOAT	Рейтинг
dateGetDoc	DATE	Дата получения водительского удостоверения

Таблица 3.2 – Атрибуты таблицы Cars

Название атрибута	Тип данных	Описание
GRZ	NVARCHAR(12)	Государственный регистрационный знак автомобиля
phoneUser	NVARCHAR(10)	FK на номер телефона
carModel	NVARCHAR(50)	Модель автомобиля
color	NVARCHAR(25)	Цвет автомобиля

Таблица 3.3 – Атрибуты таблицы Passengers

Название атрибута	Тип данных	Описание
phoneTraveler	NVARCHAR(10)	FK на номер телефона попутчика
idTravel	INT	FK на уникальный идентификатор поездки
numberPassenger	INT	Количество пассажиров

Таблица 3.4 – Атрибуты таблицы Travels

Название атрибута	Тип данных	Описание
idTravel	INT	Уникальный идентификатор поездки
carGRZ	NVARCHAR(12)	FK на ГРЗ автомобиля
startCity	NVARCHAR(100)	Город начала маршрута
endCity	NVARCHAR(100)	Город окончания маршрута
dateTime	DATETIME	Дата и время поездки

Продолжение таблицы 3.4

Название атрибута	Тип данных	Описание
numberPassenger	INT	Количество пассажиров в поездке
comment	NVARCHAR(350)	Комментарий к поездке
isActive	BIT	Статус активности

Таблица 3.5 – Атрибуты таблицы DriverTravel

Название атрибута	Тип данных	Описание
phoneDriver	NVARCHAR(10)	FK на номер телефона водителя
idTravel	INT	FK на уникальный идентификатор поездки

Таблица 3.6 – Атрибуты таблицы Chats

Название атрибута	Тип данных	Описание
idChat	INT	Уникальный идентификатор чата
phoneUser1	NVARCHAR(10)	FK на номер телефона
phoneUser2	NVARCHAR(10)	FK на номер телефона
dateCreate	DateTime	Дата и время создания чата
deleteUser1	BIT	Статус удаленного чата у первого собеседника
deleteUser2	BIT	Статус удаленного чата у второго собеседника

Таблица 3.7 – Атрибуты таблицы Messages

Название атрибута	Тип данных	Описание
idMessage	INT	Уникальный идентификатор сообщения
refChat	INT	FK на чат
senderPhone	NVARCHAR(10)	Номер телефона отправителя
content	NVARCHAR(1000)	Текст сообщения
sendDate	DateTime	дата и время отправки

Таблица 3.8 – Атрибуты таблицы Comments

Название атрибута	Тип данных	Описание
idComment	INT	Уникальный идентификатор комментария
phoneGet	NVARCHAR(10)	Телефон получателя
phoneSend	NVARCHAR(10)	Телефон отправителя
comment	NVARCHAR(200)	Текст комментария
rating	INT	Оценка
date	DATE	Дата комментария

3.4 Проектирование архитектуры программной системы

3.4.1 Компоненты программной системы

Разрабатываемая система представляет из себя серверное приложение. Для разработки backend-модуля будет использоваться язык программирования C# 10.0[40] с применением фреймворка ASP.Net для создания серверного модуля и фреймворка SqlClient[39] для обеспечения доступа к базе данных.

Используемая среда разработки – Visual Studio 2022.

В качестве СУБД необходимо использовать Microsoft SQL Server.

В качестве системы управления версиями для командной разработки необходимо использовать Git[41][42][43].

Диаграмма развёртывания программы представлена на рисунке 3.2

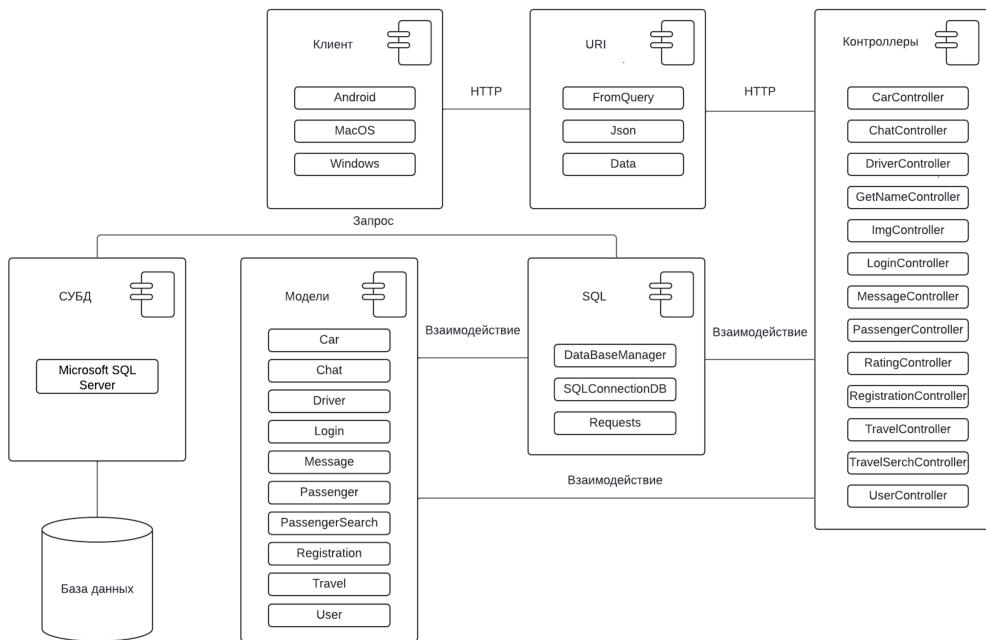


Рисунок 3.2 – Диаграмма развертывания программы

Программно-информационная система состоит из нескольких ключевых компонентов, каждый из которых выполняет свои уникальные функции, обеспечивая бесперебойную работу всей платформы. В центре этой системы находится клиентская часть, представленная приложениями для различных операционных систем, таких как Android, MacOS и Windows. Эти приложения позволяют пользователям взаимодействовать с системой через HTTP-запросы, обеспечивая доступ к необходимым функциям и данным.

Одним из важнейших компонентов системы является URI, который обрабатывает запросы, поступающие от клиентов. URI играет ключевую роль в маршрутизации и управлении данными, обрабатывая параметры запросов, работая с JSON-данными[44][45] и управляя передаваемыми данными. Благодаря этому компоненту, система может эффективно и безопасно обмениваться информацией между клиентами и серверами. URI, в свою очередь,

направляет запросы к соответствующим контроллерам, которые отвечают за выполнение конкретных задач.

Контроллеры представляют собой еще один важный элемент системы. Они отвечают за выполнение различных операций, связанных с функциональностью системы. Контроллеры управляют данными о транспортных средствах, чатами между пользователями, информацией о водителях, а также процессами аутентификации и регистрации пользователей. Они также занимаются управлением сообщениями, пассажирами, рейтингами, поездками и данными пользователей, обеспечивая координацию и выполнение всех основных функций системы. Каждый запрос, поступающий от клиента через URI, направляется к соответствующему контроллеру для обработки и выполнения необходимых операций.

Модели представляют собой набор компонентов, которые описывают основные сущности системы. Они включают в себя такие важные элементы, как транспортные средства, чаты, водители, пассажиры, поездки и пользователи. Эти модели играют ключевую роль в взаимодействии с базой данных, обеспечивая структурированный и организованный подход к хранению и обработке данных. Взаимодействие моделей с базой данных осуществляется через слой SQL, который включает в себя управление базой данных, установление и управление соединениями, а также обработку запросов к базе данных. Контроллеры взаимодействуют с моделями для выполнения операций чтения и записи данных, обеспечивая тем самым актуальность и консистентность данных в системе.

СУБД, используемая в системе, представлена Microsoft SQL Server. Эта система управления базами данных обеспечивает надежное и эффективное хранение всех данных, необходимых для работы платформы. Взаимодействие с базой данных осуществляется через SQL-компоненты, что позволяет обеспечить высокую производительность и надежность при работе с большими объемами данных. Модели обращаются к SQL для выполнения операций с данными, таких как запросы, обновления и удаления, гарантируя, что все данные, хранящиеся в базе данных, актуальны и доступны для обработки.

Таким образом, программно-информационная система представляет собой сложную и многоуровневую структуру, включающую в себя клиентскую часть, URI, контроллеры, модели, слой SQL и СУБД. Взаимодействие между этими компонентами осуществляется через четко определенные интерфейсы и протоколы, обеспечивая функциональность, надежность и безопасность всей платформы. Клиентские приложения отправляют запросы через URI, который маршрутизирует их к соответствующим контроллерам. Контроллеры, в свою очередь, взаимодействуют с моделями и SQL для выполнения необходимых операций с данными. Все данные хранятся в надежной СУБД, обеспечивая целостность и доступность информации. Взаимодействие между этими компонентами позволяет системе эффективно выполнять свои задачи и обеспечивать пользователям доступ к необходимым данным и функциям.

3.4.2 Архитектура программной системы

Архитектура MVC (Model-View-Controller) является широко распространенной парадигмой проектирования программного обеспечения, которая способствует разделению приложения на три взаимосвязанных компонента: модель, представление и контроллер. Эта архитектура позволяет улучшить управляемость, модульность и тестируемость кода.

Модель (Model)

Модель отвечает за управление данными и бизнес-логикой приложения. Она представляет собой центральную часть архитектуры, где происходят все операции с данными. Модель определяет структуру данных, правила их обработки и взаимодействие с базами данных или другими источниками данных. В модели реализуются основные функции, такие как получение, сохранение, обновление и удаление данных. Модель также уведомляет представление о необходимости обновления данных, чтобы обеспечить синхронизацию между состоянием данных и их отображением.

Представление (View)

Представление отвечает за отображение данных пользователю. Оно получает данные от модели и отображает их в удобной и понятной форме. Представление не содержит логики обработки данных; его задача — только визуализация информации. В веб-приложениях это могут быть HTML-страницы, шаблоны или компоненты пользовательского интерфейса. Важно отметить, что представление не взаимодействует напрямую с моделью, а получает данные через контроллер, что способствует снижению зависимости между компонентами.

Контроллер (Controller)

Контроллер служит посредником между моделью и представлением. Он обрабатывает входные данные от пользователя, вызывает соответствующие методы модели для обработки этих данных и обновляет представление в соответствии с результатами работы модели. Контроллеры принимают запросы от пользователей, определяют, какие действия должны быть выполнены, и управляют потоком данных между моделью и представлением. Благодаря этому контроллеры обеспечивают логическую связь между компонентами системы и позволяют изолировать бизнес-логику от пользовательского интерфейса.

Взаимодействие между компонентами:

- Модель и Представление. Модель обновляет данные и уведомляет представление о необходимости изменения отображения. Представление запрашивает данные у модели и обновляется в соответствии с изменениями.
- Модель и Контроллер. Контроллер взаимодействует с моделью для выполнения операций над данными. Он отправляет запросы модели на изменение данных и получает обновленные данные для последующей передачи представлению.
- Контроллер и Представление. Контроллер управляет отображением данных, передавая модели необходимые запросы и обновляя представление в соответствии с результатами работы модели. Контроллер обрабатывает пользовательские запросы и определяет, какие действия нужно предпринять, чтобы ответить на эти запросы.

Преимущества архитектуры MVC:

- Разделение обязанностей. Разделение кода на три компонента улучшает управляемость и упрощает сопровождение приложения.
- Модульность. Каждый компонент можно разрабатывать и тестировать независимо от других, что ускоряет процесс разработки.
- Повышенная testируемость. Благодаря четкому разделению обязанностей проще писать и выполнять автоматические тесты для каждого компонента.
- Масштабируемость. Архитектура MVC облегчает добавление новых функций и компонентов без значительного изменения существующего кода.
- Гибкость. Легче менять или обновлять отдельные компоненты системы, такие как пользовательский интерфейс, не затрагивая бизнес-логику.

Архитектура MVC, благодаря своему структурированному подходу и разделению обязанностей, помогает создавать более надежные, поддерживающие и масштабируемые приложения.

3.4.3 Архитектура программных классов моделей и контроллеров

IActionResult - это интерфейс, который представляет результат действия в ASP.NET Core. Контроллеры возвращают объекты, реализующие этот интерфейс, чтобы сообщить о результате выполнения запроса. Например, это может быть возвращение данных, сообщение об ошибке или перенаправление.

На рисунке 3.3 представлена UML-диаграмма классов программной системы, отражающих взаимодействие моделей и контроллеров.

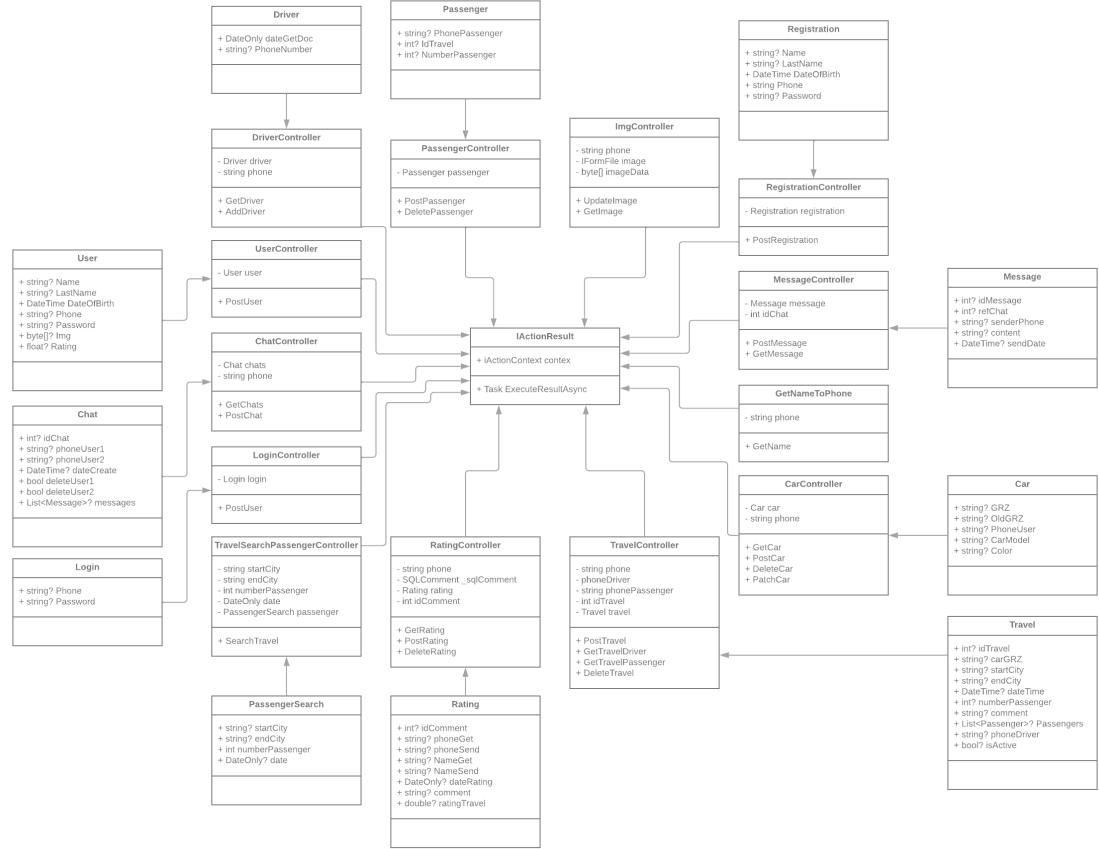


Рисунок 3.3 – Диаграмма классов моделей и контроллеров

Driver (Водитель) – хранит информацию о водителях.

Passenger (Пассажир) – содержит данные о пассажирах.

User (Пользователь) – включает основную информацию о пользователях системы.

Chat (Чат) – содержит информацию о чатах между пользователями.

Login (Логин) – хранит данные для входа в систему.

Registration (Регистрация) – содержит информацию для регистрации новых пользователей.

Message (Сообщение) – представляет сообщения в чатах.

Car (Автомобиль) – включает данные об автомобилях пользователей.

Travel (Поездка) – содержит информацию о поездках.

PassengerSearch (Поиск пассажиров) – используется для поиска пассажиров.

Rating (Рейтинг) – включает данные о рейтингах пользователей.

DriverController (Контроллер водителей) – управляет операциями, связанными с водителями.

PassengerController (Контроллер пассажиров) – обрабатывает операции, связанные с пассажирами.

UserController (Контроллер пользователей) – управляет действиями, связанными с пользователями.

ChatController (Контроллер чатов) – управляет взаимодействиями в чатах.

LoginController (Контроллер входа) – обрабатывает операции входа в систему.

RegistrationController (Контроллер регистрации) – управляет регистрацией новых пользователей.

MessageController (Контроллер сообщений) – обрабатывает операции, связанные с сообщениями в чатах.

GetNameToPhone (Контроллер получения имени по телефону) – предоставляет функциональность получения имени по номеру телефона.

CarController (Контроллер автомобилей) – управляет данными об автомобилях.

TravelController (Контроллер поездок) – обрабатывает операции, связанные с поездками.

TravelSearchPassengerController (Контроллер поиска пассажиров для поездок) – управляет поиском пассажиров для поездок.

RatingController (Контроллер рейтингов) – управляет данными о рейтингах.

3.4.4 Архитектура программных классов SQL запросов

SQLConnectionDb предоставляет стандартные параметры для подключения к базе данных. Этот класс отвечает за установку и хранение параметров подключения, таких как строка подключения и настройки, необходимые для подключения к базе данных.

`SqlConnection` представляет подключение к базе данных и управляет его состоянием. Это класс от Microsoft, который обеспечивает методы для открытия и закрытия соединения, а также выполнения команд SQL.

`DatabaseManager` предоставляет асинхронные методы для работы с базой данных. Этот класс управляет открытием и закрытием подключений в асинхронном режиме, что позволяет повысить производительность системы за счет эффективного использования ресурсов и предотвращения блокировок.

На рисунке 3.4 представлена UML-диаграмма классов программной системы, отражающих принцип работы SQL запросов.

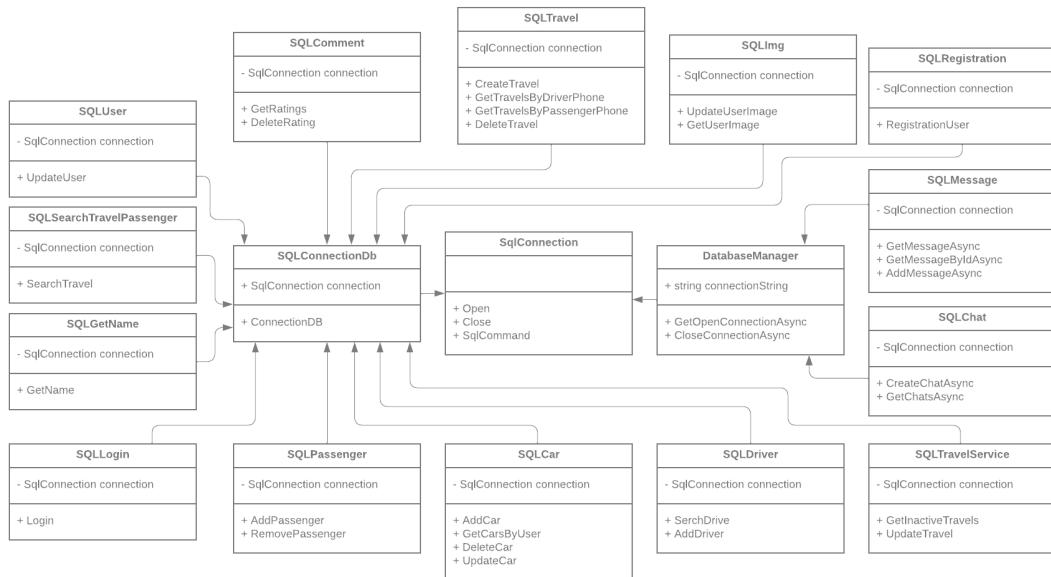


Рисунок 3.4 – Диаграмма классов SQL запросов

`SQLUser` позволяет обновлять информацию о пользователях.

`SQLSearchTravelPassenger` обеспечивает функциональность поиска пассажиров для поездок.

`SQLGetName` позволяет получать имя пользователя по номеру телефона.

`SQLLogin` управляет операциями входа в систему.

`SQLPassenger` позволяет добавлять и удалять пассажиров.

SQLCar управляет данными об автомобилях, включая добавление, получение, удаление и обновление информации о них.

SQLDriver обеспечивает поиск и добавление водителей.

SQLTravel управляет данными о поездках, включая создание, получение и удаление поездок.

SQLImg позволяет обновлять и получать изображения пользователей.

SQLRegistration управляет регистрацией новых пользователей.

SQLMessage обеспечивает функциональность для работы с сообщениями в чатах, включая их получение и добавление.

SQLChat управляет созданием чатов и получением списка чатов.

SQLTravelService обеспечивает дополнительные сервисы для работы с поездками, такие как получение неактивных поездок и обновление информации о поездках.

SQLComment управляет данными о рейтингах и комментариях, включая их получение и удаление.

4 Рабочий проект

4.1 Классы, используемые при разработке серверной части программной системы

В данном разделе описывается список классов и их методов, использованных при разработке серверной части программной системы.

4.1.1 Спецификация классов SQL запросов

Основные SQL-классы для подключения к базе данных включают `SQLConnectionDb` и `DatabaseManager`.

`SQLConnectionDb` предоставляет методы для управления соединением с базой данных, включая открытие, закрытие и проверку состояния соединения. Этот класс используется для синхронного взаимодействия с базой данных.

`DatabaseManager` - это асинхронная версия класса подключения к базе данных, которая обеспечивает возможность выполнения асинхронных запросов к базе данных для улучшения производительности и масштабируемости. Он предоставляет асинхронные версии методов для работы с соединением, что позволяет выполнять асинхронные операции базы данных без блокировки потока.

В таблицах 4.1 – 4.19 приведено описание полей и методов классов для работы с базой данных.

Таблица 4.1 – Спецификация полей класса «`SQLConnectionDb`»

Наименование	Метод доступа	Тип данных	Описание
1	2	3	4
<code>connectionString</code>	<code>private</code>	<code>string?</code>	Строка подключения к БД
<code>connection</code>	<code>protected</code>	<code>SqlConnection</code>	Состояние подключения к БД

Таблица 4.2 – Спецификация методов класса «SQLConnectionDb»

Наименование	Метод доступа	Описание
1	2	3
ConnectionDB	public	Устанавливает соединение с базой данных, используя заданную строку подключения. В случае успеха возвращает объект типа SqlConnection, представляющий активное соединение. В случае ошибки выводит сообщение об ошибке и возвращает null.
CloseDB	public	Закрывает текущее соединение с базой данных. Не принимает аргументов. Возвращает объект типа SqlConnection, представляющий закрытое соединение.

Таблица 4.3 – Спецификация полей класса «DatabaseManager»

Наименование	Метод доступа	Тип данных	Описание
1	2	3	4
connectionString	private static	string	Строка подключения к БД

Таблица 4.4 – Спецификация методов класса «DatabaseManager»

Наименование	Метод доступа	Описание
1	2	3
OpenConnection Async	public async	Асинхронно открывает соединение с базой данных, используя строку подключения. Возвращает объект типа SqlConnection, представляющий активное соединение.
CloseConnection Async	public async	Асинхронно закрывает указанное соединение с базой данных, если оно открыто. Не принимает аргументов.

Таблица 4.5 – Спецификация методов класса «SQLCar»

Наименование	Метод доступа	Описание
1	2	3
AddCar	public	Добавляет новую запись о машине в базу данных. Принимает объект типа Car в качестве аргумента. Возвращает логическое значение, указывающее на успешность операции добавления.
GetCarsByUser	public	Возвращает список машин, принадлежащих определенному пользователю. Принимает номер телефона пользователя в качестве аргумента. Возвращает список объектов типа Car.
DeleteCar	public	Удаляет запись о машине из базы данных. Принимает номер телефона пользователя и государственный регистрационный знак (ГРЗ) машины в качестве аргументов. Возвращает логическое значение, указывающее на успешность операции удаления.
UpdateCar	public	Обновляет информацию о машине в базе данных. Принимает объект типа Car, содержащий новую информацию о машине, а также старый ГРЗ машины в качестве аргументов. Возвращает логическое значение, указывающее на успешность операции обновления.

Таблица 4.6 – Спецификация методов класса «SQLChat»

Наименование	Метод доступа	Описание
1	2	3
CreateChatAsync	public	Создает чат между двумя пользователями. Если чат уже существует, возвращает информацию о нем. Принимает объект типа Chat в качестве аргумента. Возвращает объект типа Chat.

Продолжение таблицы 4.6

1	2	3
GetChatsAsync	public	Возвращает список всех чатов для определенного пользователя. Принимает номер телефона пользователя в качестве аргумента. Возвращает список объектов типа Chat.

Таблица 4.7 – Спецификация методов класса «SQLComment»

Наименование	Метод доступа	Описание
1	2	3
GetRatings	public	Возвращает список оценок для указанного пользователя. Принимает номер телефона пользователя в качестве аргумента. Возвращает список объектов типа Rating.
AddRating	public	Добавляет новую оценку в базу данных. Принимает объект типа Rating в качестве аргумента. Возвращает идентификатор новой оценки.
DeleteRating	public	Удаляет оценку из базы данных по указанному идентификатору. Принимает идентификатор оценки в качестве аргумента. Возвращает логическое значение, указывающее на успешность операции удаления.

Таблица 4.8 – Спецификация методов класса «SQLDriver»

Наименование	Метод доступа	Описание
1	2	3
SerchDrive	public	Возвращает дату получения водительских документов пользователя с указанным номером телефона. Принимает номер телефона пользователя в качестве аргумента. Возвращает объект типа DateOnly, представляющий дату получения документов. Если запись не найдена или значение поля равно NULL, возвращает null.
AddDriver	public	Обновляет дату получения водительских документов пользователя с указанным номером телефона. Принимает объект типа Driver в качестве аргумента. Возвращает логическое значение, указывающее на успешность операции обновления.

Таблица 4.9 – Спецификация методов класса «SQLGetName»

Наименование	Метод доступа	Описание
1	2	3
GetName	public	Получает имя пользователя по его номеру телефона. Принимает номер телефона пользователя в качестве аргумента. Возвращает строку с именем пользователя. Если пользователь с указанным номером телефона не найден, возвращает null.

Таблица 4.10 – Спецификация методов класса «SQLImg»

Наименование	Метод доступа	Описание
1	2	3
UpdateUserImage	public	Обновляет изображение пользователя в базе данных. Принимает номер телефона пользователя и массив байтов с изображением в качестве аргументов. Возвращает логическое значение, указывающее на успешность операции обновления.
GetUserImage	public	Получает изображение пользователя из базы данных. Принимает номер телефона пользователя в качестве аргумента. Возвращает массив байтов с изображением пользователя. Если изображение не найдено или возникает ошибка, возвращает null.

Таблица 4.11 – Спецификация методов класса «SQLLogin»

Наименование	Метод доступа	Описание
1	2	3
Login	public	Выполняет аутентификацию пользователя. Принимает объект Login, содержащий телефон и пароль пользователя. Возвращает объект User, если аутентификация успешна, или null, если неуспешна.

Таблица 4.12 – Спецификация полей класса «SQLMessage»

Наименование	Метод доступа	Тип данных	Описание
1	2	3	4
dbManager	private	Database Manager	Менеджер базы данных для управления подключениями и операциями с БД

Таблица 4.13 – Спецификация методов класса «SQLMessage»

Наименование	Метод доступа	Описание
1	2	3
GetMessageAsync	public	Асинхронно получает список сообщений для заданного чата. Принимает объект Chat, содержащий номера телефонов пользователей. Возвращает список объектов Message.
GetMessageById Async	public	Асинхронно получает список сообщений по идентификатору чата. Принимает идентификатор чата в качестве аргумента. Возвращает список объектов Message.
AddMessageAsync	public	Асинхронно добавляет новое сообщение в базу данных. Принимает объект Message, содержащий данные сообщения. Возвращает идентификатор добавленного сообщения.

Таблица 4.14 – Спецификация методов класса «SQLPassenger»

Наименование	Метод доступа	Описание
1	2	3
AddPassenger	public	Добавляет нового пассажира в базу данных. Принимает объект Passenger, содержащий данные пассажира. Возвращает результат добавления (успешно, уже существует, ошибка).
RemovePassenger	public	Удаляет пассажира из базы данных. Принимает объект Passenger, содержащий данные пассажира. Возвращает логическое значение, указывающее на успешность операции удаления.

Таблица 4.15 – Спецификация методов класса «SQLRegistration»

Наименование	Метод доступа	Описание
1	2	3
RegistrationUser	public	Регистрирует нового пользователя в базе данных. Принимает объект Registration, содержащий данные пользователя. Выполняет команду INSERT для добавления данных пользователя в таблицу Users.

Таблица 4.16 – Спецификация методов класса «SQLSearchTravelPassenger»

Наименование	Метод доступа	Описание
1	2	3
SearchTravel	public	Выполняет поиск поездок для пассажира. Принимает объект PassengerSearch, содержащий данные для поиска поездок (город отправления, город прибытия, дата и количество пассажиров). Возвращает список объектов Travel, соответствующих критериям поиска.

Таблица 4.17 – Спецификация методов класса «SQLTravel»

Наименование	Метод доступа	Описание
1	2	3
CreateTravel	public	Создаёт новую поездку в базе данных. Принимает объект Travel и возвращает идентификатор созданной поездки или null в случае ошибки.
GetTravelsBy DriverPhone	public	Возвращает список поездок, связанных с указанным номером телефона водителя. Принимает строку с номером телефона водителя и возвращает список объектов Travel.

Продолжение таблицы 4.17

1	2	3
GetTravelsBy PassengerPhone	public	Возвращает список поездок, связанных с указанным номером телефона пассажира. Принимает строку с номером телефона пассажира и возвращает список объектов Travel.
DeleteTravel	public	Удаляет поездку из базы данных по указанному идентификатору. Принимает идентификатор поездки и возвращает true, если удаление прошло успешно, или false в случае ошибки.

Таблица 4.18 – Спецификация методов класса «SQLTravelService»

Наименование	Метод доступа	Описание
1	2	3
GetInactiveTravels	public	Возвращает список неактивных поездок, которые завершились до текущей даты. Принимает текущую дату в качестве параметра и возвращает список объектов Travel.
UpdateTravel	public	Обновляет информацию о поездке в базе данных. Принимает объект Travel и изменяет его статус активности в базе данных.

Таблица 4.19 – Спецификация методов класса «SQLUser»

Наименование	Метод доступа	Описание
1	2	3
UpdateUser	public	Обновляет информацию о пользователе в базе данных. Принимает объект User и изменяет его данные (пароль, имя, фамилию и дату рождения) в базе данных по номеру телефона. Возвращает true, если операция успешна, и false в случае ошибки.

4.1.2 Описание моделей программной системы

Модели служат для описания данных, которые используются в приложении. Они представляют собой структуры данных, которые содержат информацию о различных объектах, таких как пользователи, поездки, сообщения и другие.

В таблицах 4.20 – 4.29 приведено описание полей моделей.

Таблица 4.20 – Спецификация полей класса «Car»

Наименование	Метод доступа	Тип данных	Описание
1	2	3	4
GRZ	public	string?	Государственный регистрационный знак автомобиля
OldGRZ	public	string?	Предыдущий государственный регистрационный знак автомобиля
PhoneUser	public	string?	Номер телефона пользователя, владеющего автомобилем
CarModel	public	string?	Модель автомобиля
Color	public	string?	Цвет автомобиля

Таблица 4.21 – Спецификация полей класса «Chat»

Наименование	Метод доступа	Тип данных	Описание
1	2	3	4
idChat	public	int?	Уникальный идентификатор чата
phoneUser1	public	string?	Номер телефона первого пользователя
phoneUser2	public	string?	Номер телефона второго пользователя
dateCreate	public	DateTime?	Дата создания чата

Продолжение таблицы 4.21

1	2	3	4
deleteUser1	public	bool	Показывает, удален ли чат у первого пользователя
deleteUser2	public	bool	Показывает, удален ли чат у второго пользователя
messages	public	List<Message>	Список сообщений в чате

Таблица 4.22 – Спецификация полей класса «Driver»

Наименование	Метод доступа	Тип данных	Описание
1	2	3	4
dateGetDoc	public	DateOnly	Дата получения водительского удостоверения
PhoneNumber	public	string?	Номер телефона водителя

Таблица 4.23 – Спецификация полей класса «Login»

Наименование	Метод доступа	Тип данных	Описание
1	2	3	4
Phone	public	string?	Номер телефона пользователя
Password	public	string?	Пароль пользователя

Таблица 4.24 – Спецификация полей класса «Message»

Наименование	Метод доступа	Тип данных	Описание
1	2	3	4
idMessage	public	int?	Идентификатор сообщения

Продолжение таблицы 4.24

1	2	3	4
refChat	public	int?	Идентификатор чата, к которому относится сообщение
senderPhone	public	string?	Номер телефона отправителя сообщения
content	public	string?	Содержимое сообщения
sendDate	public	DateTime?	Дата и время отправки сообщения

Таблица 4.25 – Спецификация полей класса «Passenger»

Наименование	Метод доступа	Тип данных	Описание
1	2	3	4
PhonePassenger	public	string?	Номер телефона пассажира
IdTravel	public	int?	Идентификатор поездки
NumberPassenger	public	int?	Количество пассажиров

Таблица 4.26 – Спецификация полей класса «PassengerSearch»

Наименование	Метод доступа	Тип данных	Описание
1	2	3	4
startCity	public	string?	Город отправления
endCity	public	string?	Город прибытия
numberPassenger	public	int	Количество пассажиров
date	public	DateOnly?	Дата отправления

Таблица 4.27 – Спецификация полей класса «Registration»

Наименование	Метод доступа	Тип данных	Описание
1	2	3	4
Name	public	string?	Имя пользователя
LastName	public	string?	Фамилия пользователя
DateOfBirth	public	DateTime	Дата рождения пользователя
Phone	public	string	Номер телефона пользователя
Password	public	string?	Пароль пользователя

Таблица 4.28 – Спецификация полей класса «Travel»

Наименование	Метод доступа	Тип данных	Описание
1	2	3	4
idTravel	public	int?	Идентификатор поездки
carGRZ	public	string?	Номерной знак автомобиля
startCity	public	string?	Город отправления
endCity	public	string?	Город назначения
dateTime	public	DateTime?	Время отправления
numberPassenger	public	int?	Количество пассажиров
comment	public	string?	Комментарий к поездке
Passengers	public	List<Passenger>?	Список пассажиров
phoneDriver	public	string?	Номер телефона водителя
isActive	public	bool?	Показатель активности поездки

Таблица 4.29 – Спецификация полей класса «User»

Наименование	Метод доступа	Тип данных	Описание
1	2	3	4
Name	public	string?	Имя пользователя
LastName	public	string?	Фамилия пользователя
DateOfBirth	public	DateTime	Дата рождения пользователя
Phone	public	string?	Номер телефона пользователя
Password	public	string?	Пароль пользователя
Img	public	byte[]?	Фотография пользователя в виде массива байтов
Rating	public	float?	Рейтинг пользователя

4.1.3 Описание контроллеров

Контроллеры в нашем приложении действуют как посредники между фронтендом и базой данных, обрабатывая запросы, поступающие от клиента, и предоставляя соответствующие данные. Некоторые из наших контроллеров, такие как ChatAPIController, TravelAPIController, и MessageAPIController, работают асинхронно, что означает, что они могут обрабатывать несколько запросов параллельно, что повышает производительность системы и улучшает отзывчивость интерфейса для пользователей. Это особенно важно в случае обработки большого количества данных или при взаимодействии с внешними сервисами.

В то время как другие контроллеры, такие как UserAPIController и RegistrationAPIController, выполняются процедурно, обрабатывая запросы последовательно и возвращая результаты по мере готовности. Это позволяет им обеспечивать надежное взаимодействие с базой данных и выполнение операций в строгом порядке, что особенно важно для операций, которые требуют транзакционности и целостности данных.

В таблицах 4.30 – 4.42 приведено описание методов контроллеров.

Таблица 4.30 – Спецификация методов класса «CarController»

Наименование	Метод доступа	Описание
1	2	3
GetCar	[HttpGet]	Получает список машин для указанного пользователя. Принимает номер телефона пользователя в качестве параметра запроса. Возвращает список объектов Car или код 404, если машины не найдены.
PostCar	[HttpPost]	Добавляет новую машину в базу данных. Принимает объект Car в теле запроса. Возвращает добавленный объект Car или код 400 в случае ошибки.
DeleteCar	[HttpDelete]	Удаляет машину из базы данных. Принимает объект Car с указанием телефона пользователя и номера машины в теле запроса. Возвращает сообщение об успешном удалении или код 404 в случае ошибки.
PatchCar	[HttpPatch]	Обновляет информацию о машине в базе данных. Принимает объект Car с обновленными данными в теле запроса. Возвращает сообщение об успешном обновлении или код 400 в случае ошибки.

Таблица 4.31 – Спецификация методов класса «ChatAPIController»

Наименование	Метод доступа	Описание
1	2	3
GetChats	[HttpGet]	Получает список чатов для указанного пользователя. Принимает номер телефона пользователя в качестве параметра запроса. Возвращает список объектов Chat или код 400, если чаты не найдены.

Продолжение таблицы 4.31

1	2	3
PostChat	[HttpPost]	Создает новый чат в базе данных. Принимает объект Chat в теле запроса. Возвращает созданный объект Chat или код 400 в случае ошибки.

Таблица 4.32 – Спецификация методов класса «DriverController»

Наименование	Метод доступа	Описание
1	2	3
GetDriver	[HttpGet]	Получает информацию о водителе по указанному номеру телефона. Принимает номер телефона в качестве параметра запроса. Возвращает объект Driver или код 404, если водитель не найден.
AddDriver	[HttpPost]	Добавляет нового водителя в базу данных. Принимает объект Driver в теле запроса. Возвращает созданный объект Driver или код 404 в случае ошибки.

Таблица 4.33 – Спецификация методов класса «GetNameToPhone»

Наименование	Метод доступа	Описание
1	2	3
GetName	[HttpGet]	Получает имя пользователя по указанному номеру телефона. Принимает номер телефона в качестве параметра запроса. Возвращает имя пользователя или код 404, если пользователь не найден.

Таблица 4.34 – Спецификация методов класса «ImgAPIController»

Наименование	Метод доступа	Описание
1	2	3
UpdateImage	[HttpPatch]	Обновляет изображение пользователя по указанному номеру телефона. Принимает номер телефона и файл изображения в формате multipart/form-data. Возвращает код состояния 200 (OK) в случае успешного обновления, или код 400 (BadRequest), если файл изображения отсутствует.
GetImage	[HttpGet]	Получает изображение пользователя по указанному номеру телефона. Принимает номер телефона в качестве параметра пути. Возвращает изображение в формате jpeg или код 404 (NotFound), если изображение не найдено.

Таблица 4.35 – Спецификация методов класса «LoginAPIController»

Наименование	Метод доступа	Описание
1	2	3
PostUser	[HttpPost]	Аутентификация пользователя. Принимает объект Login, содержащий номер телефона и пароль пользователя. Возвращает результат аутентификации: true, если пользователь успешно аутентифицирован, false в противном случае.

Таблица 4.36 – Спецификация методов класса «MessageAPIController»

Наименование	Метод доступа	Описание
1	2	3
PostMessage	[HttpPost]	Отправка сообщения. Принимает объект Message, содержащий информацию о сообщении (id чата, отправитель, содержание и дата отправки). Возвращает идентификатор добавленного сообщения.
GetMessage	[HttpGet]	Получение сообщений по id чата. Принимает id чата. Возвращает список сообщений, относящихся к указанному чату.

Таблица 4.37 – Спецификация методов класса «PassengerAPIController»

Наименование	Метод доступа	Описание
1	2	3
PostPassenger	[HttpPost]	Добавление пассажира к поездке. Принимает объект Passenger, содержащий информацию о пассажире (телефон пассажира, идентификатор поездки и количество пассажиров). Возвращает соответствующий HTTP-ответ в зависимости от результата операции (пассажир добавлен, пассажир уже существует в поездке, ошибка).
DeletePassenger	[HttpDelete]	Удаление пассажира из поездки. Принимает объект Passenger, содержащий информацию о пассажире (телефон пассажира и идентификатор поездки). Возвращает соответствующий HTTP-ответ в зависимости от результата операции (пассажир удален из поездки, пассажир не найден в поездке или произошла ошибка).

Таблица 4.38 – Спецификация методов класса «RatingAPIController»

Наименование	Метод доступа	Описание
1	2	3
GetRating	[HttpGet]	Получение рейтинга пользователя. Принимает телефонный номер пользователя. Возвращает список объектов Rating, содержащих информацию о рейтинге данного пользователя. Если рейтинг не найден или не существует, возвращает соответствующий HTTP-ответ.
PostRating	[HttpPost]	Добавление рейтинга. Принимает объект Rating, содержащий информацию о рейтинге (телефон отправителя, телефон получателя и текст комментария). Возвращает идентификатор добавленного комментария или соответствующий HTTP-ответ в случае ошибки.
DeleteRating	[HttpDelete]	Удаление рейтинга. Принимает идентификатор комментария. Возвращает соответствующий HTTP-ответ в зависимости от результата операции (рейтинг успешно удален, рейтинг не найден или произошла ошибка).

Таблица 4.39 – Спецификация методов класса «RegistrationAPIController»

Наименование	Метод доступа	Описание
1	2	3
PostRegistration	[HttpPost]	Регистрация пользователя. Принимает объект Registration, содержащий информацию о новом пользователе (имя, фамилия, дата рождения, телефон и пароль). Выполняет регистрацию пользователя в базе данных. Возвращает соответствующий HTTP-ответ.

Таблица 4.40 – Спецификация методов класса «TravelAPIController»

Наименование	Метод доступа	Описание
1	2	3
PostTravel	[HttpPost]	Создание новой поездки. Принимает объект Travel, содержащий информацию о новой поездке. Выполняет создание новой поездки в базе данных. Возвращает идентификатор созданной поездки или код ошибки BadRequest.
GetTravelDriver	[HttpGet]	Получение списка поездок водителя. Принимает телефон водителя. Возвращает список поездок, связанных с указанным телефоном водителя, или соответствующий HTTP-ответ.
GetTravelPassenger	[HttpGet ("passenger")]	Получение списка поездок пассажира. Принимает телефон пассажира. Возвращает список поездок, связанных с указанным телефоном пассажира, или соответствующий HTTP-ответ.
DeleteTravel	[HttpDelete]	Удаление поездки. Принимает идентификатор поездки. Выполняет удаление поездки из базы данных. Возвращает соответствующий HTTP-ответ.

Таблица 4.41 – Спецификация методов класса «TravelSearchPassengerAPI»

Наименование	Метод доступа	Описание
1	2	3
SearchTravel	[HttpGet]	Поиск поездок для пассажира. Принимает параметры для поиска: startCity (начальный город), endCity (конечный город), numberPassenger (количество пассажиров), date (дата отправления). Выполняет поиск поездок, соответствующих указанным параметрам. Возвращает список найденных поездок или соответствующий HTTP-ответ[46]. В случае ошибки возвращает код состояния 500[47] с сообщением об ошибке.

Таблица 4.42 – Спецификация методов класса «UserAPIController»

Наименование	Метод доступа	Описание
1	2	3
PostUser	[HttpPatch]	Обновление информации о пользователе. Принимает объект типа User и обновляет информацию о пользователе в базе данных. Возвращает результат операции: true в случае успешного обновления и false в случае ошибки. В случае ошибки возвращает HTTP-ответ со статусом 400 (BadRequest).

4.2 Тестирование серверной части программной системы

4.2.1 Инструмент тестирования Postman

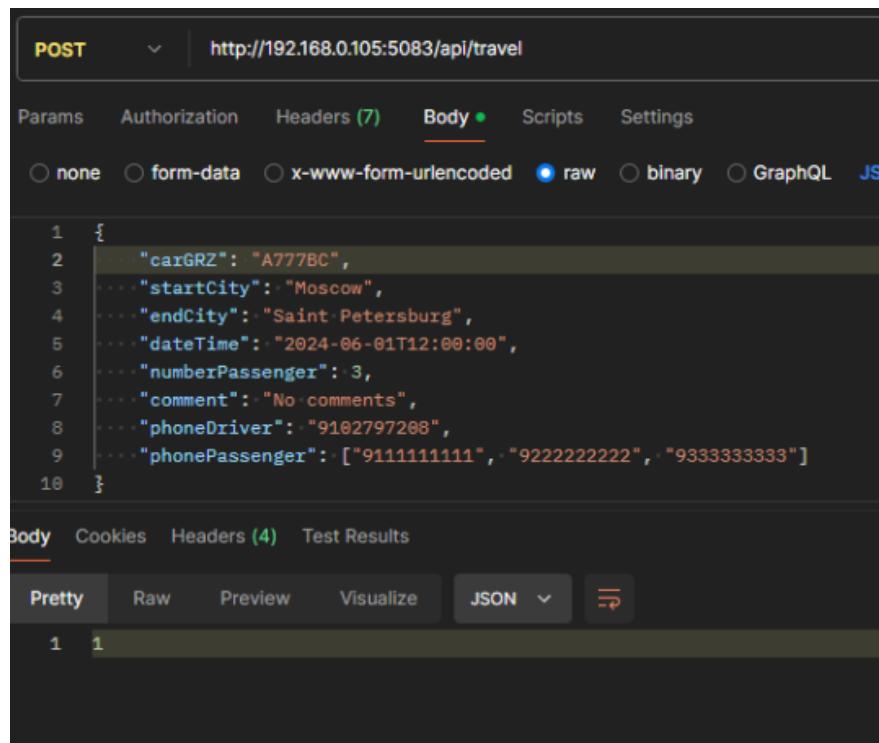
Postman предоставляет удобный интерфейс для отправки HTTP запросов различных типов, таких как GET, POST, PUT, DELETE и других. Этот инструмент позволяет легко настраивать URI, заголовки, параметры запроса и тело запроса. Он также предоставляет возможность работы с различ-

ными форматами данных, такими как JSON, XML[48][49], HTML и другими. Благодаря функциональности автоматической генерации кода, Postman значительно упрощает процесс тестирования API и взаимодействия с ними. Важно отметить, что Postman поддерживает средства автоматизации тестирования, что позволяет интегрировать его в CI/CD[50] процессы для обеспечения непрерывного контроля качества приложений.

4.2.2 Тестирование URI запросов

Тестирование было проведено с использованием инструмента Postman. В рамках тестирования были проверены основные запросы, и результаты показали, что все запросы были обработаны корректно. Каждый запрос включал в себя следующий цикл действий: запрос данных из базы данных, выполнение соответствующего действия в базе данных и получение ответа от сервера.

На рисунках 4.1 – 4.11 представлены результаты тестирования.



```
POST http://192.168.0.105:5083/api/travel

Params Authorization Headers (7) Body Scripts Settings
 none  form-data  x-www-form-urlencoded  raw  binary  GraphQL  JS

1 {
2     "carGRZ": "A777BC",
3     "startCity": "Moscow",
4     "endCity": "Saint Petersburg",
5     "dateTime": "2024-06-01T12:00:00",
6     "numberPassenger": 3,
7     "comment": "No comments",
8     "phoneDriver": "9102797208",
9     "phonePassenger": ["9111111111", "9222222222", "9333333333"]
10 }

Body Cookies Headers (4) Test Results
Pretty Raw Preview Visualize JSON 
1 1
```

Рисунок 4.1 – Тестирование запроса на создание поездки

POST | http://192.168.0.106:5083/api/login

Params | Authorization | Headers (9) | **Body** | Scripts | Settings

none | form-data | x-www-form-urlencoded | raw | binary | Graph

```

1  {
2    "Phone": "9102797208",
3    "Password": "pass"
4  }
5

```

Body | Cookies | Headers (4) | Test Results

Pretty | Raw | Preview | Visualize | **JSON** |

```

1  {
2    "name": "Максим",
3    "lastName": "Шеховцов",
4    "dateOfBirth": "1999-08-26T00:00:00",
5    "phone": "9102797208",
6    "password": "pass",
7    "img": "/9j/4AAQSkZJRgABAQAAAQABAAAD/2wBDAkGBwgHBgkIBwgKCg
     2wBDAQoKCg0MDRoP0xo3JR8lNzc3Nzc3Nzc3Nzc3Nzc3Nzc3Nz
     EABkBAQEBAQEAAAAAAAAAAAAAAAgMEBf/aAAwDQACEAMQAAAB7
     +OWEUFsosFBS1AUAAAACAAIAVAlgAgJYACFzwzEsABSUFgoAAAFg

```

Рисунок 4.2 – Тестирование запроса на вход в систему

GET | http://192.168.0.105:5083/api/car?phone=9102797208

Params | Authorization | Headers (9) | **Body** | Scripts | Settings

Key | Value

phone | 9102797208

Key | Value

Body | Cookies | Headers (4) | Test Results

Pretty | Raw | Preview | Visualize | **JSON** |

```

1  [
2    {
3      "grz": "A123BC",
4      "phoneUser": "9102797208",
5      "carModel": "Toyota Corolla",
6      "color": "Red"
7    },
8    {
9      "grz": "A133BC",
10     "phoneUser": "9102797208",
11     "carModel": "Toyota Corolla",
12     "color": "Red"
13   }
14 ]

```

Рисунок 4.3 – Тестирование запроса на получение автомобилей

HTTP http://192.168.0.105:5083/api/driver?phone=9102797208

DELETE http://192.168.0.105:5083/api/car

Params Authorization Headers (9) Body Scripts Settings

Body raw binary

```
1 {  
2   "GRZ": "A123BC",  
3   "PhoneUser": "9102797208"  
4 }  
5
```

Body Cookies Headers (4) Test Results

Pretty Raw Preview Visualize JSON

```
1 {  
2   "message": "Car successfully deleted"  
3 }
```

Рисунок 4.4 – Тестирование запроса на удаление автомобиля

PATCH http://192.168.0.105:5083/api/car

Params Authorization Headers (9) Body Scripts Settings

Body raw binary GraphQL JSON

```
1 {  
2   "GRZ": "A777BC",  
3   "OldGRZ": "A777BC",  
4   "PhoneUser": "9102797208",  
5   "CarModel": "Honda Civic",  
6   "Color": "Blue"  
7 }  
8
```

Body Cookies Headers (4) Test Results

Pretty Raw Preview Visualize JSON

```
1 {  
2   "message": "Car update"  
3 }
```

Рисунок 4.5 – Тестирование запроса на изменение автомобиля

GET http://192.168.0.105:5083/api/Travel?phoneDriver=9102797208

Params • Authorization Headers (9) Body • Scripts Settings

Query Params

	Key	Value
<input checked="" type="checkbox"/>	phoneDriver	910279
	Key	Value

Body Cookies Headers (4) Test Results

Pretty Raw Preview Visualize JSON

```
1 [  
2 {  
3     "idTravel": 5,  
4     "carGRZ": "A777BC",  
5     "startCity": "Moscow",  
6     "endCity": "Saint Petersburg",  
7     "dateTime": "2024-06-01T12:00:00",  
8     "numberPassenger": 100,  
9     "comment": "No comments",  
10    "passengers": [  
11        {  
12            "phonePassenger": "9102797208",  
13            "idTravel": null,  
14            "numberPassenger": 3  
15        },  
16        {  
17            "phonePassenger": "123",  
18            "idTravel": null,  
19            "numberPassenger": 3  
20        }  
21    ],  
22    "phoneDriver": null,  
23    "isActive": true  
24 }]  
25 ]
```

Рисунок 4.6 – Тестирование запроса на получение водительских поездок

DELETE <http://192.168.0.105:5083/api/Travel?IdTravel=1>

Params • Authorization Headers (9) Body • Scripts Settings

Query Params

<input checked="" type="checkbox"/>	Key
<input checked="" type="checkbox"/>	idTravel
	Key

Body Cookies Headers (4) Test Results

Pretty Raw Preview Visualize JSON ↻

```
1 "message": "Travel deleted successfully"
2
3
```

Рисунок 4.7 – Тестирование запроса на удаление поездки

GET http://192.168.0.105:5083/api/Travel/passenger?phonePassenger=9102797208

Params • Authorization Headers (9) Body • Scripts Settings

Query Params

Key	Value
phonePassenger	9102797208
Key	Value

Body Cookies Headers (4) Test Results

Pretty Raw Preview Visualize JSON

```
1 [  
2 {  
3     "idTravel": 6,  
4     "cargRZ": "A777BC",  
5     "startCity": "Moscow",  
6     "endCity": "Saint Petersburg",  
7     "dateTime": "2024-06-01T12:00:00",  
8     "numberPassenger": 100,  
9     "comment": "No comments",  
10    "passengers": [  
11        {  
12            "phonePassenger": "9102797208",  
13            "idTravel": null,  
14            "numberPassenger": 3  
15        },  
16        {  
17            "phonePassenger": "123",  
18            "idTravel": null,  
19            "numberPassenger": 3  
20        }  
21    ],  
22    "phoneDriver": "9102797208",  
23    "isActive": true  
24 }  
25 ]
```

Рисунок 4.8 – Тестирование запроса получение пассажирских поездок

POST | http://192.168.0.105:5083/api/chat

Params Authorization Headers (9) Body Scripts Set

none form-data x-www-form-urlencoded raw binary

```
1  {
2      "phoneUser1": "123",
3      "phoneUser2": "9182797288"
4 }
```

Body Cookies Headers (4) Test Results

Pretty Raw Preview Visualize JSON ↗

```
1  {
2      "idChat": 1,
3      "phoneUser1": "123",
4      "phoneUser2": "9182797288",
5      "dateCreate": "2024-05-29T01:36:06.68",
6      "deleteUser1": true,
7      "deleteUser2": false,
8      "messages": [
9          {
10              "idMessage": 7,
11              "refChat": 1,
12              "senderPhone": "123",
13              "content": "?",
14              "sendDate": "2024-05-29T19:10:41.5"
15          },
16          {
17              "idMessage": 6,
18              "refChat": 1,
19              "senderPhone": "123",
20              "content": "И что с того?",
21              "sendDate": "2024-05-29T19:09:03.767"
22      ]}
```

Рисунок 4.9 – Тестирование запроса на получение пользовательских чатов

GET http://192.168.0.105:5083/api/message?idChat=1

Params • Authorization Headers (9) Body • Scripts Settings

Query Params

<input checked="" type="checkbox"/>	Key
<input checked="" type="checkbox"/>	idChat
	Key

Body Cookies Headers (4) Test Results

Pretty Raw Preview Visualize JSON

```
1 [  
2 {  
3     "idMessage": 1082,  
4     "refChat": 1,  
5     "senderPhone": "123",  
6     "content": "?",  
7     "sendDate": "2024-05-29T21:31:59.893"  
8 },  
9 {  
10    "idMessage": 7,  
11    "refChat": 1,  
12    "senderPhone": "123",  
13    "content": "?",  
14    "sendDate": "2024-05-29T19:10:41.5"  
15 },  
16 {  
17    "idMessage": 6,  
18    "refChat": 1,  
19    "senderPhone": "123",  
20    "content": "И что с того?",  
21    "sendDate": "2024-05-29T19:09:03.767"  
22 }]
```

Рисунок 4.10 – Тестирование запроса на получение сообщений по чату

The screenshot shows a POST request in the Postman application. The URL is `http://192.168.0.105:5083/api/rating`. The **Body** tab is selected, showing a JSON payload:

```
1 {  
2   "phoneGet": "9102797208",  
3   "phoneSend": "123",  
4   "comment": "Отличная поездка!",  
5   "ratingTravel": 5.0  
6 }
```

The **Body** tab also includes tabs for **Pretty**, **Raw**, **Preview**, **Visualize**, and **JSON**.

Рисунок 4.11 – Тестирование запроса на создание комментария

ЗАКЛЮЧЕНИЕ

Разработка серверной части кроссплатформенного приложения для поиска попутчиков является актуальной и перспективной задачей. Современный рынок показывает растущий интерес к приложениям для совместных поездок, что обусловлено экономическими и экологическими факторами, а также удобством использования данных сервисов. В процессе работы были изучены потребности и ожидания пользователей, конкурентная ситуация и особенности технологической реализации серверной части приложения.

Основные результаты работы:

1. Проведен анализ предметной области, что позволило выявить ключевые требования и ожидания пользователей от подобных приложений, а также определить основные конкурентные решения и их функциональные особенности.
2. Разработана концептуальная модель программно-информационной системы, включающая варианты использования системы и требования к серверной части.
3. Спроектирована и реализована серверная часть программной системы. Разработана архитектура системы с описанием основных компонентов и их взаимодействия, а также ключевых классов и методов.
4. Проведено тестирование разработанной серверной части программной системы. Выполнено функциональное и системное тестирование для обеспечения корректной работы всех компонентов системы.

Все требования, объявленные в техническом задании, были полностью реализованы. Все задачи, поставленные в начале разработки проекта, были также решены.

Предполагается, что разрабатываемая программа система будет использоваться широким кругом лиц, заинтересованных в совместных поездках, включая тех, кто ищет экономичные и экологически чистые способы передвижения.

Перспективы дальнейшей разработки программной системы включают расширение функционала приложения, увеличение числа пользователей, улучшение алгоритмов поиска и предложения попутчиков, а также интеграцию новых сервисов, таких как динамическое ценообразование и дополнительные средства безопасности для пользователей.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Фленов, М.Е. Библия C#. 5-е издание. / М.Е. Фленов. – СПб. : БХВ, 2022. – 464 с. – ISBN 978-5-9775-6827-2. – Текст: непосредственный.
2. Троелсен, Э. Язык программирования C# 9 и платформа .NET 5: основные принципы и практики программирования, 10-е издание. / Э. Троелсен, Ф. Джепикс. – Москва: Диалектика, 2022. – 1392 с. – ISBN 978-5-907458-67-3. – Текст: непосредственный.
3. Умрихин, Е.Д. Разработка веб-приложений с помощью ASP.Net Core MVC. / Е.Д. Умрихин. – СПб. : БХВ, 2023. – 416 с. – ISBN 978-5-9775-1206-0. – Текст: непосредственный.
4. Лок, Э. ASP.NET CORE в действии / Э. Лок. – Москва: ДМК Пресс, 2021. – 906 с. – ISBN 978-5-97060-550-9. – Текст: непосредственный.
5. Смит, Д. Entity Framework Core в действии / Д. Смит. – Москва: ДМК Пресс, 2023. – 690 с. – ISBN 978-5-93700-114-6. – Текст: непосредственный.
6. Фримен, А. Entity Framework Core 2 для ASP.NET Core MVC для профессионалов / А. Фримен. – Москва: Диалектика, 2019. – 624 с. – ISBN 978-5-907114-86-9. – Текст: непосредственный.
7. Шилдс, У. SQL: быстрое погружение / У. Шилдс. – СПб. : Питер, 2022. – 224 с. – ISBN 978-5-4461-1835-9. – Текст: непосредственный.
8. Болье, А. Изучаем SQL. Генерация, выборка и обработка данных / А. Болье. – Москва: Диалектика-Вильямс, 2021. – 400 с. – ISBN 978-5-907365-54-4. – Текст: непосредственный.
9. Осипов, Д.Л. Технологии проектирования баз данных / Д.Л. Осипов. – М.: ДМК Пресс, 2019. – 498 с. – ISBN 978-5-97060-737-4. – Текст: непосредственный.
10. Уотсон, Б. Высокопроизводительный код на платформе .NET. 2-е издание / Б. Уотсон. – СПб: Питер, 2020. – 416 с. – ISBN 978-5-4461-0911-1. – Текст: непосредственный.

11. Литвиненко, Н.А. Программирование на C# для платформы .NET Core 3. Курс лекций / Н.А. Литвиненко. – Москва: Горячая Линия - Телеком, 2021. – 328 с. – ISBN 978-5-9912-0874-1. – Текст: непосредственный.
12. Черткова, Е.А. Программная инженерия. Визуальное моделирование программных систем / Е.А. Черткова. – Москва: ЮРАЙТ, 2022. – 148 с. – ISBN 978-5-534-09823-5. – Текст: непосредственный.
13. Хориков, В. Принципы юнит-тестирования / В. Хориков. – СПб. : Питер, 2022. – 320 с. – ISBN 978-5-4461-1683-6. – Текст: непосредственный.
14. Маурисио, А. Эффективное тестирование программного обеспечения / А. Маурисио. – Москва: ДМК Пресс, 2023. – 370 с. – ISBN 978-5-97060-997-2. – Текст: непосредственный.
15. Игнатьев, А. В. Тестирование программного обеспечения / А. В. Игнатьев. – Москва: Лань, 2021. – 56 с. – ISBN 978-5-8114-8072-2. – Текст: непосредственный.
16. Плаксин, М. А. Тестирование и отладка программ для профессионалов будущих и настоящих / М. А. Плаксин. – Москва: БИНОМ, 2020. – 170 с. – ISBN 978-5-00101-810-0. – Текст: непосредственный.
17. Ганди, Р. Head First. Git / Р. Ганди. – СПб. : БХВ, 2023. – 464 с. – ISBN 978-5-9775-1777-5. – Текст: непосредственный.
18. Чакон, С. Git для профессионального программиста / С. Чакон. – Санкт-Петербург: Питер, 2016. – 496 с. – ISBN 978-5-496-01763-3. – Текст: непосредственный.
19. Фримен, А. ASP.NET MVC 5 с примерами на C# 5.0 для профессионалов / А. Фримен. – Москва: Вильямс, 2018. – 736 с. – ISBN: 978-5-8459-1911-3. – Текст: непосредственный.
20. Дронов, В.А. JavaScript. 20 уроков для начинающих. / В.А. Дронов. – СПб. : БХВ, 2020. – 352 с. – ISBN 978-5-9775-6589-9. – Текст: непосредственный.
21. Дронов, В.А. JavaScript. Дополнительные уроки для начинающих. / В.А. Дронов. – СПб. : БХВ, 2021. – 352 с. – ISBN 978-5-9775-6781-7. – Текст: непосредственный.

22. Джепикс, Ф. Язык программирования C# 7 и платформы .NET и .NET Core / Ф. Джепикс, Э. Троелсен. – Москва: Вильямс, 2018. – 1328 с. – ISBN: 978-1-4842-3017-6. – Текст: непосредственный.
23. Роберт, А. UML для простых смертных / А. Роберт. – Москва: Манн, Иванов и Фербер, 2014. – 272 с. – ISBN 978-5-85582-367-7. – Текст: непосредственный.
24. Пайлон, Д. UML 2 для программистов / Д. Пайлон. – СПб: Питер, 2012. – 240 с. – ISBN 978-5-459-01684-0. – Текст: непосредственный.
25. Ларман, К. Применение UML и шаблонов проектирования. Введение в объектно-ориентированный анализ, проектирование и итеративную разработку: учебник и практикум для бакалавриата и магистратуры / К. Ларман. – М.: ООО “И.Д. Вильямс”, 2013. – 426 с. – ISBN 978-5-8459-1185-8.. – Текст: непосредственный.
26. Гаско, Р. Объектно Ориентированное Программирование / Р. Гаско. – Москва: Солон-Пресс, 2021. – 298 с. – ISBN 978-5-91359-285-9. – Текст: непосредственный.
27. Вайсфельд, М. Объектно-ориентированное мышление / М. Вайсфельд. – СПб: Питер, 2014. – 304 с. – ISBN 978-5-496-00793-1. – Текст: непосредственный.
28. Джонсон, Р. Приемы объектно-ориентированного проектирования. Паттерны проектирования / Р. Джонсон, Г. Эрих, Р. Хелм, Д. Влисседес. – Санкт-Петербург: Питер, 2016. – 366 с. – ISBN 978-5-459-01720-5. – Текст: непосредственный.
29. Коннолли, Т. Базы данных. Проектирование, реализация и сопровождение / Т. Коннолли, К. Бегг. – Москва: Логосфера, 2021. – 1448 с. – ISBN 978-5-94774-185-0. – Текст: непосредственный.
30. Маккарт, Дж. Чистый код. Создание, анализ и рефакторинг / Дж. Маккарт. – Москва: Диалектика, 2020. – 464 с. – ISBN 978-5-907114-85-2. – Текст: непосредственный.

31. Бло, Дж. Чистая архитектура. Искусство разработки программного обеспечения / Дж. Бло. – Москва: Питер, 2019. – 432 с. – ISBN 978-5-496-02482-2. – Текст: непосредственный.
32. Сандерс, Г. Паттерны интеграции корпоративных приложений / Г. Сандерс. – Москва: Бином, 2020. – 368 с. – ISBN 978-5-001-01304-0. – Текст: непосредственный.
33. Смит, Д. Enterprise Integration Patterns / Д. Смит. – Москва: ДМК Пресс, 2018. – 528 с. – ISBN 978-5-94074-808-4. – Текст: непосредственный.
34. Джордж, Р. Архитектура корпоративных приложений. Паттерны проектирования / Р. Джордж. – Москва: Логосфера, 2020. – 488 с. – ISBN 978-5-94774-204-8. – Текст: непосредственный.
35. Голов, А. Микросервисы. Паттерны проектирования / А. Голов. – СПб. : БХВ, 2023. – 544 с. – ISBN 978-5-9775-1347-0. – Текст: непосредственный.
36. Рихтер, Д. CLR via C#. Программирование на платформе Microsoft .NET Framework 4.5 на языке C#. / Д. Рихтер. – М.: Вильямс, 2016. – 896 с. – ISBN 978-5-8459-1913-7. – Текст: непосредственный.
37. Джонсон, Р. Spring в действии / Р. Джонсон, Д. Хоулер, М. Лайт, Т. Фишер. – М.: Вильямс, 2019. – 688 с. – ISBN 978-5-8459-1917-5. – Текст: непосредственный.
38. Шапошников, К. Программирование на языке C#. Полное руководство / К. Шапошников. – СПб.: БХВ-Петербург, 2020. – 768 с. – ISBN 978-5-9775-6906-4. – Текст: непосредственный.
39. Блошин, А. Эффективный C#: 50 специфичных рекомендаций для улучшения ваших программ на C# / А. Блошин. – М.: Вильямс, 2018. – 384 с. – ISBN 978-5-8459-1948-9. – Текст: непосредственный.
40. Мезенцев, Е. Основы разработки ASP.NET Core 5 / Е. Мезенцев. – СПб.: БХВ-Петербург, 2021. – 352 с. – ISBN 978-5-9775-7891-2. – Текст: непосредственный.

41. Гудман, Д. JavaScript jQuery. Исчерпывающее руководство / Д. Гудман. – СПб.: Питер, 2019. – 1072 с. – ISBN 978-5-496-01342-0. – Текст: непосредственный.
42. Албахари, Дж. C# 8.0 и .NET Core 3.0. Карманный справочник / Дж. Албахари, Б. Албахари. – СПб.: Питер, 2020. – 1040 с. – ISBN 978-5-4461-1464-1. – Текст: непосредственный.
43. Петцольд, Ч. Программирование для Windows на языке C#. / Ч. Петцольд. – М.: Вильямс, 2021. – 1104 с. – ISBN 978-5-8459-1905-2. – Текст: непосредственный.
44. Кроули, М. Эффективное управление базами данных. / М. Кроули. – СПб.: Питер, 2019. – 672 с. – ISBN 978-5-496-01601-8. – Текст: непосредственный.
45. Харгривз, Д. Разработка приложений на платформе Microsoft .NET. / Д. Харгривз. – СПб.: БХВ-Петербург, 2018. – 512 с. – ISBN 978-5-9775-7102-8. – Текст: непосредственный.
46. Фримен, А. Программирование ASP.NET MVC 4 с примерами на C# 5.0 / А. Фримен. – М.: Вильямс, 2017. – 832 с. – ISBN 978-5-8459-1832-1. – Текст: непосредственный.
47. Бертон, Б. Программирование на C#. Полное руководство / Б. Бертон. – СПб.: Питер, 2021. – 864 с. – ISBN 978-5-4461-1878-6. – Текст: непосредственный.
48. Криспин, Л. Гибкое тестирование: как сделать тестирование ценным для всех / Л. Криспин, Д. Грегори. – М.: Вильямс, 2020. – 456 с. – ISBN 978-5-8459-1958-8. – Текст: непосредственный.
49. Шрайбер, Р. Архитектура программного обеспечения: от концепций к практике / Р. Шрайбер. – М.: Вильямс, 2019. – 368 с. – ISBN 978-5-8459-1916-8. – Текст: непосредственный.
50. Куусела, Р. Разработка корпоративных приложений на платформе .NET / Р. Куусела. – СПб.: БХВ-Петербург, 2020. – 576 с. – ISBN 978-5-9775-7643-6. – Текст: непосредственный.

ПРИЛОЖЕНИЕ А

Представление графического материала

Графический материал, выполненный на отдельных листах, изображен на рисунках А.1–А.8.

Сведения о ВКРБ

Минобрнауки России
Юго-Западный государственный университет

Кафедра программной инженерии

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА ПО ПРОГРАММЕ БАКАЛАВРИАТА

«Бизнес-проект «Кроссплатформенная программная система поиска попутчиков для автомобильных поездок». Разработка серверной части»

Руководитель ВКРБ
к.ф.-м.н., доцент
Кочура Евгения Павловна

Автор ВКРБ
студент группы ПО-01б
Шеховцов Максим Евгеньевич

ВКРБ 2068443.09.03.04.24.021		
Сведения о ВКРБ		
Номер работы	Фамилия К. О.	Имя И.И.
	Шеховцов М.	
Наименование кафедры	Фамилия А.А.	Имя И.И.
Кафедра		
Наименование кафедры	Фамилия А.А.	Имя И.И.
Бакалаврская квалификационная работа бакалавра		
		ОЭГУ ПО-01б

Рисунок А.1 – Сведения о ВКРБ

BKPB 206844309.03.04.24.021		2																																										
<h2>Цель и задачи разработки</h2> <p>Цель настоящей работы – разработка серверной части кроссплатформенного приложения для поиска попутчиков. Для достижения поставленной цели необходимо решить следующие задачи:</p> <ul style="list-style-type: none">• анализ предметной области;• разработать концептуальную модель программно-информационной системы;• спроектировать и реализовать серверную часть программной системы;• провести тестирование серверной части программной системы.																																												
<table border="1"><tr><td colspan="3"></td><td colspan="3">BKPB 206844309.03.04.24.021</td></tr><tr><td colspan="3"></td><td>Лист</td><td>Номер</td><td>Номер</td></tr><tr><td colspan="3"></td><td>1</td><td>1</td><td>1</td></tr><tr><td colspan="3"></td><td colspan="3">Цель и задачи разработки</td></tr><tr><td colspan="3"></td><td>Лист 2</td><td>Лист 9</td><td></td></tr><tr><td colspan="3"></td><td colspan="3">Выпускная квалификационная работа бакалавра</td></tr><tr><td colspan="3"></td><td colspan="3">ОЭГУ по-016</td></tr></table>						BKPB 206844309.03.04.24.021						Лист	Номер	Номер				1	1	1				Цель и задачи разработки						Лист 2	Лист 9					Выпускная квалификационная работа бакалавра						ОЭГУ по-016		
			BKPB 206844309.03.04.24.021																																									
			Лист	Номер	Номер																																							
			1	1	1																																							
			Цель и задачи разработки																																									
			Лист 2	Лист 9																																								
			Выпускная квалификационная работа бакалавра																																									
			ОЭГУ по-016																																									

Рисунок А.2 – Цель и задачи разработки



Рисунок А.3 – Диаграммы вариантов использования



Рисунок А.4 – ER - Диаграмма базы данных

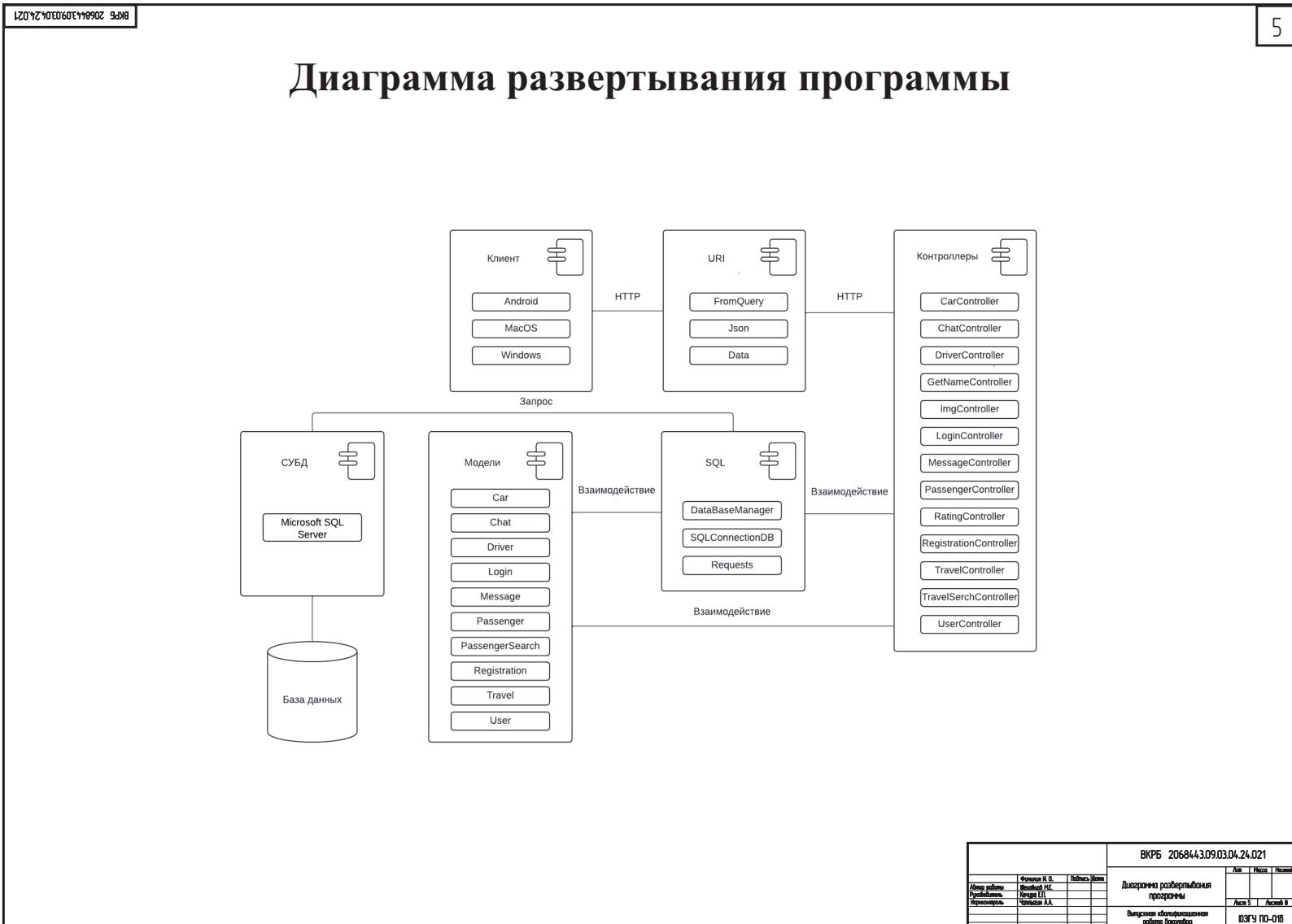
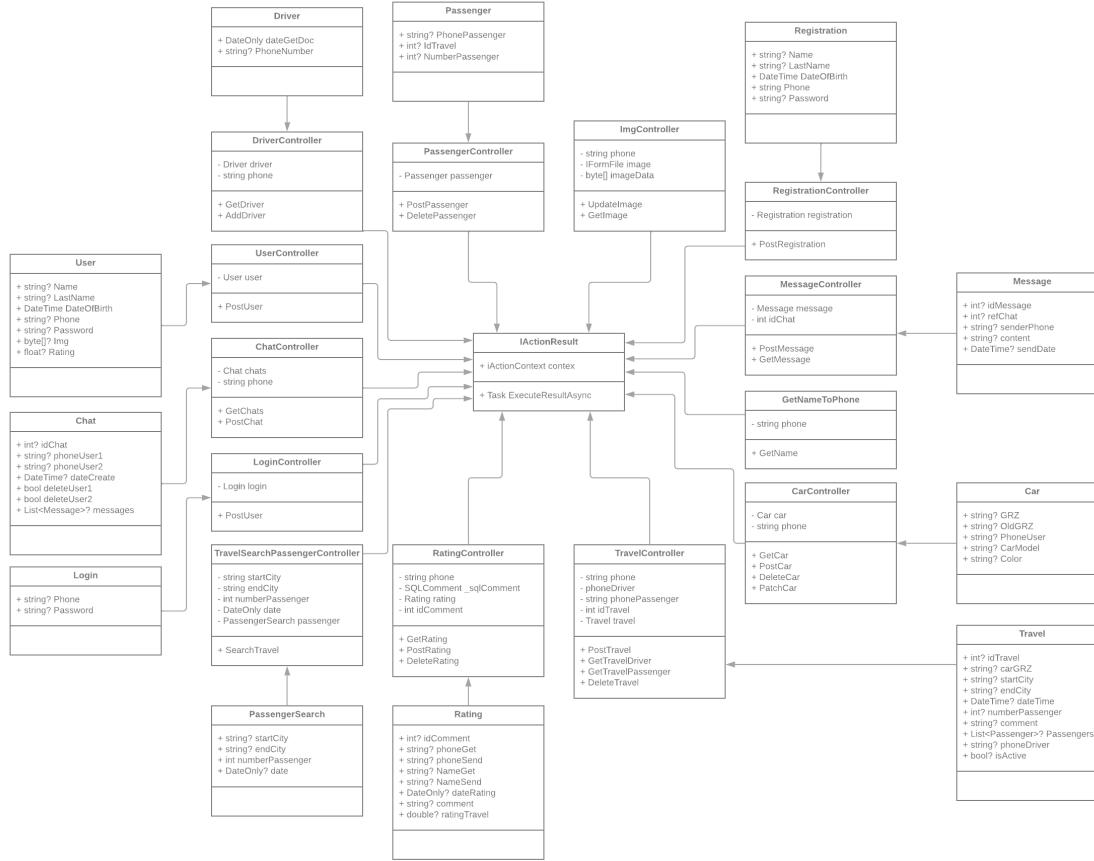


Рисунок А.5 – Диаграмма развертывания программы

Диаграмма классов моделей и контроллеров



ВКРБ 2068443.09.03.04.24.021		
Лист работы	Фамилия К. О.	Номер листа
Лист работы	Бондарук НЕ	
Диаграмма классов моделей и контроллеров		
Лист 5	Лист 9	
Выпускной квалификационный документ		
ОГУ №-016		

Рисунок А.6 – Диаграммы классов моделей и контроллеров

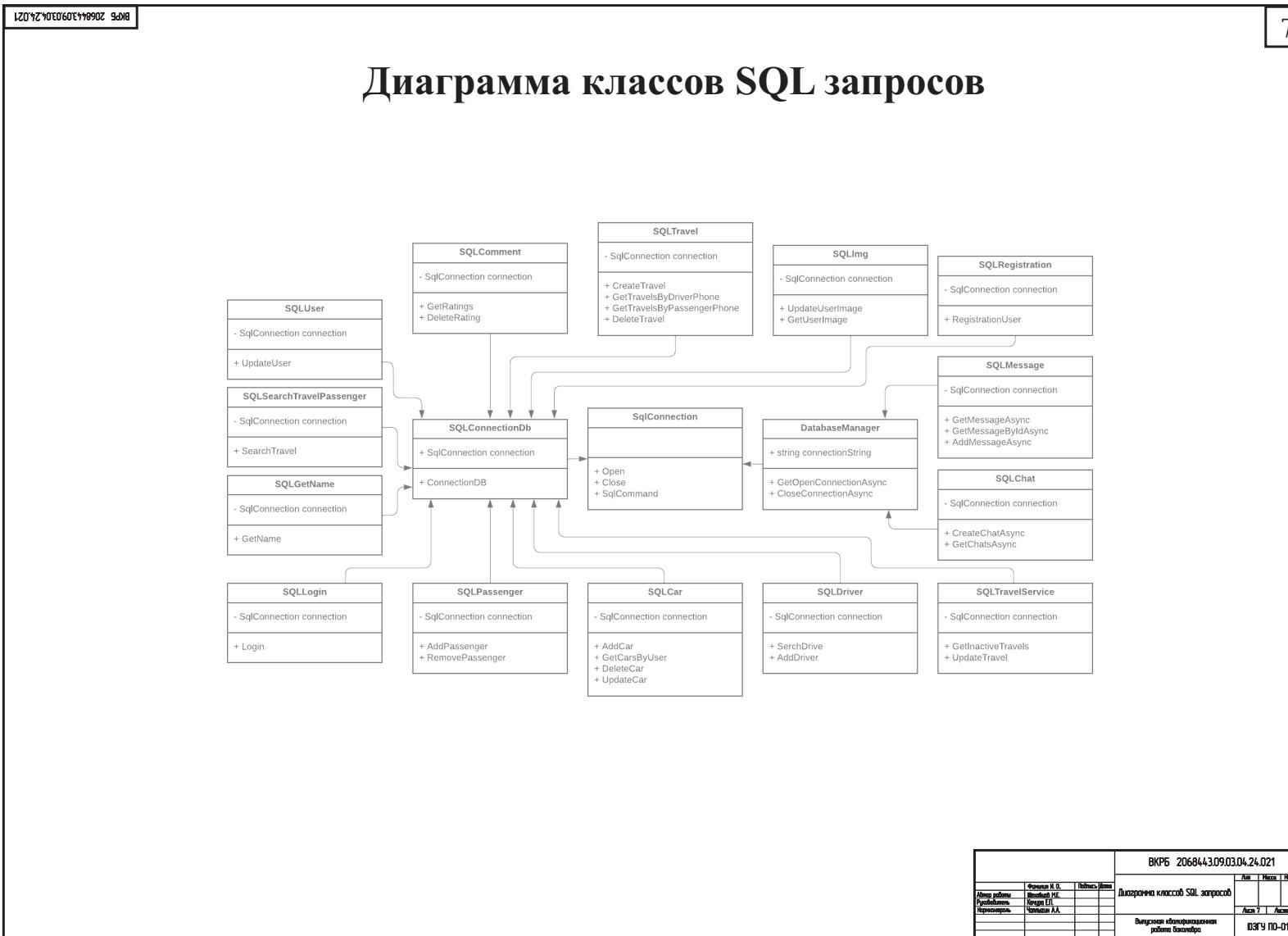


Рисунок А.7 – Диаграмма классов SQL запросов

Заключение

Разработка серверной части кроссплатформенного приложения для поиска попутчиков является актуальной и перспективной задачей. Современный рынок показывает растущий интерес к приложениям для совместных поездок, что обусловлено экономическими и экологическими факторами, а также удобством использования данных сервисов.

В процессе работы были изучены потребности и ожидания пользователей, конкурентная ситуация и особенности технологической реализации серверной части приложения.

Основные результаты работы:

113

1. Проведен анализ предметной области, что позволило выявить ключевые требования и ожидания пользователей от подобных приложений, а также определить основные конкурентные решения и их функциональные особенности.
 2. Разработана концептуальная модель программно-информационной системы, включающая варианты использования системы и требования к серверной части.
 3. Спроектирована и реализована серверная часть программной системы. Разработана архитектура системы с описанием основных компонентов и их взаимодействия, а также ключевых классов и методов.
 4. Проведено тестирование разработанной серверной части программной системы. Выполнено функциональное и системное тестирование для обеспечения корректной работы всех компонентов системы.

Рисунок А.8 – Заключение

ПРИЛОЖЕНИЕ Б

Фрагменты исходного кода программы

Program.cs

```
1  using Microsoft.Extensions.DependencyInjection;
2  using Microsoft.Extensions.Hosting;
3  using AutoStopAPI.Models.SQL;
4
5  namespace AutoStopAPI
6  {
7      public class Program
8      {
9          public static void Main(string[] args)
10         {
11             var builder = WebApplication.CreateBuilder(args);
12
13             // Add services to the container.
14             builder.Services.AddControllersWithViews();
15
16             // Register SQLTravel and SQLConnectionDb
17             builder.Services.AddSingleton<SQLConnectionDb>();
18             builder.Services.AddSingleton<SQLTravelService>();
19
20             // Register the background service
21             builder.Services.AddHostedService<TravelCheckService>();
22
23             var app = builder.Build();
24
25             if (!app.Environment.IsDevelopment())
26             {
27                 app.UseExceptionHandler("/Home/Error");
28                 app.UseHsts();
29             }
30
31             app.UseHttpsRedirection();
32             app.UseStaticFiles();
33
34             app.UseRouting();
35
36             app.UseAuthorization();
37
38             app.MapControllerRoute(
39                 name: "default",
40                 pattern: "{controller=Home}/{action=Index}/{id?}");
41
42             app.Run();
43         }
44     }
45 }
46 }
```

CarController.cs

```

1  using AutoStopAPI.Models;
2  using AutoStopAPI.Models.SQL;
3  using Microsoft.AspNetCore.Mvc;
4  using System.Collections.Generic;
5
6  namespace AutoStopAPI.Controllers
7  {
8      [ApiController]
9      [Route("api/car")]
10     public class CarController : Controller
11     {
12         [HttpGet]
13         public IActionResult GetCar([FromQuery] string phone)
14         {
15             SQLCar sqlCar = new SQLCar();
16             List<Car> cars = sqlCar.GetCarsByUser(phone);
17             if (cars.Count == 0)
18             {
19                 return NotFound(new { message = "No cars found for this user" });
20             }
21
22             return Ok(cars);
23         }
24
25         [HttpPost]
26         public IActionResult PostCar([FromBody] Car car)
27         {
28             SQLCar sqlCar = new SQLCar();
29             bool result = sqlCar.AddCar(car);
30             if (result == false)
31             {
32                 // Возвращаем 400, если добавление не удалось
33                 return BadRequest(new { message = "Failed to add car" });
34             }
35
36             return Ok(car);
37         }
38
39         [HttpDelete]
40         public IActionResult DeleteCar([FromBody] Car car)
41         {
42             SQLCar sqlCar = new SQLCar();
43             bool result = sqlCar.DeleteCar(car.PhoneUser, car.GRZ);
44             if (result == false)
45             {
46                 // Возвращаем 404, если удаление не удалось
47                 return NotFound(new { message = "Car not found or failed to delete" });
48             }
49
50             return Ok(new { message = "Car successfully deleted" });
51         }
52
53         [HttpPatch]

```

```

54     public IActionResult PatchCar([FromBody] Car car)
55     {
56         SQLCar sqlCar = new SQLCar();
57         bool result = sqlCar.UpdateCar(car);
58         if (result == false)
59         {
60             // Возвращаем 400, если обновление не удалось
61             return BadRequest(new { message = "Failed to update car" });
62         }
63
64         return Ok(new { message = "Car update" });
65     }
66 }

```

ChatAPIController.cs

```

1  using AutoStopAPI.Models;
2  using AutoStopAPI.Models.SQL;
3  using Microsoft.AspNetCore.Http;
4  using Microsoft.AspNetCore.Mvc;
5  using System;
6  using System.Collections.Generic;
7  using System.Threading.Tasks;
8
9  namespace AutoStopAPI.Controllers
10 {
11     [Route("api/chat")]
12     [ApiController]
13     public class ChatAPIController : ControllerBase
14     {
15         [HttpGet]
16         public async Task<IActionResult> GetChats([FromQuery] string phone)
17         {
18             SQLChat sqlchat = new SQLChat();
19             List<Chat> chats = await sqlchat.GetChatsAsync(phone);
20             if (chats == null)
21                 return BadRequest("Error. Chat not found!");
22             return Ok(chats);
23         }
24
25         [HttpPost]
26         public async Task<IActionResult> PostChat([FromBody] Chat chat)
27         {
28             SQLChat sqlChat = new SQLChat();
29             chat = await sqlChat.CreateChatAsync(chat);
30             if (chat == null)
31                 return BadRequest("Error. Chat don't create!");
32             return Ok(chat);
33         }
34
35         [HttpDelete]
36         public IActionResult DeleteChat([FromBody] Chat chat)
37         {
38             return Ok();

```

```
39     }
40   }
41 }
```

DriverController.cs

```
1  using AutoStopAPI.Models.SQL;
2  using AutoStopAPI.Models;
3  using Microsoft.AspNetCore.Mvc;
4
5  namespace AutoStopAPI.Controllers
6  {
7      [ApiController]
8      [Route("api/driver")]
9      public class DriverController : ControllerBase
10     {
11         //Проверяем является ли User водителем
12         [HttpGet]
13         public IActionResult GetDriver([FromQuery] string phone)
14         {
15             SQLDriver sqlDriver = new SQLDriver();
16             var driver = sqlDriver.SerchDrive(phone);
17
18             if (driver == null)
19             {
20                 // Возвращаем 404, если запись не найдена
21                 return NotFound(new { message = "Driver not found" });
22             }
23
24             return Ok(driver);
25         }
26
27         [HttpPost]
28         public IActionResult AddDriver([FromBody] Driver driver)
29         {
30             SQLDriver sqlDriver = new SQLDriver();
31             bool result = sqlDriver.AddDriver(driver);
32             if (result == false)
33             {
34                 // Возвращаем 404, если запись не найдена
35                 return NotFound(new { message = "Driver not found" });
36             }
37
38             return Ok(driver);
39         }
40     }
41 }
```

GetNameToPhone.cs

```
1  using AutoStopAPI.Models.SQL;
2  using Microsoft.AspNetCore.Http;
3  using Microsoft.AspNetCore.Mvc;
4
5  namespace AutoStopAPI.Controllers
```

```

6  {
7      [Route("api/GetName")]
8      [ApiController]
9      public class GetNameToPhone : ControllerBase
10     {
11         [HttpGet]
12         public IActionResult GetName([FromQuery] string phone)
13         {
14             SQLGetName getname = new SQLGetName();
15             string name = getname.GetName(phone);
16             return Ok(name);
17         }
18     }
19 }
```

ImgAPIController.cs

```

1  using Microsoft.AspNetCore.Http;
2  using Microsoft.AspNetCore.Mvc;
3  using System.IO;
4  using System.Threading.Tasks;
5  using AutoStopAPI.Models.SQL;
6
7  namespace AutoStopAPI.Controllers
8  {
9      [Route("api/img")]
10     [ApiController]
11     public class ImgAPIController : ControllerBase
12     {
13         private readonly SQLImg _sqlImg;
14
15         public ImgAPIController()
16         {
17             _sqlImg = new SQLImg();
18         }
19
20         [HttpPatch]
21         public async Task<IActionResult> UpdateImage(string phone, IFormFile
22             image)
23         {
24             if (image == null || image.Length == 0)
25             {
26                 return BadRequest("No image file provided.");
27             }
28
29             byte[] imageData;
30             using (var ms = new MemoryStream())
31             {
32                 await image.CopyToAsync(ms);
33                 imageData = ms.ToArray();
34             }
35
36             bool updateSuccess = _sqlImg.UpdateUserImage(phone, imageData);
37             if (updateSuccess)
38             {
```

```

38         return Ok("Image updated successfully.");
39     }
40     else
41     {
42         return StatusCode(StatusCodes.Status500InternalServerError, "Error
43             updating image.");
44     }
45
46     [HttpGet("{phone}")]
47     public IActionResult GetImage(string phone)
48     {
49         var imageData = _sqlImg.GetUserImage(phone);
50         if (imageData != null)
51         {
52             return File(imageData, "image/jpeg"); // Assuming the image format
53                 is jpeg
54         }
55         else
56         {
57             return NotFound("Image not found.");
58         }
59     }
60 }
```

LoginAPIController.cs

```

1  using AutoStopAPI.Models;
2  using AutoStopAPI.Models.SQL;
3  using Microsoft.AspNetCore.Mvc;
4
5  namespace AutoStopAPI.Controllers
6  {
7      [ApiController]
8      [Route("api/login")]
9      public class LoginAPIController : Controller
10     {
11         [HttpPost]
12         public IActionResult PostUser([FromBody] Login login)
13         {
14             SQLLogin sqlLogin = new SQLLogin();
15             var result = sqlLogin.Login(login);
16             return Ok(result);
17         }
18     }
19 }
```

MessageAPIController.cs

```

1  using AutoStopAPI.Models;
2  using AutoStopAPI.Models.SQL;
3  using Microsoft.AspNetCore.Http;
4  using Microsoft.AspNetCore.Mvc;
5  using System.Collections.Generic;
```

```

6   using System.Threading.Tasks;
7
8   namespace AutoStopAPI.Controllers
9   {
10      [Route("api/message")]
11      [ApiController]
12      public class MessageAPIController : ControllerBase
13      {
14          [HttpPost]
15          public async Task<IActionResult> PostMessage([FromBody] Message message)
16          )
17          {
18              SQLMessage sqlMessage = new SQLMessage();
19              int idMessage = await sqlMessage.AddMessageAsync(message);
20              return Ok(idMessage);
21          }
22
23          [HttpGet]
24          public async Task<IActionResult> GetMessage([FromQuery] int idChat)
25          {
26              var sqlMessage = new SQLMessage();
27              List<Message> messages = await sqlMessage.GetMessageByIdAsync(idChat)
28              ;
29              return Ok(messages);
30          }
31      }
32  }

```

PassengerAPIController.cs

```

1   using AutoStopAPI.Models.SQL;
2   using AutoStopAPI.Models;
3   using Microsoft.AspNetCore.Mvc;
4
5   [Route("api/passenger")]
6   [ApiController]
7   public class PassengerAPIController : ControllerBase
8   {
9       [HttpPost]
10      public IActionResult PostPassenger([FromBody] Passenger passenger)
11      {
12          SQLPassenger sqlPassenger = new SQLPassenger();
13          var result = sqlPassenger.AddPassenger(passenger);
14
15          switch (result)
16          {
17              case AddResult.Added:
18                  return Ok("Passenger added to travel");
19              case AddResult.AlreadyExists:
20                  return Conflict("Passenger already exists in travel");
21              case AddResult.Error:
22                  return BadRequest("Error! Passenger not added to travel");
23              default:
24                  return StatusCode(StatusCodes.Status500InternalServerError, "Unknown error");
25          }
26      }
27  }

```

```

25     }
26 }
27
28 [HttpDelete]
29 public IActionResult DeletePassenger([FromBody] Passenger passenger)
30 {
31     SQLPassenger sqlPassenger = new SQLPassenger();
32     bool result = sqlPassenger.RemovePassenger(passenger);
33
34     if (result)
35     {
36         return Ok("Passenger removed from travel");
37     }
38     else
39     {
40         return NotFound("Passenger not found in travel or error occurred");
41     }
42 }
43 }
```

RatingAPIController.cs

```

1  using Microsoft.AspNetCore.Http;
2  using Microsoft.AspNetCore.Mvc;
3  using AutoStopAPI.Models.SQL;
4  using System.Collections.Generic;
5
6  namespace AutoStopAPI.Controllers
7  {
8      [Route("api/rating")]
9      [ApiController]
10     public class RatingAPIController : ControllerBase
11     {
12         private readonly SQLComment _sqlComment;
13
14         public RatingAPIController()
15         {
16             _sqlComment = new SQLComment();
17         }
18
19         [HttpGet]
20         public IActionResult GetRating(string phone)
21         {
22             try
23             {
24                 List<Rating> ratings = _sqlComment.GetRatings(phone);
25                 if (ratings == null || ratings.Count == 0)
26                 {
27                     return NotFound("No ratings found for the given phone number.");
28                 }
29                 return Ok(ratings);
30             }
31             catch (Exception ex)
32             {
```

```

33     return StatusCode(StatusCodes.Status500InternalServerError, $"Error
34         retrieving data: {ex.Message}");
35     }
36 }
37 [HttpPost]
38 public IActionResult PostRating([FromBody] Rating rating)
39 {
40     if (rating == null || string.IsNullOrEmpty(rating.phoneGet) || string
41         .IsNullOrEmpty(rating.phoneSend))
42     {
43         return BadRequest("Invalid rating data.");
44     }
45     try
46     {
47         int idComment = _sqlComment.AddRating(rating);
48         if (idComment > 0)
49         {
50             return Ok(idComment);
51         }
52         else
53         {
54             return StatusCode(StatusCodes.Status500InternalServerError, "Error
55                 adding rating.");
56         }
57     }
58     catch (Exception ex)
59     {
60         return StatusCode(StatusCodes.Status500InternalServerError, $"Error
61             adding rating: {ex.Message}");
62     }
63 }
64 [HttpDelete]
65 public IActionResult DeleteRating(int idComment)
66 {
67     try
68     {
69         bool isDeleted = _sqlComment.DeleteRating(idComment);
70         if (isDeleted)
71         {
72             return Ok("Rating deleted successfully.");
73         }
74         else
75         {
76             return NotFound("Rating not found.");
77         }
78     }
79     catch (Exception ex)
80     {
81         return StatusCode(StatusCodes.Status500InternalServerError, $"Error
82             deleting rating: {ex.Message}");
83     }
84 }
```

```
82    }
83  }
84 }
```

RegistrationAPIController.cs

```
1  using AutoStopAPI.Models;
2  using AutoStopAPI.Models.SQL;
3  using Microsoft.AspNetCore.Mvc;
4
5  namespace AutoStopAPI.Controllers
6  {
7      [ApiController]
8      [Route("api/registration")]
9      public class RegistrationAPIController : Controller
10     {
11         [HttpPost]
12         public IActionResult PostRegistration([FromBody] Registration
13             registration)
14         {
15             SQLRegistration SQLReg = new SQLRegistration();
16             SQLReg.RegistrationUser(registration);
17             return Ok();
18         }
19     }
}
```

TravelAPIController.cs

```
1  using AutoStopAPI.Models.SQL;
2  using AutoStopAPI.Models;
3  using Microsoft.AspNetCore.Mvc;
4
5  namespace AutoStopAPI.Controllers
6  {
7      [Route("api/Travel")]
8      [ApiController]
9      public class TravelAPIController : Controller
10     {
11         [HttpPost]
12         public IActionResult PostTravel([FromBody] Travel travel)
13         {
14             SQLTravel sqlTravel = new SQLTravel();
15             int? result = sqlTravel.CreateTravel(travel);

16             if (result == null)
17             {
18                 return BadRequest();
19             }
20             return Ok(result);
21         }

22         [HttpGet]
23         public IActionResult GetTravelDriver([FromQuery] string phoneDriver)
24         {
25 }
```

```

27     SQLTravel sqlTravel = new SQLTravel();
28     List<Travel> travels = sqlTravel.GetTravelsByDriverPhone(phoneDriver)
29         ;
30
31     if (travels == null || travels.Count == 0)
32     {
33         return NotFound(new { message = "No travels found for the given
34             driver phone number" });
35     }
36     return Ok(travels);
37 }
38
39 [HttpGet("passenger")]
40 public IActionResult GetTravelPassenger([FromQuery] string
41     phonePassenger)
42 {
43     SQLTravel sqlTravel = new SQLTravel();
44     List<Travel> travels = sqlTravel.GetTravelsByPassengerPhone(
45         phonePassenger);
46
47     if (travels == null || travels.Count == 0)
48     {
49         return NotFound(new { message = "No travels found for the given
50             passenger phone number" });
51     }
52     return Ok(travels);
53 }
54
55 [HttpDelete]
56 public IActionResult DeleteTravel(int idTravel)
57 {
58     SQLTravel sqlTravel = new SQLTravel();
59     bool result = sqlTravel.DeleteTravel(idTravel);
60
61     if (!result)
62     {
63         return NotFound(new { message = "Travel not found or could not be
64             deleted" });
65     }
66     return Ok(new { message = "Travel deleted successfully" });
67 }
68 }
69 }
```

TravelSearchPassengerAPIController.cs

```

1  using AutoStopAPI.Models.SQL;
2  using AutoStopAPI.Models;
3  using Microsoft.AspNetCore.Mvc;
4  using Microsoft.Data.SqlClient;
5
6  namespace AutoStopAPI.Controllers
7  {
8      [Route("api/searchTravel")]
9      [ApiController]
```

```

10 public class TravelSearchPassengerAPIController : Controller
11 {
12     [HttpGet]
13     public IActionResult SearchTravel([FromQuery] string startCity, string
14         endCity, int numberPassenger, DateOnly date)
15     {
16         PassengerSearch passenger = new PassengerSearch();
17         passenger.startCity = startCity;
18         passenger.endCity = endCity;
19         passenger.numberPassenger = numberPassenger;
20         passenger.date = date;
21         try
22         {
23             SQLSearchTravelPassenger search = new SQLSearchTravelPassenger();
24             List<Travel> result = search.SearchTravel(passenger);
25             return Ok(result);
26         }
27         catch (Exception ex)
28         {
29             // Логируем ошибку
30             Console.WriteLine("Ошибка при поиске поездки: " + ex.Message);
31             return StatusCode(500, "Внутренняя ошибка сервера");
32         }
33     }
34 }

```

UserAPIController.cs

```

1 using AutoStopAPI.Models;
2 using AutoStopAPI.Models.SQL;
3 using Microsoft.AspNetCore.Http;
4 using Microsoft.AspNetCore.Mvc;
5
6 namespace AutoStopAPI.Controllers
7 {
8     [Route("api/User")]
9     [ApiController]
10    public class UserAPIController : ControllerBase
11    {
12        [HttpPatch]
13        public IActionResult PostUser([FromBody] User user)
14        {
15            SQLUser sqlUser = new SQLUser();
16            bool result = sqlUser.UpdateUser(user);
17            if (result == false)
18            {
19                return BadRequest();
20            }
21            return Ok(result);
22        }
23    }
24 }

```

Место для диска