МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное бюджетное образовательное учреждение
высшего образования
«Тольяттинский государственный университет»

Кафедра «Прикладная математика и информатика»
(наименование)

09.03.03 Прикладная информатика
(код и наименование направления подготовки, специальности)

Разработка программного обеспечения
(направленность (профиль) / специализация)

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА (БАКАЛАВРСКАЯ РАБОТА)

на тему «Разработка мобильного приложения для заказа такси»

Обучающийся А.В. Томов
(Инициалы Фамилия) (личная подпись)

Руководитель канд. техн. наук Д.Г. Токарев
ученая степень (при наличии), ученое звание (при наличии), Инициалы Фамилия)

Тольятти 2025

Аннотация

Тема выпускной квалификационной работы: «Разработка мобильного приложения для заказа такси».

Объект исследования – процесс организации взаимодействия с целью организации поездок между клиентом и водителем посредством мобильного приложения.

Предмет исследования – технологии разработки клиент–серверных приложений для Android с использованием C# ASP.NET, Avalonia и SQLite.

Цель работы: разработка мобильного приложения для платформы Android и серверной части на Windows, обеспечивающего заказ такси, управление заказами водителями и просмотр истории заказов и отзывов.

Задачи:

1. Анализ и характеристика предметной области (рынок такси, требования к функциональности приложений);

2. Выбор технологии разработки программного обеспечения (C# ASP.NET, Avalonia, SQLite);

3. Разработка и тестирование мобильного и серверного приложения с использованием ASP.NET посредством HTTP запросов для обмена данными.

Работа представлена в трех главах, посвященными изучению предметной области, выбору технологии, описанию разработки и тестированию приложения. А также материалы в виде схем, таблиц, рисунков с демонстрацией интерфейса приложения.

В первой главе выполнен анализ предметной области, обзор существующих технологий реализации, обоснование выбора технологий реализации.

Во второй главе выполнено моделирование программного обеспечения и бизнес-процессов, а также проектирование графического интерфейса и разработка логики общения между клиентом и сервером.

В третьей главе произведена разработка мобильного и серверного приложения с использованием выбранных технологий, а также разработаны автоматические unit-тесты и ручные тесты, которые отражают правильность работы всего приложения.

В заключении даны выводы по главам и итоги практической реализации разработанного программного обеспечения.

Результатом выпускной квалификационной работы является правильно функционирующий комплекс из мобильного и серверного приложений для заказа такси, который обеспечивает обработку поступающих от пользователей заказов, выбор активных заказов водителем, и хранение информации о завершенных заказах и их оценки.

Работа содержит три главы, 53 страницы, 25 рисунков, 3 таблицы, 5 приложений, 26 библиографических источников.

Оглавление

Введение

Применение мобильных технологий позволяет существенно расширить доступность услуг такси и повысить удобство для пользователей. Рынок приложений для заказа такси активно развивается, что обуславливает необходимость разработки новых решений, отвечающих современным требованиям к функциональности, надежности и безопасности.

Объектом исследования является процесс организации взаимодействия между клиентами и водителями посредством мобильного приложения.

Предмет исследования – технологии разработки клиент–серверных приложений для Android с использованием C# ASP.NET, Avalonia и SQLite.

Новизна работы заключается в разработке мобильного приложения для заказа такси с акцентом на удобство использования, стабильную работу и интеграцию с серверной частью.

Методы, используемые при работе над ВКР – клиент–серверные технологии разработки приложений; объектно–ориентированное программирование на языке C# ASP.NET [15]; проектирование баз данных SQLite; использование HTTP запросов для обмена данными между клиентом и сервером; диаграммы состояния процессов в нотациях IDEF0 и BPMN.

Цель работы: разработка мобильного приложения для платформы Android и серверной части, обеспечивающего заказ такси, управление заказами водителями и просмотр истории заказов и отзывов.

Задачами выпускной квалификационной работы являются: анализ и характеристика предметной области; выбор технологии разработки программного обеспечения; реализация и тестирование мобильного приложения и серверной части.

Актуальность заявленной темы обусловлена высоким спросом на услуги такси и потребностью в удобных и надежных решениях для заказа такси с использованием мобильных устройств. В основу концепции разрабатываемого приложения заложена идея предоставления пользователям

простого и интуитивно понятного интерфейса для заказа такси с отслеживанием водителя в режиме реального времени.

Область применения – потенциальные пользователи услуг такси, а также водители, использующие приложение для получения заказов.

Выпускная квалификационная работа состоит из введения, трех глав, заключения, списка литературы и приложений.

В первой главе выполнен анализ предметной области, обзор существующих технологий реализации, обоснование выбора технологий реализации.

Во второй главе выполнено моделирование программного обеспечения и бизнес-процессов, а также проектирование графического интерфейса и разработка логики общения между клиентом и сервером.

В третьей главе произведена разработка мобильного и серверного приложения с использованием выбранных технологий, а также разработаны автоматические unit-тесты и ручные тесты, которые отражают правильность работы всего приложения.

В заключении даны выводы по главам и итоги практической реализации разработанного программного обеспечения.

Результатом выпускной квалификационной работы является правильно функционирующий комплекс из мобильного и серверного приложений для заказа такси, который обеспечивает обработку поступающих от пользователей заказов, выбор активных заказов водителем, и хранение информации о завершенных заказах и их оценки.

При работе над выпускной квалификационной работой была изучена литература и информационные ресурсы на русском и английском языках [1–26].

Работа содержит три главы, 53 страницы, 25 рисунков, 3 таблицы, 5 приложений, 26 библиографических источников.

Глава 1 Анализ проблемы разработки программного обеспечения для клиент-серверного приложения

1.1 Анализ и характеристика предметной области

На начальном этапе разработки приложения проводится анализ предприятия и его структуры, а затем определение модели будущей системы [9].

Сфера пассажирских перевозок отличается высокой конкуренцией и устойчивым ростом потребности в комфортных и бюджетных решениях. Многочисленные игроки предлагают клиентам мобильные платформы для вызова транспорта, что требует разработки продукта с уникальными преимуществами, способного отвечать ожиданиям пользователей и конкурировать с существующими предложениями.

Ключевые участники экосистемы:

– потребители: лица, использующие сервис для внутригородских или междугородних поездок;

– исполнители: водители, осуществляющие перевозку пассажиров.

– Координаторы: в классических системах – сотрудники, распределяющие заявки. В цифровых решениях эти функции выполняются автоматизированными алгоритмами.

Современные требования к функционалу приложений включают:

– минималистичный интерфейс для мгновенного вызова транспорта;

– интерактивную карту с отслеживанием автомобиля в реальном времени;

– четкую систему тарификации с опцией выбора подходящего варианта;

– поддержку различных способов оплаты (наличные, карты, цифровые кошельки);

– инструменты для оценки качества сервиса и обратной связи.

Создаваемая мобильная платформа для вызова транспорта направлена на решение выявленных запросов пользователей, предлагая комфортное и стабильное решение для городских перемещений. Архитектура серверной компоненты будет построена на базе ASP.NET, обеспечивающей обработку высоких нагрузок и гибкое расширение функционала. Для клиентской части выбрана кроссплатформенная разработка под Android с применением языка C# .NET и фреймворка Avalonia, что гарантирует адаптивность интерфейса.

Ключевые технические решения:

– хранение данных: для хранения информацией о заказах, водителях и оценках клиентов используется файловая база данных SQLite, оптимизированная для мобильных решений;

– функциональная автономия: приложение проектируется как замкнутая система, объединяющая модуль бронирования транспорта, систему мониторинга статусов и механизм обратной связи.

Преимущества подхода:

– использование SQLite позволяет работать с данными офлайн, критически важное для пользователей в зонах слабого интернет–покрытия;

– кроссплатформенность Avalonia упрощает потенциальный перенос решения на iOS без полного переписывания кода;

– ASP.NET на стороне сервера обеспечивает безопасную синхронизацию данных между клиентом и серверной инфраструктурой.

1.2 Обзор существующих технологий реализации

Выбор способа разработки программного обеспечения основывается на выборе и анализе технологий, выявлении плюсов и минусов рассматриваемых альтернатив.

Проект состоит из двух частей: клиентской и серверной. Клиентская часть представляет собой интерфейс пользователя, реализуется на мобильном устройстве. Серверная часть, отвечает на запросы клиентов,

выступает хранилищем данных и обеспечивает серверную логику работы приложения.

Для клиентской части планируется использование языка программирования C# .NET с фреймворком Avalonia для создания кроссплатформенного интерфейса. Это позволит обеспечить современный и удобный пользовательский опыт [15, 21].

Существуют различные подходы к разработке клиентской части приложения (таблица 1):

– применение фреймворков позволяет ускорить разработку за счет использования готовых компонентов и шаблонов. Однако выбор фреймворка должен быть обоснованным и соответствовать требованиям проекта;

– разработка интерфейса с использованием библиотек JavaScript (например, JQuery) может быть эффективной для реализации отдельных элементов интерфейса, но не подходит для создания сложного приложения.

Таблица 1 – Сравнительная характеристика фреймворков

Фреймворк	Поддержка мобильных платформ	Поддерживаемые языки	Платформы разработки	Особенности
React Native	Да	JavaScript	Android, iOS	Кроссплатформенная разработка, высокая производительность
Flutter	Да	Dart	Android, iOS	Быстрая разработка, богатый набор виджетов
Avalonia	Да	C#	Android, iOS	Использование .NET, кроссплатформенность

В качестве серверной технологии используется ASP.NET в сочетании с базой данных SQLite. SQLite выбрана как легковесная и надежная база данных, не требующая сложной настройки и администрирования [17].

Разработка мобильного приложения ведется для платформы Android с использованием языка программирования C# .NET. Для разработки требуется установка Android SDK и конфигурирование среды разработки.

«В настоящее время существует несколько сред разработки:

– Android Studio – официальная среда разработки от компании Google, которая предоставляет необходимые компоненты и инструменты для разработки качественных Android –приложений;

– Visual Studio – интегрированная среда разработки от компании Microsoft, поддерживающая разработку на C# .NET и позволяет разрабатывать кроссплатформенные приложения в том числе и для платформы Android (с использованием Xamarin или других инструментов)» [1];

– JetBrains Rider – это мощная интегрированная среда разработки (IDE), разработанная компанией JetBrains. Она предназначена для разработки много платформенных приложений на основе технологического стека .NET, включая C#, F# и VB.NET.

Таблица 2 демонстрирует разные инструменты разработки мобильных приложений для платформы Android. В качестве сравнение, были выбраны особенности сред и поддерживаемые языки программирования.

Таблица 2 Инструменты разработки мобильных приложений

Среда разработки	Поддержка актуальных версий платформы	Интеграция с SDK	Интеграция с AVD	Особенности
Android Studio	+	+	+	Официальная среда разработки, богатый функционал
Visual Studio	+ (с Xamarin)	–	–	Поддержка C# .NET, кроссплатформенность
JetBrains Rider	+	+ (через SDK)	+ (через плагины/SDK)	Мощная поддержка .NET и Unity, кроссплатформенность, интеллектуальные инструменты разработки.

1.3 Обоснование выбора технологии реализации

Изучение доступных решений помогло выделить ключевые преимущества и недостатки рассматриваемых альтернатив. Это, в свою очередь, способствовало выбору методов и инструментов, наиболее полно соответствующих требованиям разработки проекта.

Данный проект основывается на клиент-серверной архитектуре. В этой архитектуре мобильное приложение функционирует на стороне клиента, а серверная часть обрабатывает запросы и управляет данными.

Клиент–серверная архитектура подразумевает следующие компоненты:

– мобильное приложение: предоставляет интерфейс для пользователей (клиентов и водителей), позволяя им взаимодействовать с системой. Реализуется с помощью C# .NET на Avalonia;

– серверная часть: отвечает за обработку запросов, управление данными и реализацию бизнес–логики приложения. Реализуется с помощью C# ASP.NET;

– база данных: хранит информацию о пользователях, поездках, водителях, текущих заказов и тарифов. Используется файловая база данных SQLite

Взаимодействие компонентов:

– пользователь: регистрируется/авторизуется через мобильное приложение, оформляет заказ, вводит данные откуда и куда хочет добрать, желаемую сумму и создает заказ, ожидая прием его со стороны водителей. Все запросы выполняются через HTTP запросы;

– серверная часть: обрабатывает запросы на авторизацию, регистрации, формирования и обработки заказов, отправляет данные о созданных заказов водителям, которые принимают заказ. Вся информация обрабатывается через HTTP запросы.

Преимущества клиент–серверной архитектуры:

– централизованное управление: упрощает администрирование и обновление системы, так как все данные и ресурсы находятся на сервере;

– безопасность: данные хранятся на сервере, что позволяет реализовать эффективные меры безопасности и снизить риск утечек информации;

– масштабируемость: легко добавлять новых пользователей и расширять серверные ресурсы по мере роста нагрузки, что обеспечивает гибкость системы;

– параллельная обработка: сервер может обрабатывать несколько запросов одновременно, что значительно повышает производительность приложения.

– поддержка различных устройств: клиент–серверная архитектура позволяет разрабатывать приложения для разных платформ (мобильные, веб и десктопные), обеспечивая доступ к одной и той же серверной логике.

Клиентская часть (фронтенд) реализована на базе фреймворке Avalonia UI, на языке C# .NET. Данный инструмент обеспечивает разработку универсальных интерфейсов, адаптированных под разные разрешения экранов, с сохранением визуальной согласованности и оптимизированной скорости работы, благодаря встроенному во фреймворке движку SkiaEngine, который отвечает за отрисовку графического интерфейса. Avalonia дает возможность создавать приложения с единой кодовой базой, что гарантирует идентичное поведение системы на устройствах с разными техническими характеристиками [21].

Серверная часть (бэкенд) реализована на базе платформы ASP.NET, обеспечивающая:

– поддержку высоких нагрузок за счет асинхронной обработки запросов;

– механизмы шифрования данных для защиты пользовательской информации;

– гибкое масштабирование под растущие объемы транзакций.

В качестве основного хранилища для управления данными задействована файловая база данных SQLite – легковесное решение с минимальными требованиями к ресурсам [12]. Её преимущества включают:

– автономную работу без необходимости развертывания отдельного сервера;

– встроенную поддержку транзакций для сохранения целостности данных.

В качестве инструментария разработки используется интегрированная среда разработки JetBrains Rider, которая предлагает:

– глубокую интеграцию с технологическим стеком (C#, Avalonia, ASP.NET);

– интеллектуальные функции: автоматический рефакторинг, профилирование кода, Git–интеграцию;

– кроссплатформенную поддержку (Windows, macOS, Linux).

При выборе технологий учитывались следующие факторы:

– производительность: приложение должно работать быстро и эффективно на целевых устройствах;

– масштабируемость: в будущем должна быть возможность дальнейшего улучшения и расширение кодовой базы для покрытия всех необходимых потребностей;

– безопасность: данные пользователей должны быть надежно защищены от несанкционированного доступа;

– удобство разработки: использование современных инструментов и технологий должно упростить процесс разработки и тестирования приложения.

Изучение ключевых факторов показало потребность в создании мобильного решения с интуитивным интерфейсом и возможностью анализа архивных данных о поездках и откликах пользователей.

Выводы по первой главе

В качестве основной операционной системы выбран Android, и среда разработки – JetBrains Rider. В роли ключевого инструмента выступает фреймворк AvaloniaUI на языке C# .NET, обеспечивающий кроссплатформенность и современный дизайн. В качестве серверной технологии используется ASP.NET, а для хранения данных – база данных SQLite.

Объединение этих технологий позволяет создать полноценное программное обеспечение для заказа такси с удобным пользовательским интерфейсом, надежной серверной частью и эффективным хранением данных. Выбор указанных технологий обоснован их преимуществами в плане производительности, масштабируемости, безопасности и удобства разработки. JetBrains Rider обеспечивает высокую продуктивность разработки благодаря интеллектуальным инструментам и поддержке .NET.

Глава 2 Проектирование программного обеспечения

2.1 Моделирование бизнес–процессов

«Жизненный цикл информационной системы и программного обеспечения состоит из четырех основных этапов:

- анализ,
- дизайн,
- программирование,
- тестирование.

Перед началом аналитической работы проводится оценка осуществимости проекта, включая выбор оптимальных методов его реализации» [5]. На этапе системного исследования данные фиксируются с помощью наглядных диаграмм, отображающих движение информации [9].

Разработка архитектуры программного обеспечения для службы такси начинается с изучения основных цепочек транспортной компании. Основным подходом является проектирование системы на основе методологии IDEF, где ключевая роль отводится функциональным моделям. Сначала строится схема процесса «Приема и исполнения заказа» в соответствии со стандартом IDEF0, детализирующая последовательность действий – от инициирования запроса на поездку до окончательного расчета с клиентом.

При построении «диаграммы как есть» (рисунок 1) отражается существующий алгоритм взаимодействия диспетчера, водителя и пассажира без участия мобильного приложения: приём заказа по телефону или через веб-портал, передача заявки водителю, ручное отслеживание маршрута и получение обратной связи. Этот анализ служит отправной точкой для выявления узких мест и обоснования целесообразности внедрения мобильного клиента, который автоматизирует регистрацию пользователя, показ доступных машин на карте, оформление маршрута и онлайн-оплату.

«Для категоризации потоков информаций используется четыре вида стрелок:

− входящие – указывают на то, что потребляется в ходе выполнения исследуемого процесса;

− управляющие – указывают ограничения и инструкции, которые влияют на ход процесса;

− выход – результат выполнения процесса;

− исполняющий механизм (Mechanism) – исполнитель процесса и/или то, что используется для исполнения процесс» [1, 6].

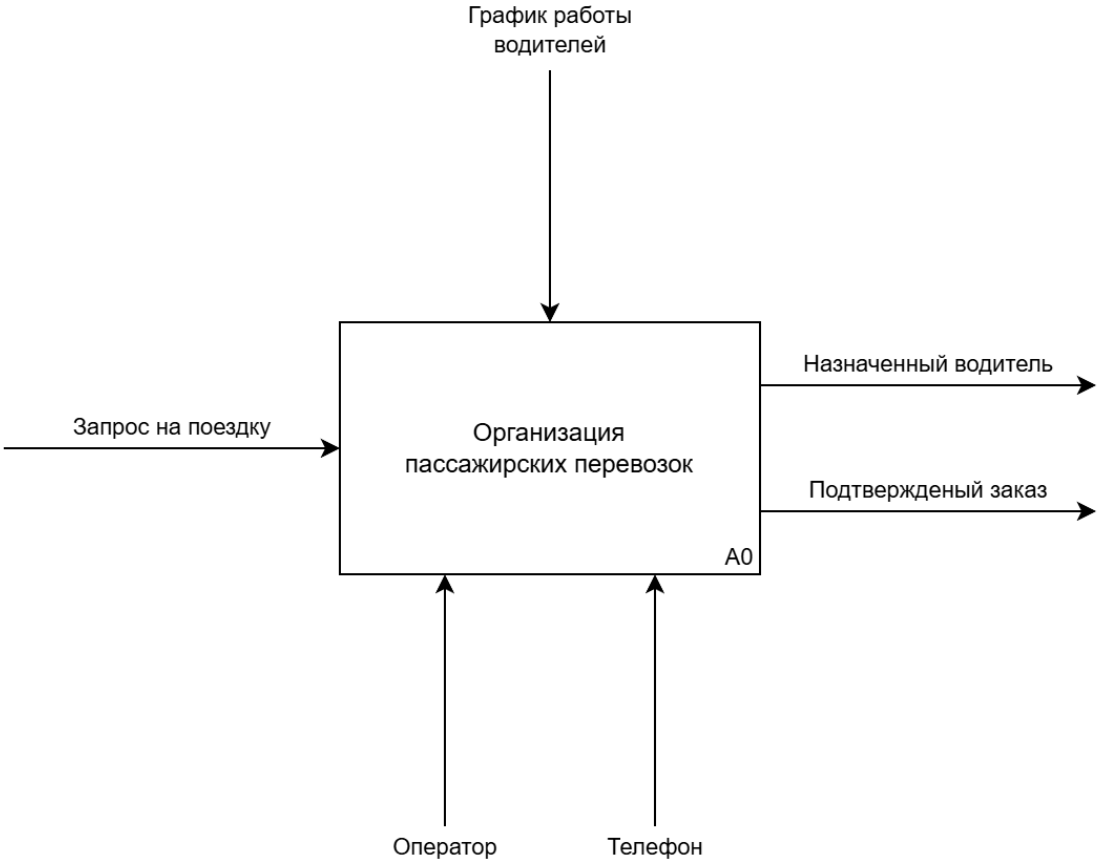

Рисунок 1 – Диаграмма «Как есть»

Декомпозиция классического процесса заказа такси разделяет его на три взаимосвязанных подпроцесса:

а) принятие звонка от клиента:

1) оператор принимает звонок, фиксирует точку посадки и высадки, уточняет пожелания клиента;

2) данные клиента (имя, контактный номер, адрес) записываются вручную или вводятся в электронную систему учёта;

3) формируется заказ с указанием адресов посадки и прибытия, времени подачи машины заказчику и предварительной стоимости заказа.

б) поиск и назначение водителя

1) диспетчер проверяет доступность водителей в зоне запроса через электронную базу или радиообмен;

2) выбирается ближайший свободный водитель с учётом его текущего местоположения и загруженности дорог;

3) информация о заказе передаётся водителю по рации или телефону, включая адрес клиента и детали маршрута.

в) подтверждение заказа клиенту

1) оператор подтверждает клиенту принятие заказа по телефону;

2) после назначения водителя клиенту звонит диспетчер или сам водитель для уточнения времени прибытия;

В процессе поездки при необходимости корректируется время подачи машины на основе обратной связи от водителя (рисунок 2).

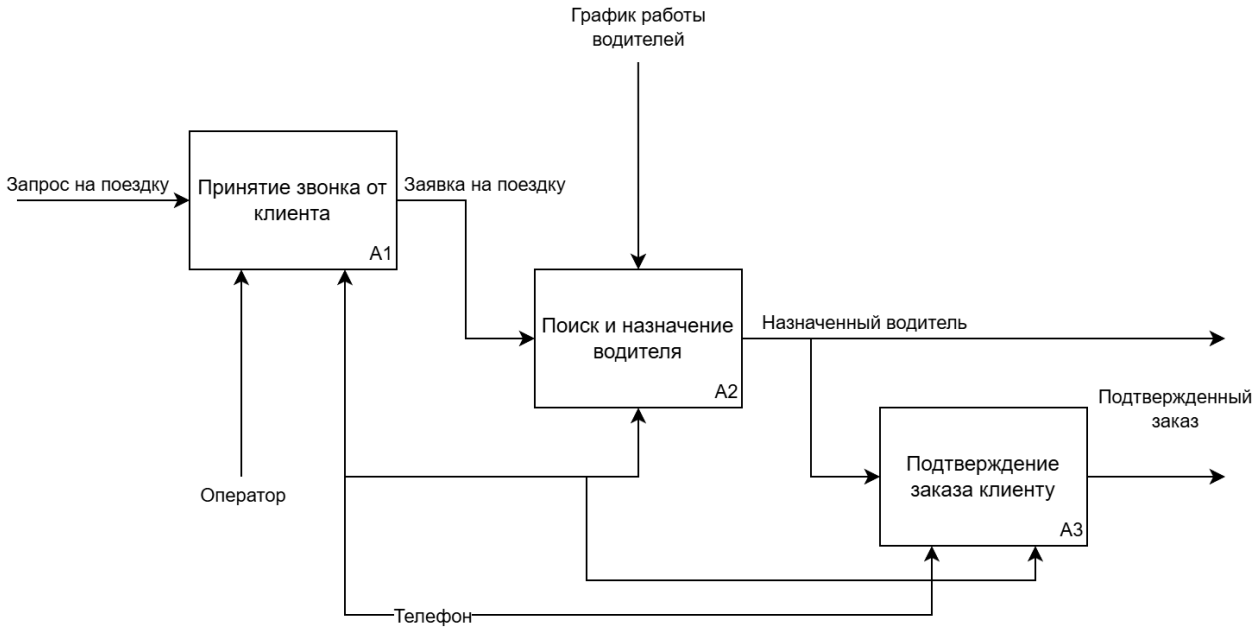

Рисунок 2 – Декомпозиция диаграммы «Как есть»

На входе каждого подпроцесса система использует:

– устные данные клиента (адрес, контактные данные, пожелания к поездке);

– актуальную информацию о доступности водителей (полученную через радио, телефон или электронную систему);

– тарифные параметры (фиксированные расценки, стоимость за километр, дополнительные услуги).

Ограничители сверху включают бизнес–правила сервиса: правила расчёта стоимости, политику конфиденциальности клиентских данных, требования к интеграции с телефонными сетями и ручными системами учёта.

Снизу – возможности инфраструктуры:

– скорость обработки звонков диспетчерами;

– качество связи с водителями;

– ограничения ручного ввода данных.

На выходе формируется:

– устное подтверждение заказа клиенту по телефону;

– информация о назначенном водителе и ориентировочном времени прибытия;

– бумажный или электронный чек, передаваемый клиенту по завершении поездки.

Анализ процесса «как есть» выявил ключевые проблемы:

– ручной ввод данных и зависимость от человеческого фактора приводят к ошибкам и задержкам;

– отсутствие автоматизированного отслеживания водителя вынуждает клиентов повторно звонить для уточнения времени подачи машины;

– рассылка SMS или звонки требуют дополнительных временных затрат, снижая оперативность обслуживания.

Необходимость оптимизации заключается не только в ускорении обработки заявок, но и во внедрении электронных систем учёта для замены ручных операций. Улучшенная классическая система должна обеспечить:

– быстрый поиск водителей через электронную базу;

– автоматизированное формирование заявок и мгновенное оповещение водителей;

– интеграцию с платёжными терминалами для генерации чеков без ручного расчёта.

Методика IDEF0 позволяет смоделировать процесс «как будет», где входящие данные (информация от клиента, статус водителей) и ограничения сохраняются, но механизмы модернизируются:

– ручные журналы заменяются электронными системами учёта;

– радиообмен дополняется автоматическими уведомлениями для водителей;

– обратная связь с клиентом осуществляется через мобильное приложение.

Результат – сокращение времени обработки заказа, минимизация ошибок при передаче данных и повышение прозрачности для клиентов за счёт чёткого информирования о статусе поездки. Это создаёт преимущества перед полностью ручными системами, где задержки и несовершенство коммуникации снижают лояльность клиентов.

Модель «Как будет» представлена на рисунке 3.

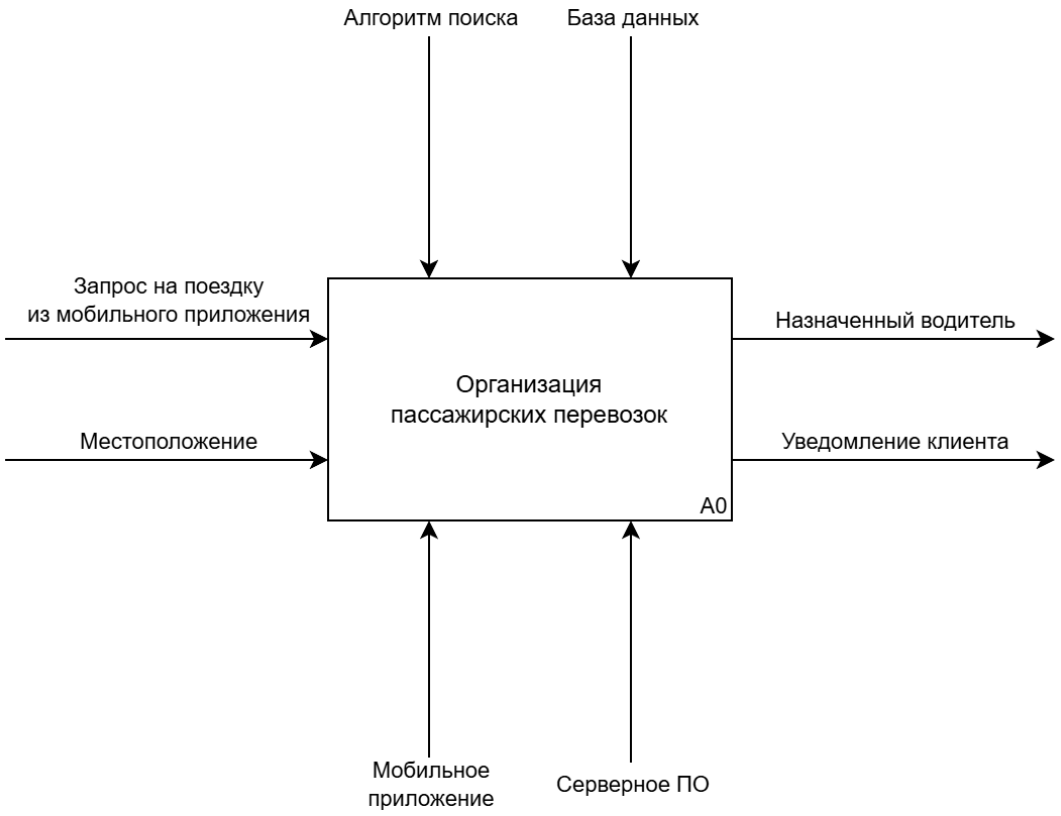

Рисунок 3 – Диаграмма «Как будет»

«Красные стрелки механизма исполнения и выхода отражают направления модернизации процесса. Разработка изменений ведётся с сохранением ключевых участников – водителей.

В результате на этапе декомпозиции (рисунок 4) корректируется способ обработки заказов:

Внедряется электронная система учёта заявок вместо ручных журналов.

Данные о клиентах и водителях структурируются в БД, что упрощает поиск и анализ» [5].

На выходе изменяется доступ к информации.

Данные группируются в логические блоки:

– история заказов (дата, время, статус);
– параметры поездок (маршрут, стоимость);
– контакты клиентов и водителей.

Это упрощает формирование отчётов, уточнение деталей прошлых заказов и прогнозирование загруженности.

Итог – переход от полностью ручного учёта к автоматизированной обработке данных снижает время обработки заявок, минимизирует ошибки при передаче информации и делает историю заказов доступной для оперативного анализа.

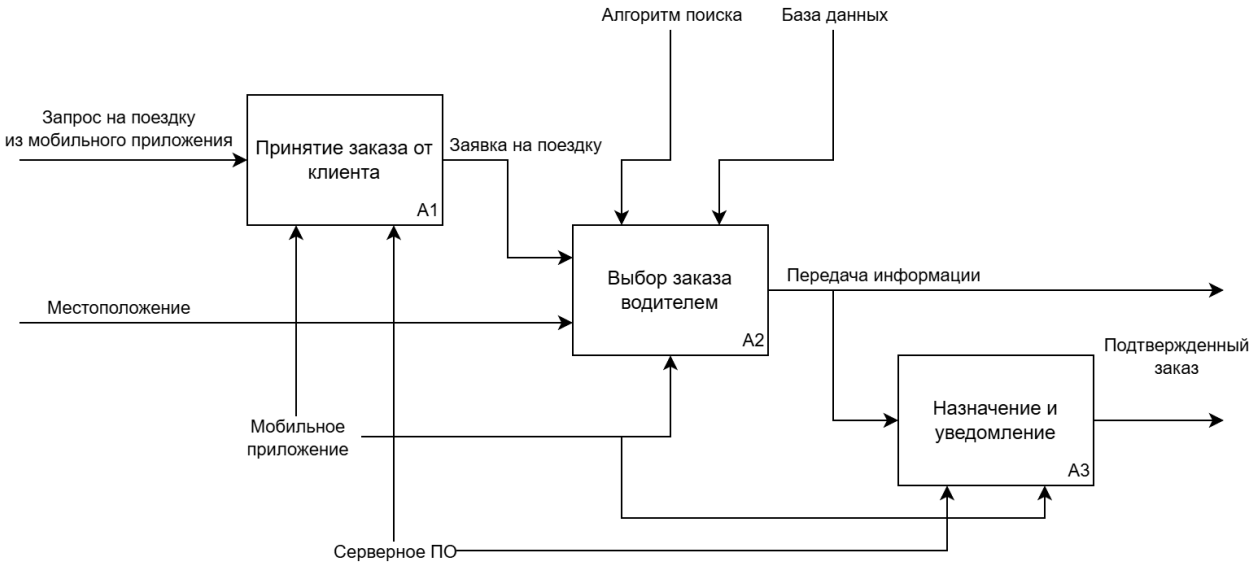

Рисунок 4 – Декомпозиция диаграммы «Как будет»

Декомпозиция основного процесса заказа такси разделяется на три взаимосвязанных подпроцесса:

– указание пункта отправки и назначения – пользователь через мобильное приложение указывает начальную и конечную точки заказа;

– самостоятельный выбор заказа водителем – водители получают доступ к списку активных заказов и выбирают подходящие на основе критерий своего местоположения и предпочтений.

Оформление и завершение заказа – после подтверждения выбора заказа, система:

– фиксирует назначенного водителя,

– отслеживает выполнение поездки,

– позволяет оценить поездку после выполнения заказа.

На входе каждого подпроцесса система использует:

– геоданные пунктов отправки и назначения (координаты, адреса);

– информацию о доступных заказах, отображаемую водителям.

На выходе формируется:

– мгновенное уведомление пользователя о принятии заказа,

– пункты назначения и отправки.

Анализ процесса «как есть» выявил ключевую проблему: зависимость от ручного взаимодействия заказчиков через диспетчеров службы такси, что может сильно замедлять обработку заказов и снижает оперативность обратной связи с заказчиком. Отсутствие отслеживания статуса заказа в реальном времени ухудшают взаимодействия клиентов и водителей.

Необходимость оптимизации заключается в цифровизации в виде автоматизированной платформы, где она должна обеспечивать:

– интерактивное и удобное указание точек отправки и назначения;

– мгновенные уведомления о изменениях в статусе заказа;

– сохранение истории о поездках и их оценке.

Методика IDEF0 демонстрирует процесс «Как будет», сохраняя входные данные и ограничения модели «Как есть», но модернизируя некоторые механизмы. Это позволяет наглядно показать преимущества цифровизации системы:

– водители самостоятельно управляют выбором заказов, что снижает нагрузку на координацию через диспетчеров;

– заказчики получают прозрачный процесс с автоматизированным и быстрым оформлением, выполнением, отслеживанием, оценке и просмотр истории заказов.

Итог – устранение задержек, характерных при работе со звонками и/или SMS–коммуникации, ускоряет скорость обработки заказов и удобства пользователей за счёт чёткого и быстрого контроля каждого этапа поездки.

2.2 Моделирование технологии реализации

«Этапы реализации программного обеспечения для мобильного приложения такси могут быть представлены в виде линейно связанных процессов, наглядно описываемых с помощью нотации BPMN (Business Process Model And Notation)» [5]. В этой модели участники разработки (Product Owner, разработчики и тестировщики) располагаются в отдельных пулах и дорожках (pools & lanes), что позволяет чётко разграничить зоны их ответственности при реализации каждого этапа (рисунок 5).

Процесс начинается с заявки на разработку (Start Event), после чего следует анализ требований и отбор функциональных модулей: регистрация пользователей, выбор адресов отправки и назначения, оформление заказа пользователем и получение заказа водителем, а также завершение и оценка заказа.

На этой стадии в пуле Product Owner работают над уточнением требований, а в пуле разработчиков – над первичной декомпозицией задач.

Далее создаётся прототип экранных форм, основанный на техническом задании и принципах UX/UI. В пуле дизайнеров генерируются wireframes и mockups, которые затем передаются разработчикам для реализации в Avalonia UI.

После утверждения макетов заказчиком через Gateway: Approval процесс переходит к этапу физической разработки, где код пишут с учётом SOLID-принципов и паттерна MVVM.

Завершающие шаги – модульное и интеграционное тестирование и CI/CD-деплой. При успешном прохождении тестов процесс заканчивается релизом. Все потоки и элементы соединены последовательными стрелками, чётко отображающими порядок выполнения задач и обмен данными между участниками [8].

Для хранения данных используется реляционная модель. Данные организованы в виде таблиц (relations), где каждая таблица соответствует сущности бизнес-домена: User, Driver, Order, и Rating:

– строки (classes) представляют отдельные записи (например, одна поездка);

– столбцы таблиц (например, address_from, address_to) определяют параметры поездки;

– первичные ключи (primary keys) гарантируют уникальность записей (order_uuid, user_uuid) и не могут содержать NULL;

– внешние ключи (foreign keys) обеспечивают связи между таблицами (например, driver_uuid в таблице Order).

«Проектирование базы данных проходит три этапа:

– концептуальная модель – определение основных сущностей и их связей, например, сущность Order связывает User и Driver через отношения «один-ко-многим»;

– логическая модель – детализация атрибутов, типов данных и ограничений (NOT NULL, UNIQUE), без привязки к конкретной СУБД;

– физическая модель – реализация структуры» [7] в выбранной СУБД (например, SQLite), создание индексов и оптимизация запросов для уменьшения времени отклика приложения.

Для удобства документирования всех моделей и обмена ими между командой используется UML (диаграммы классов и сущностей), а также ER-диаграммы (Entity-Relationship) – универсальный язык моделирования данных. Это обеспечивает единую нотацию и прозрачность архитектуры как для разработчиков, так и для заказчика.

Таким образом, сочетание BPMN для описания процессов разработки и реляционно-ER/UML-моделей для проектирования базы данных создаёт чёткую, модульную и легко поддерживаемую структуру реализации мобильного приложения для заказа такси.

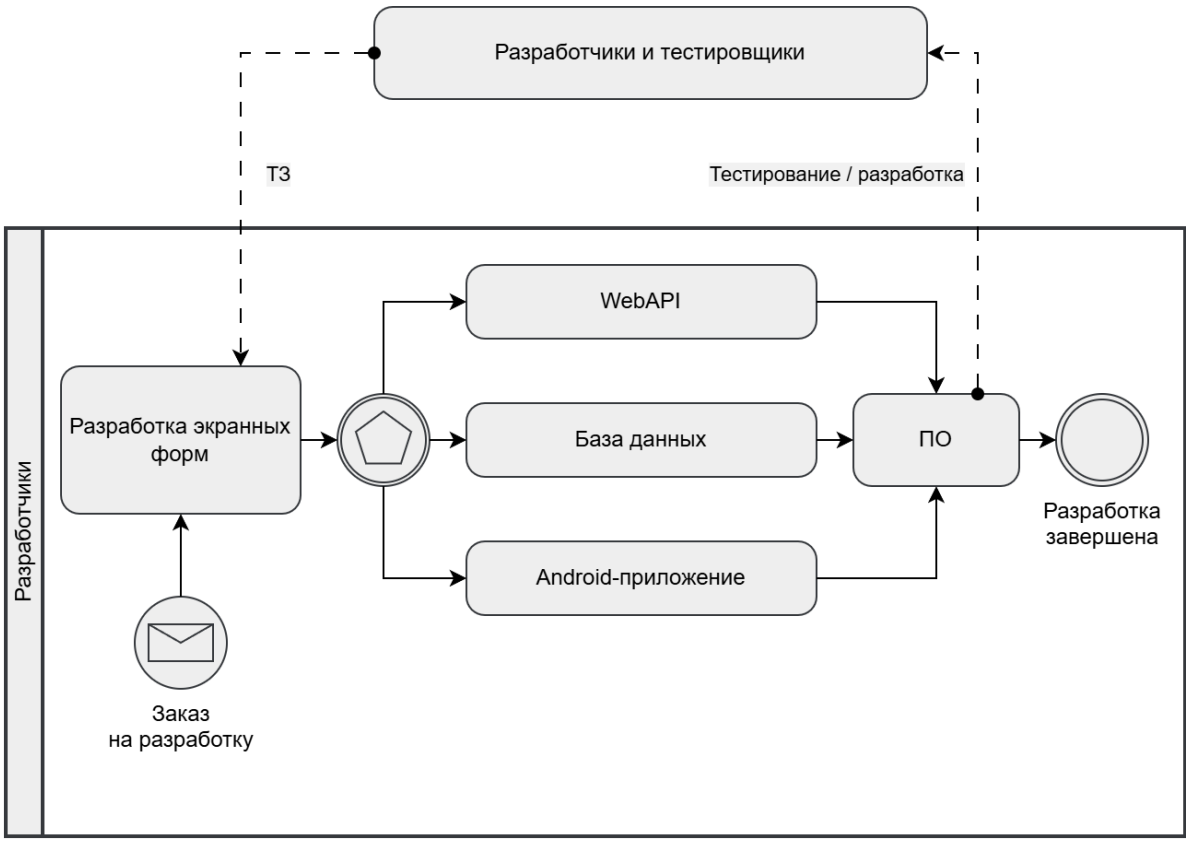

Рисунок 5 – Иллюстрация процесса разработки ПО в BPMN

«Концептуальная модель базы данных для мобильного приложения заказа такси отражает основные сущности и связи между ними, задавая структуру будущих таблиц [7]. На этапе логического проектирования ER-модель дополняется деталями – к таблицам добавляются связи и уточняется типизация полей [11].

Существует три основных типа связей между сущностями:

– «один-к-одному», когда одной записи в таблице «Пользователь» соответствует ровно одна запись в таблице «Профиль»;

– «один-ко-многим», например, одному пользователю может соответствовать несколько записей в таблице «Заказы» (каждый новый заказ связывается с тем же пользователем).

На логической модели (рисунок 6) для приложения такси все поля сущностей дополнены типами данных:

– text для идентификаторов,

– text для отметок времени вызова и завершения поездки,

– text для строковых полей (адреса, комментарии),

– real для стоимости и т.д.

Такой подход обеспечивает однозначное соответствие структуры базы данных требованиям мобильного приложения по приёму и обработке заказов.

Рисунок 6 – Логическая модель базы данных «TomDriveDb»

2.3 Описание технологии реализации

Для разработки клиентской части мобильного приложения такси используется кроссплатформенный фреймворк Avalonia UI. В основе интерфейса лежит модель MVVM (Model–View–ViewModel), которая обеспечивает чёткое разделение представления и логики и упрощает тестирование и поддержку кода.

Ключевой компонент Avalonia UI – система визуальных контролов и макетов, реализованная через дерево элементов (Visual Tree). С помощью контейнеров StackPanel, Grid, DockPanel и других макетов можно гибко выстраивать адаптивные интерфейсы, автоматически подстраивающиеся под разные размеры и ориентации экранов мобильных устройств.

В приложении для заказа такси активно используется компонент Avalonia WebView для отображения интерактивной карты. WebView встраивает браузерный движок прямо в окно Avalonia, что позволяет загружать веб–версию картографического сервиса, для определения адресов

пунктов отправки и назначения через двунаправленный обмен сообщениями между JavaScript и .NET [10, 13, 14].

Работа с темами и стилями реализуется через XAML–подобную разметку и динамическую систему ресурсов (Resource Dictionaries). Это позволяет централизованно задавать цвета, шрифты и отступы, а при необходимости – переключать тему (светлую/тёмную) без правки кода [21].

Навигация между экранами построена на паттерне NavigationView и собственном навигационном сервисе. Пользователь переходит от экрана выбора точки посадки к экрану подтверждения заказа и далее к истории поездок, причём состояние каждого экрана хранится в соответствующей ViewModel.

Таким образом, использование Avalonia UI обеспечивает:

– один и тот же код интерфейса под Android, iOS и десктопные платформы;

– гибкость вёрстки и адаптивность под любые экраны;

– плавную интеграцию с веб–компонентами (WebView) для карт;

– чистую архитектуру MVVM, упрощающую поддержку и расширение функционала.

Данный компонент на C# изначально разрабатывался для кроссплатформенного фреймворка Avalonia UI, где визуальные элементы определяются с помощью XAML. Вместо встроенного браузера используется чистый UI-контрол, а взаимодействие с картографическими сервисами и другими внешними ресурсами реализуется через Avalonia WebView. Для визуального проектирования интерфейса в среде JetBrains Rider применяется плагин Avalonia XAML Plugin, который позволяет в графическом режиме настраивать размеры, привязки и стили компонентов [21].

В приложении для заказа такси компонент WebView предоставляет возможность встраивать интерактивную карту в полноэкранном режиме. URL сервиса карт загружается программно, а точка входа задаётся из кода ViewModel. Благодаря двунаправленному взаимодействию JavaScript ↔ C#

приложение может обращаться как к локальным ресурсам (иконки, стили), так и к REST API сервера, передавая координаты пользователя и получая маршруты [16].

На рисунке 7 представлена архитектура мобильного приложения, где взаимодействия между клиентским интерфейсом на Avalonia UI и серверной частью происходит через механизмы HTTP–запросов в C# через платформу ASP.NET [20]. Использование Avalonia UI и JetBrains Rider с плагином XAML обеспечивает единую кодовую базу при разработке интерфейсов для любой платформы, что, позволяет сочетать преимущества разработки кроссплатформенных приложений в одном решении.

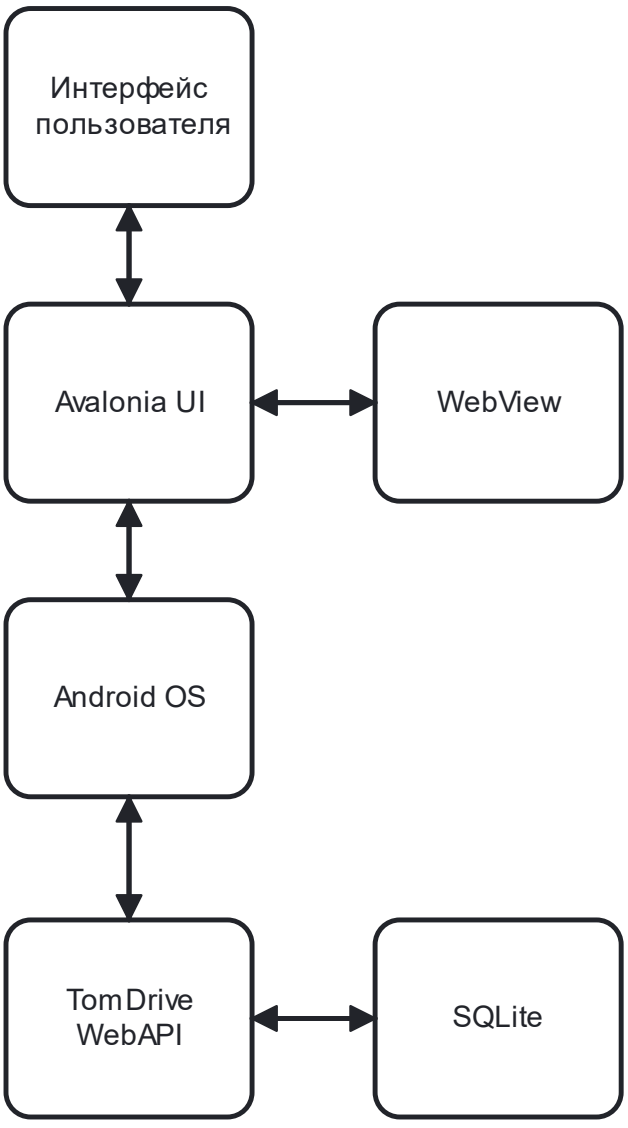

Рисунок 7 – Архитектура мобильного приложения

Выводы по второй главе

1. Проведено моделирование бизнес-процессов заказа такси и этапов разработки ПО.

2. Подробно описана архитектура мобильного приложения на базе Avalonia UI, включая компоненты для работы с картами и уведомлениями.

3. Использование паттернов MVVM, HTTP-клиента на C# и визуальных инструментов JetBrains Rider с Avalonia XAML Plugin обеспечивает гибкость, адаптивность и единообразие интерфейса на всех поддерживаемых платформах.

Глава 3 Физическая реализация программного обеспечения

3.1 Разработка интерфейса

«В качестве среды разработки программного обеспечения было выбрано свободно распространяемая IDE JetBrains Rider, который позволяет создавать рабочий проект, включающий все необходимые библиотеки: Avalonia UI и WebView для создания интерфейса» (рисунок 8).

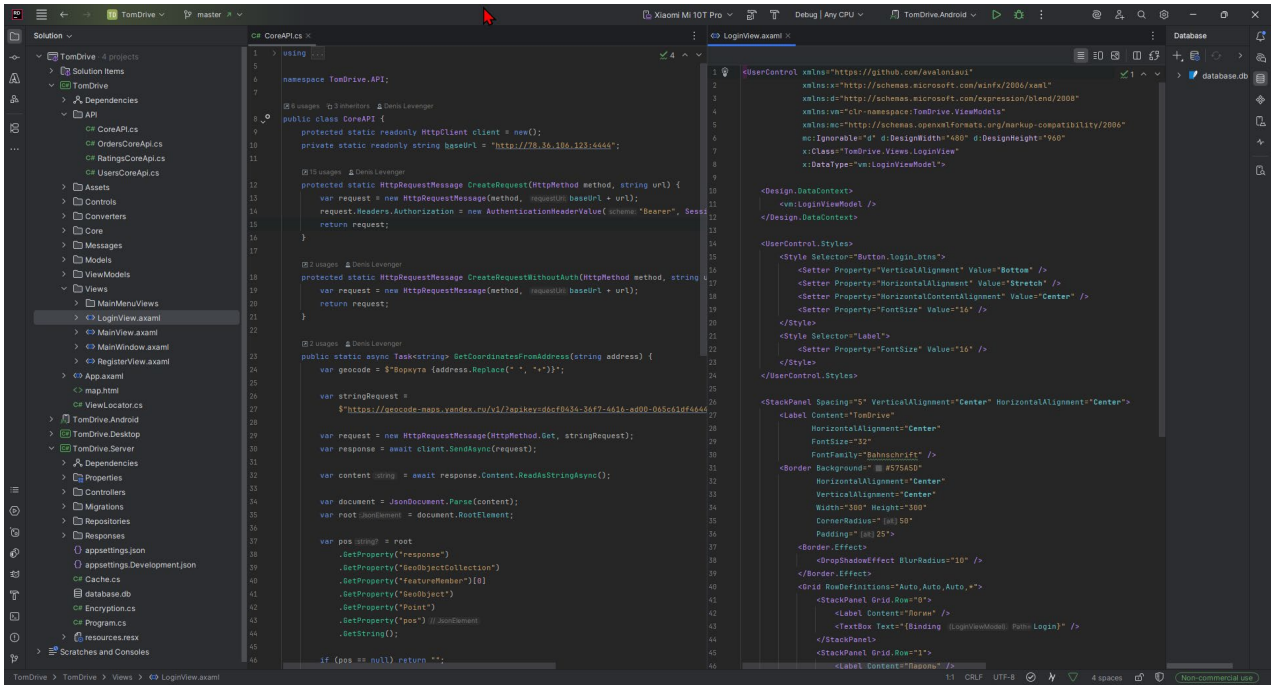

Рисунок 8 – Интерфейс и рабочее пространство среды JetBrains Rider

Разработка графических окон интерфейсов производится с помощью языка разметки XAML [24, 25, 26]:

Код разметки окна авторизации в приложении (приложение А).

Интерфейс окна авторизации в приложении представлен на рисунке 9.

Рисунок 9 – Окно авторизации приложения

На рисунке 10 представлено основное окно приложения с боковым меню разделов. Разметка данного меню представлена в приложении Б.

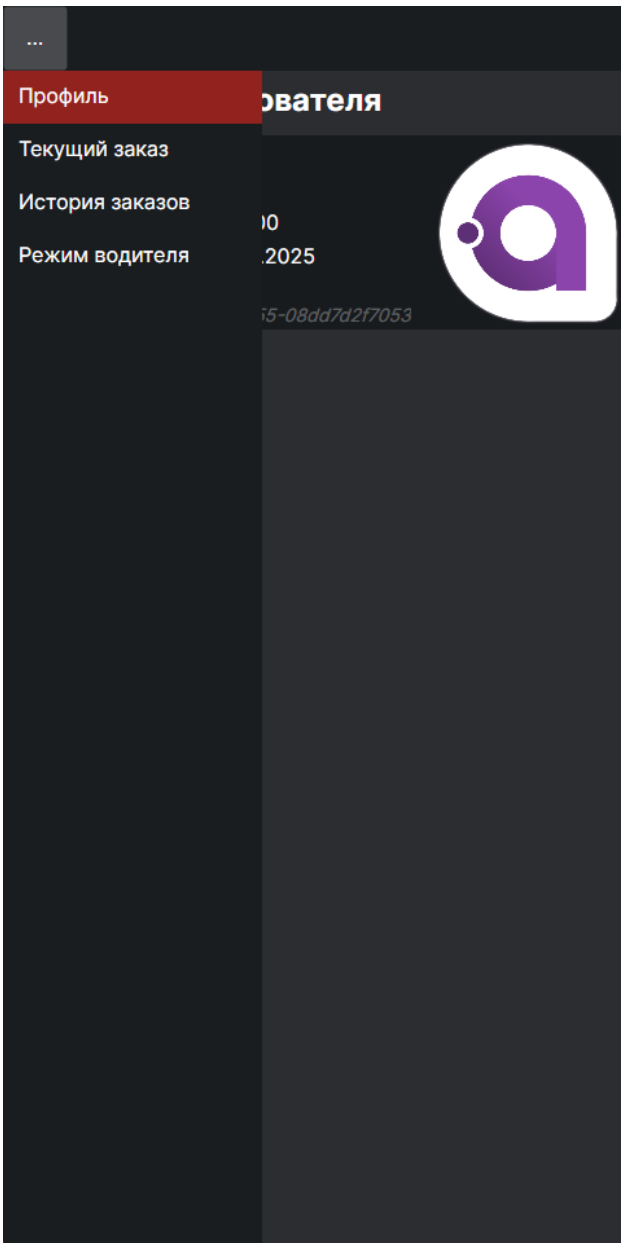

Рисунок 10 – Основное окно приложения с открытым меню разделов

В качестве сервиса для определения координат по адресу используется Яндекс.Геокодер (рисунок 11), а картографический сервис Яндекс.MapKit (рисунок 12). Для использования данных API требуется зарегистрироваться в личном кабинете разработчика Яндекс и получить соответствующие ключи для доступа к API [22, 23].

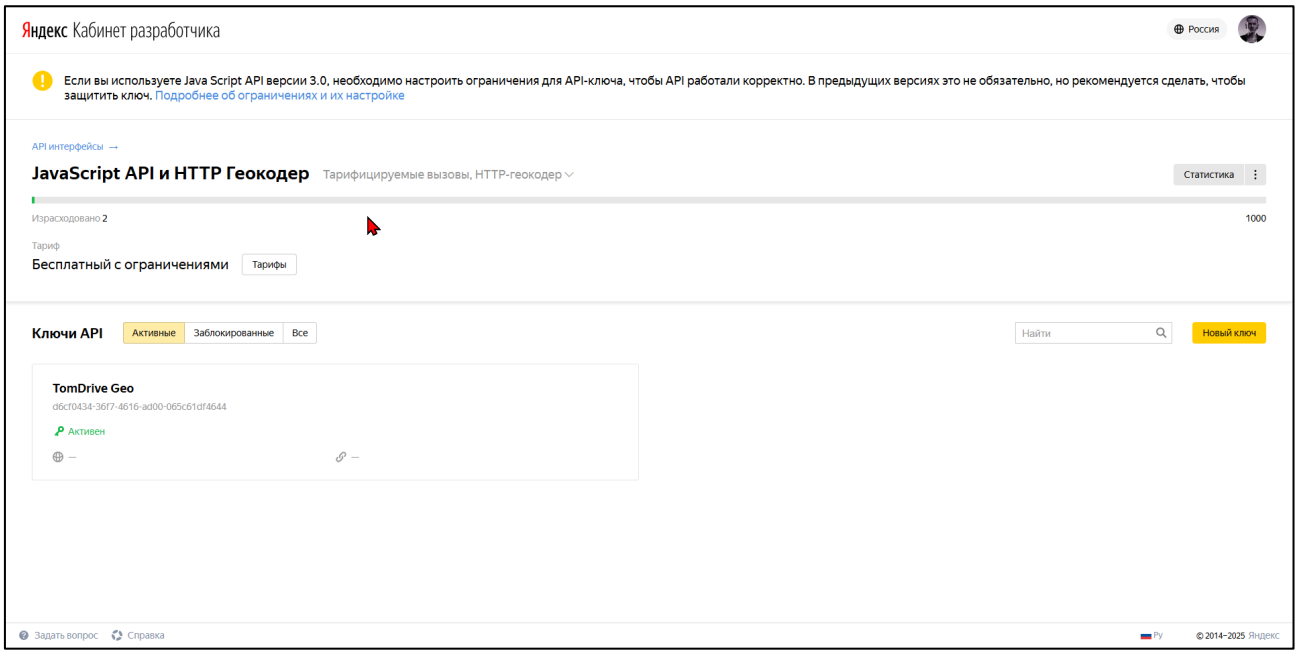

Рисунок 11 – Личный кабинет разработчика, API Геокодер

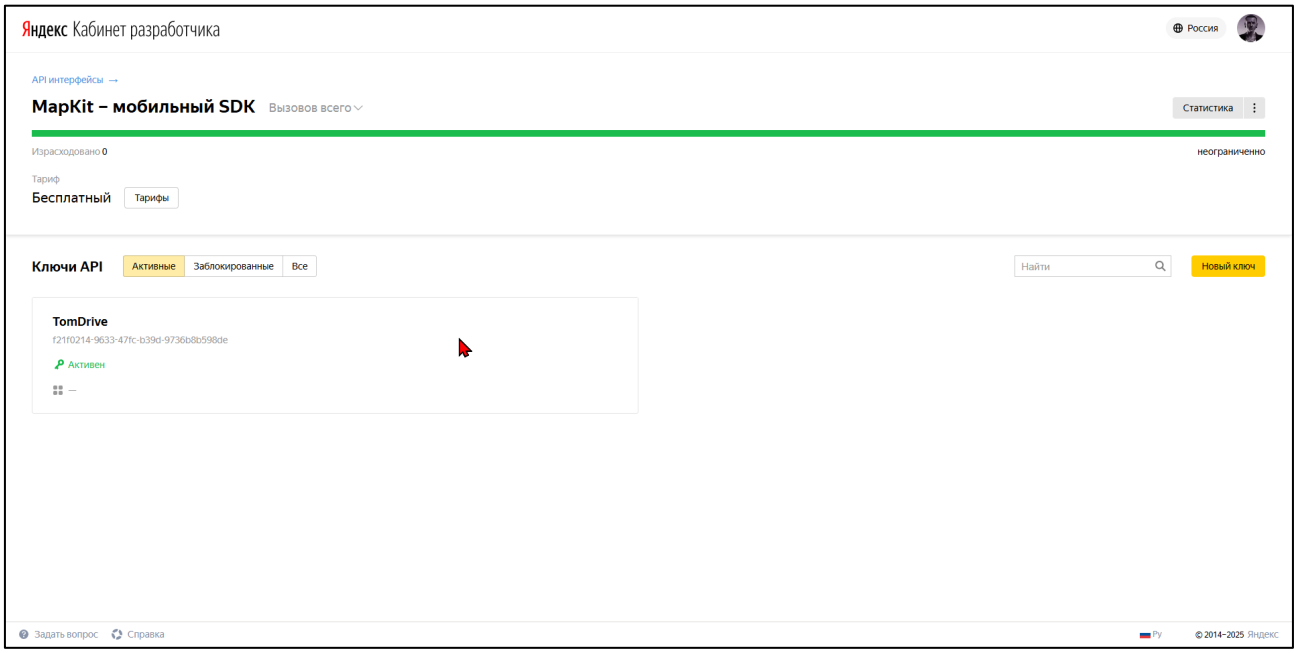

Рисунок 12 – Личный кабинет разработчика, API MapKit

Отображение карты с пунктами отправления и прибытия в приложении реализована при помощи компонента WebView, HTML и JavaScript (приложение В).

На рисунке 13 представлен интерфейс окна текущего заказа. Код модели представления окна описан в приложении Г.

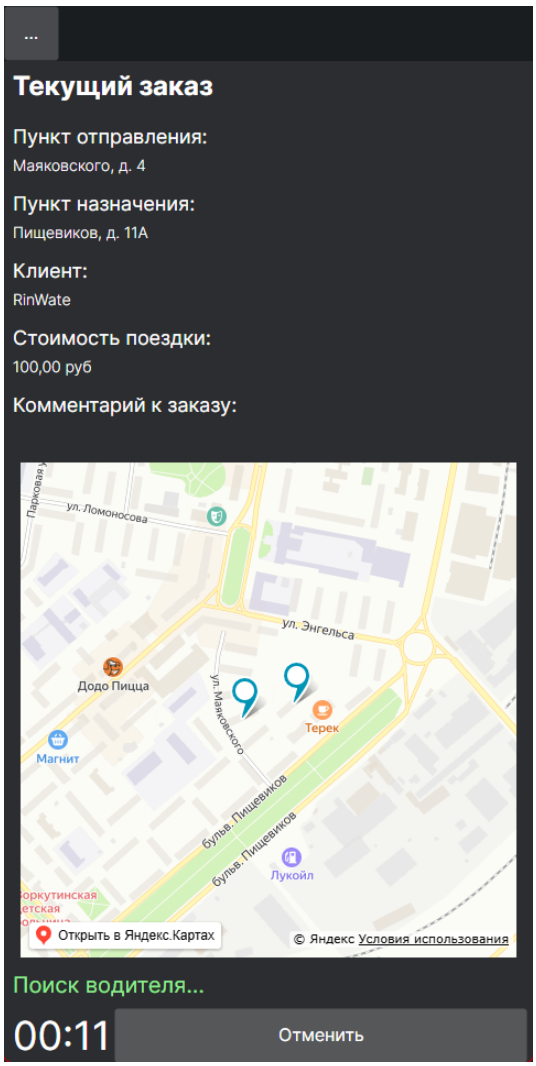

Рисунок 13 – Интерфейс окна текущего заказа

3.2 Разработка базы данных

Разработка таблиц базы данных в SQLite осуществляется при помощи программы SQLite Studio. Создание таблиц происходит с помощью SQL запросов (рисунок 14) [18]. Структура разработанной базы данных представлена на рисунке 15.

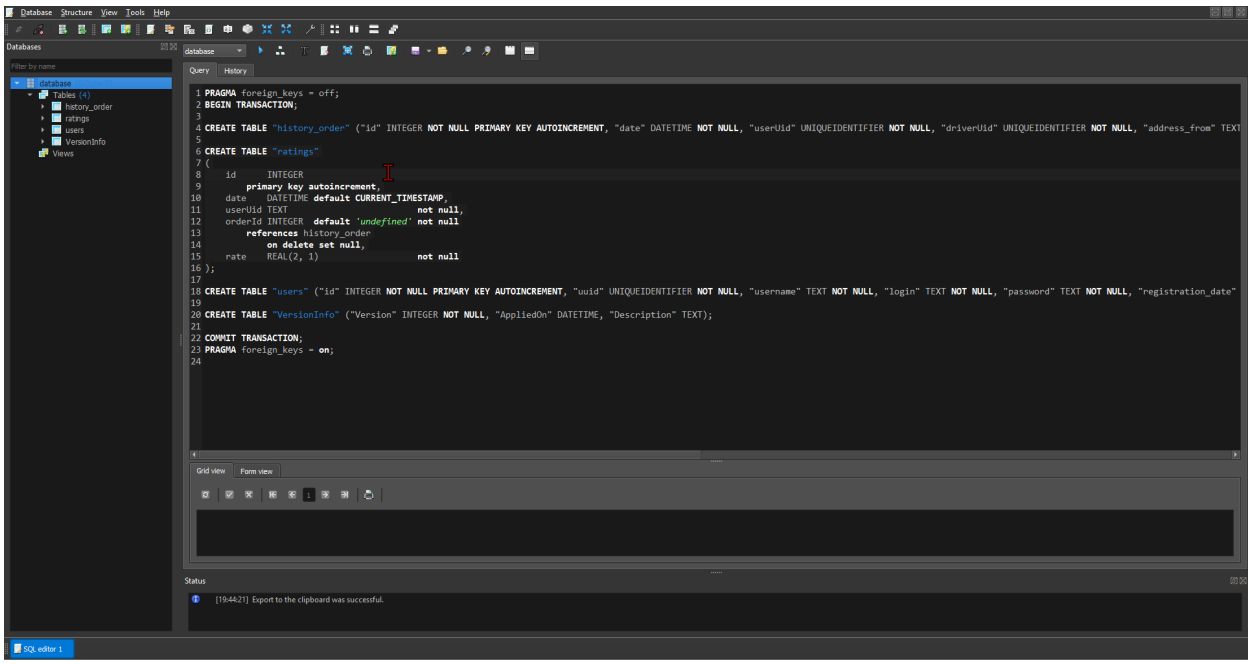

Рисунок 14 – Создание таблиц в базе данных «TomDriveDb» в программе SQLite Studio

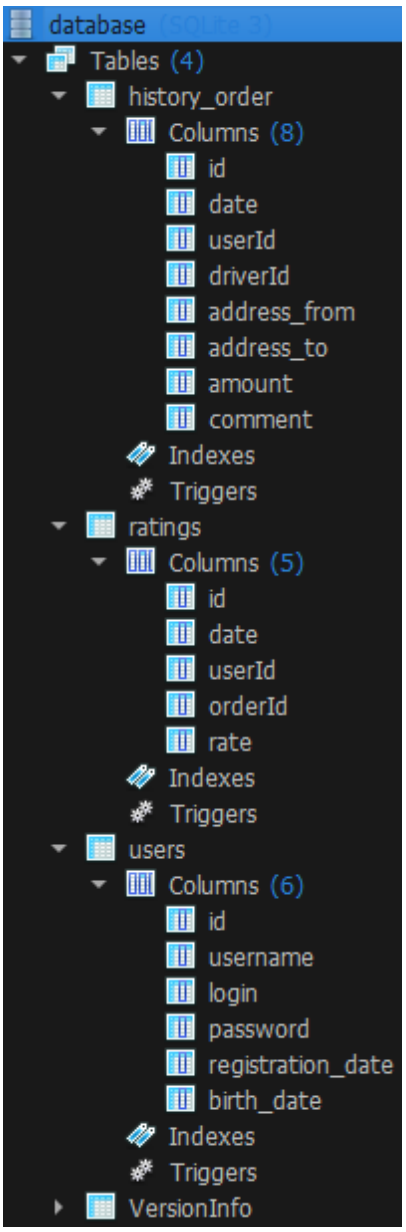

Рисунок 15 – Структура базы данных в SQLite Studio

«Таблицы «users», «history_order» и «ratings» создаются с указанием уникального первичного ключа таблицы и внешних ключей от линковочных таблиц. Генерация производится в соответствии с физической моделью базы данных» [7] (рисунок 16).

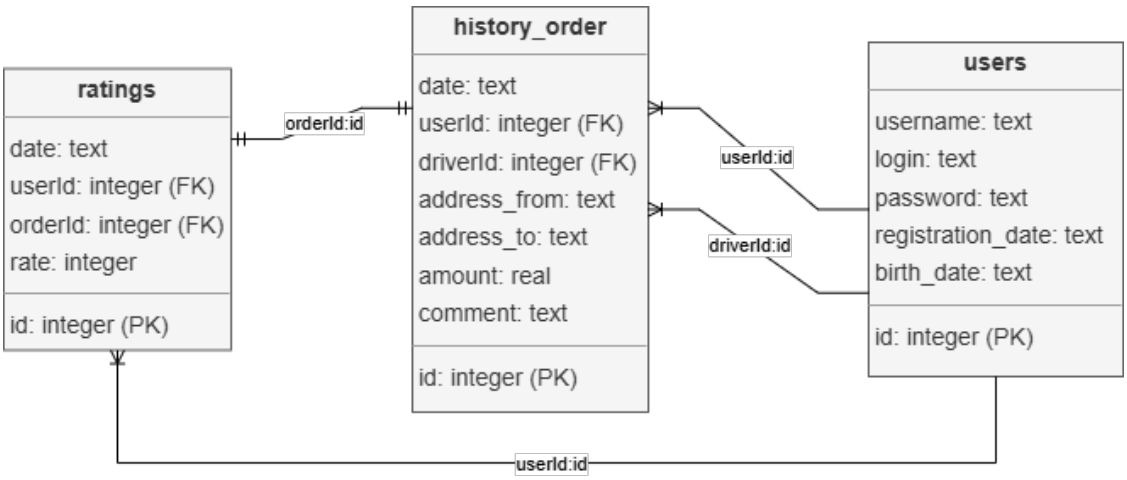

Рисунок 16 – Физическая модель базы данных «TomDriveDb»

Линковочные таблицы создаются с указанием внешних ключей смежных таблицы. Пример создания промежуточной таблицы при помощи запроса SQL: CREATE TABLE Ratings (id INTEGER PRIMARY KEY, date DATETIME NOT NULL, userUid UNIQUEIDENTIFIER).

Следующим этапом работы с базой данных является установка связи с приложением на ASP.NET при помощи следующего кода [19]:

```
await using var connection = new SqliteConnection("Data Source=tomdrivedb");
```

Так как база данных SQLite является файловой, то для подключения используется относительный или абсолютный путь до файла базы данных.

3.3 Разработка мобильного приложения

«Разработка кроссплатформенного мобильного приложения для заказа такси выполнялась с использованием фреймворка Avalonia UI в среде .NET. Описание функций элемента WebView, отвечающего за отображение интерактивной карты, располагается в файле CoreAPI.cs» [9] (приложение Д). Здесь же настраиваются компоненты окна – задаётся свойство Source для загрузки URL картографического сервиса, а также реализуется вызов методов JavaScript для передачи координат и установки точек маршрута:

В файле «AndroidManifest.xml» задаются разрешения приложения (рисунок 17). Так как приложение использует выход в сеть Интернет,

требуется указать правила в файле манифеста: <uses–permission android:name=”android.permission.INTERNET”/>. Так же дополнительно назначается название приложения и ссылка на логотип:

android:icon=”drawable/Icon”

android:label=”TomDrive”

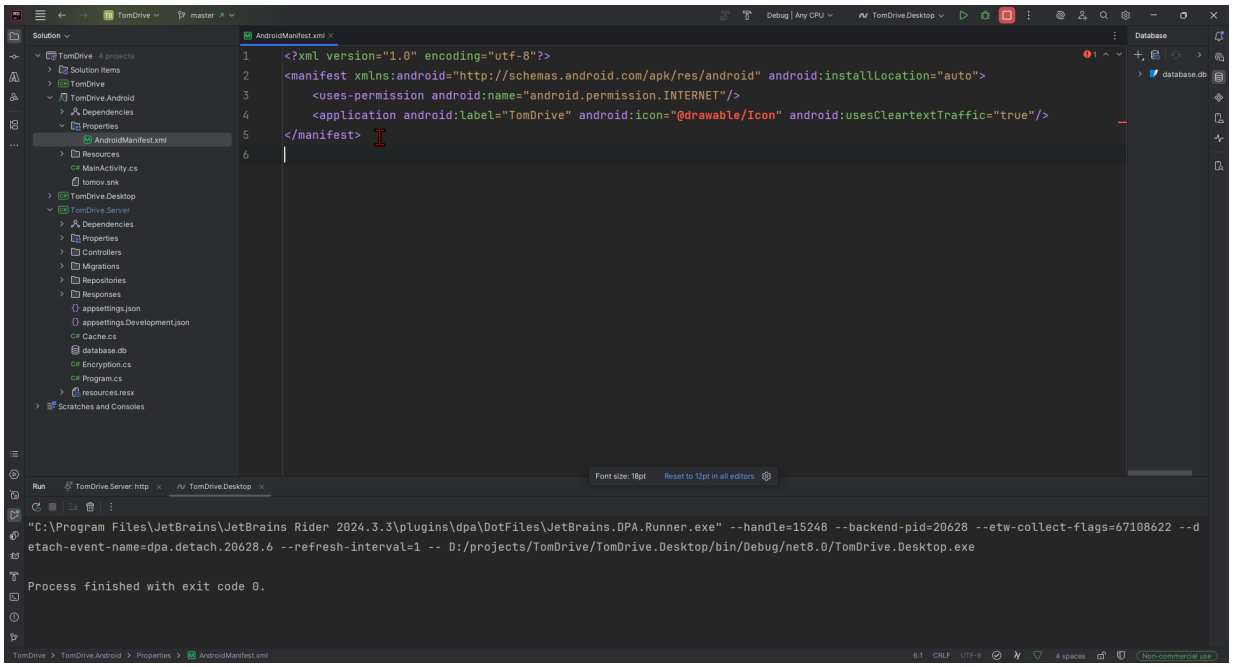

Рисунок 17 – Файл «AndroidManifest.xml» в среде JetBrains Rider

Тестирование приложения проводилось на устройстве: Xiaomi Mi 10T Pro и Android эмуляторе. Это позволило проверить результат работы компонента WebView Android и их работу в операционных системах Android 12 и 14 соответственно.

3.4 Тестирование приложения

Целью тестирования является обеспечение корректности и надежности работы серверной части (ASP.NET Web API) мобильного приложения для заказа такси. Основное внимание уделяется проверке правильности обработки HTTP–запросов и выполнения SQL–запросов к базе данных, а также корректная работа клиентской части мобильного приложения.

Произведено тестирование приложений с помощью автоматических unit–тестов и ручного тестирования.

Unit–тестами были покрыты критические места в коде, в данном случае это HTTP и SQL запросы. На рисунке 18 предоставлены успешно пройденные автоматические unit–тесты.

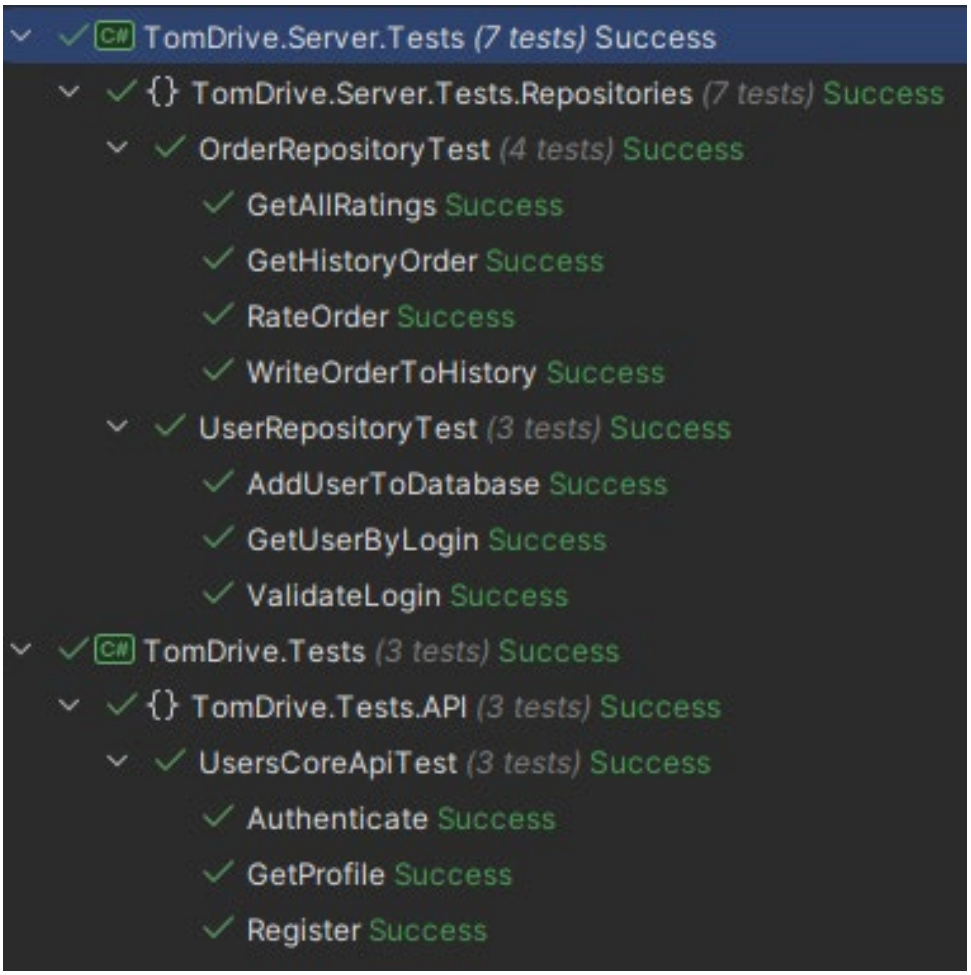

Рисунок 18 – Успешность прохождения автоматических Unit–тестов

Посредством ручного тестирования была проверена полная работа клиентской части (таблица 3). Запуск приложения, регистрация (рисунок 19), авторизация пользователя и водителя (рисунок 20), оформление заказа (рисунок 21), окно текущего заказа (рисунок 22), принятие заказа водителем (рисунок 23), завершение и оценка поездки (рисунок 24), просмотр истории заказов (рисунок 25).

Таблица 3 – Ручное тестирования работы приложения

Тестируемый сценарий	Ожидаемый результат	Статус
Запуск приложения, регистрация	Окно регистрации корректно отрабатывает правильность ввода данных регистрации	Пройден
Авторизация пользователя и водителя	При правильности ввода логина и пароля авторизация пользователя успешна	Пройден
Оформление заказа	При частичном вводе адреса отправления и прибытия, формируемый заказ корректно назначает реальные адреса на местности.	Пройден
Окно текущего	Корректное отображение на карте пунктов отправления и прибытия и корректное отображение данных о заказе.	Пройден
Принятие заказа водителем	Правильное отображение новых и доступных заказов у водителя.	Пройден
Завершение и оценка поездки	Правильность отображение данных о завершенном заказе и доступность выбора его оценки.	Пройден
Просмотр истории заказов	Корректное отображение истории завершенных заказов и информация о нем у пользователя.	Пройден

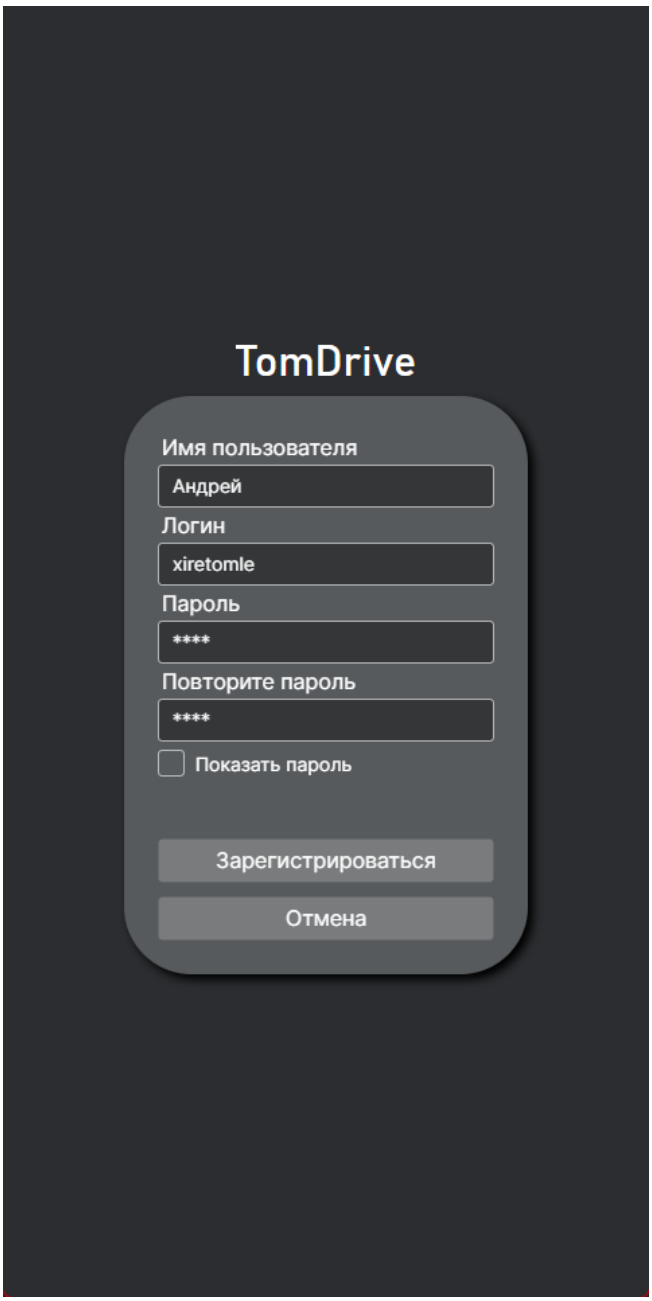

Рисунок 19 – Регистрация пользователя

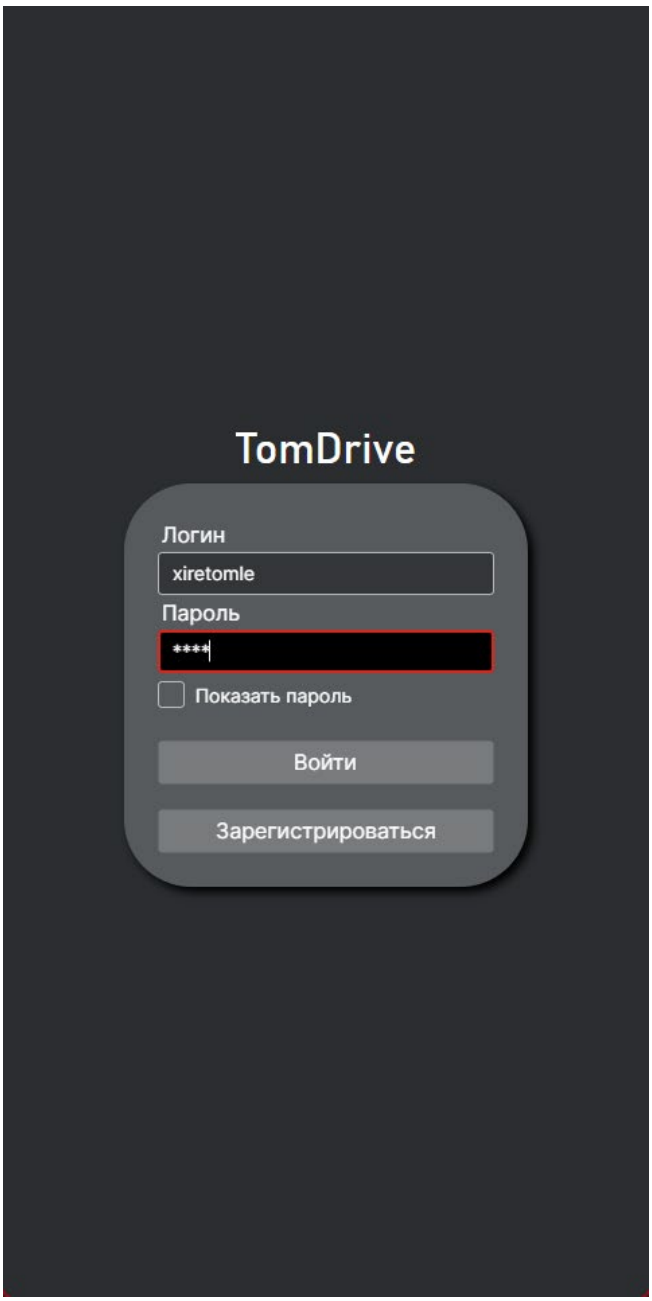

Рисунок 20 – Авторизация пользователя

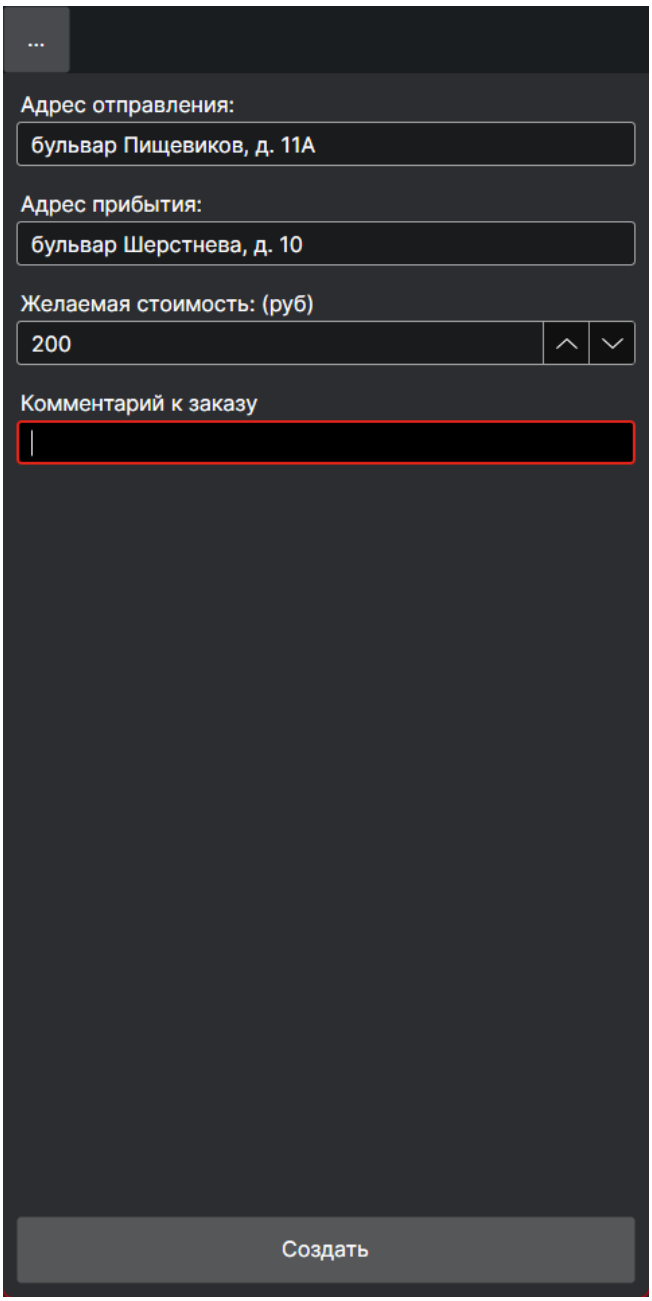

Рисунок 21 – Оформление заказа

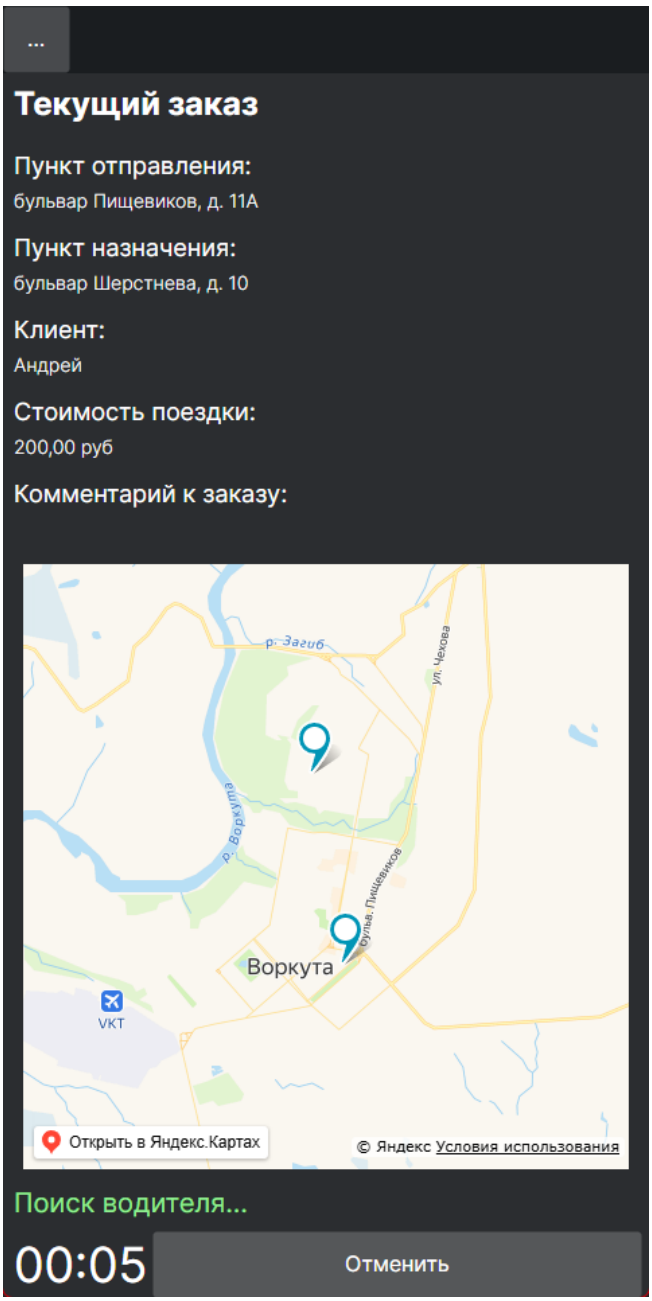

Рисунок 22 – Окно текущего заказа

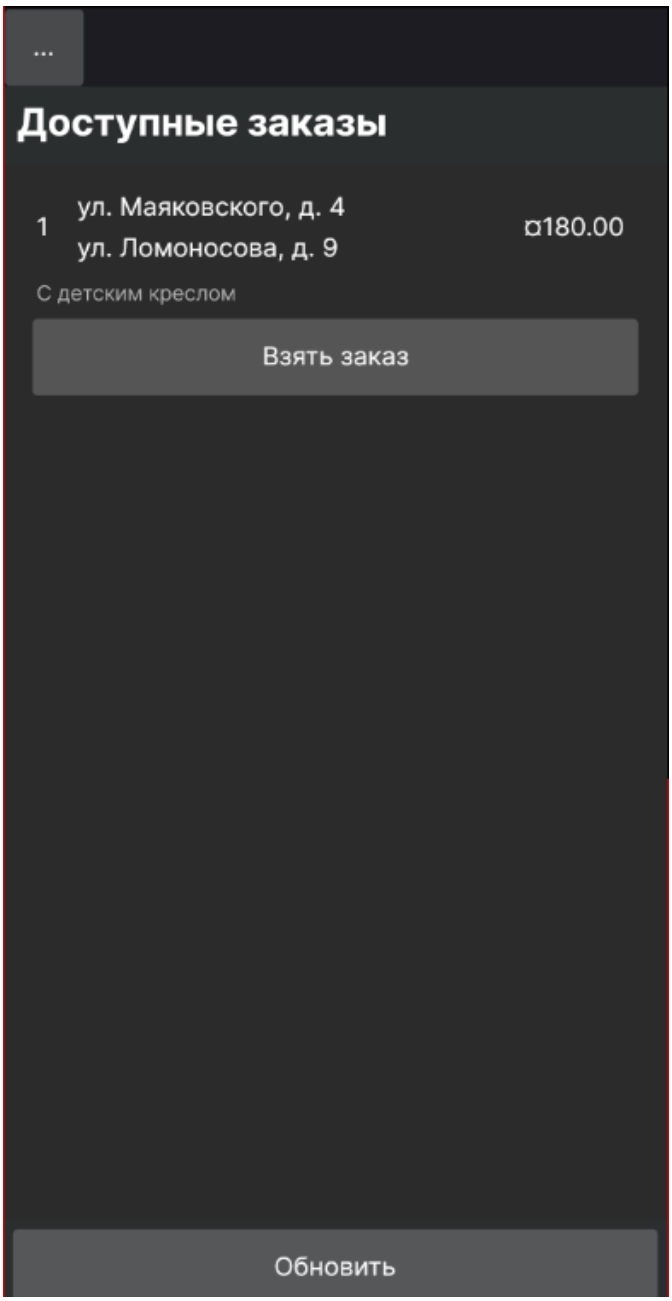

Рисунок 23 – Принятие заказа водителем

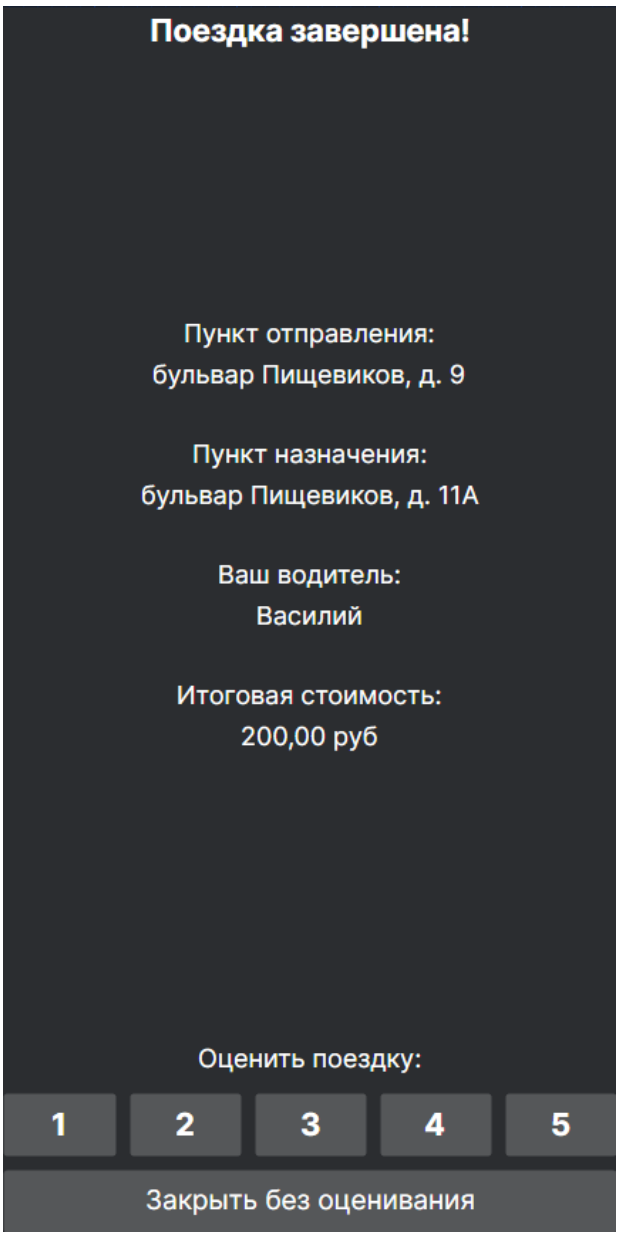

Рисунок 24– Завершение и оценка поездки

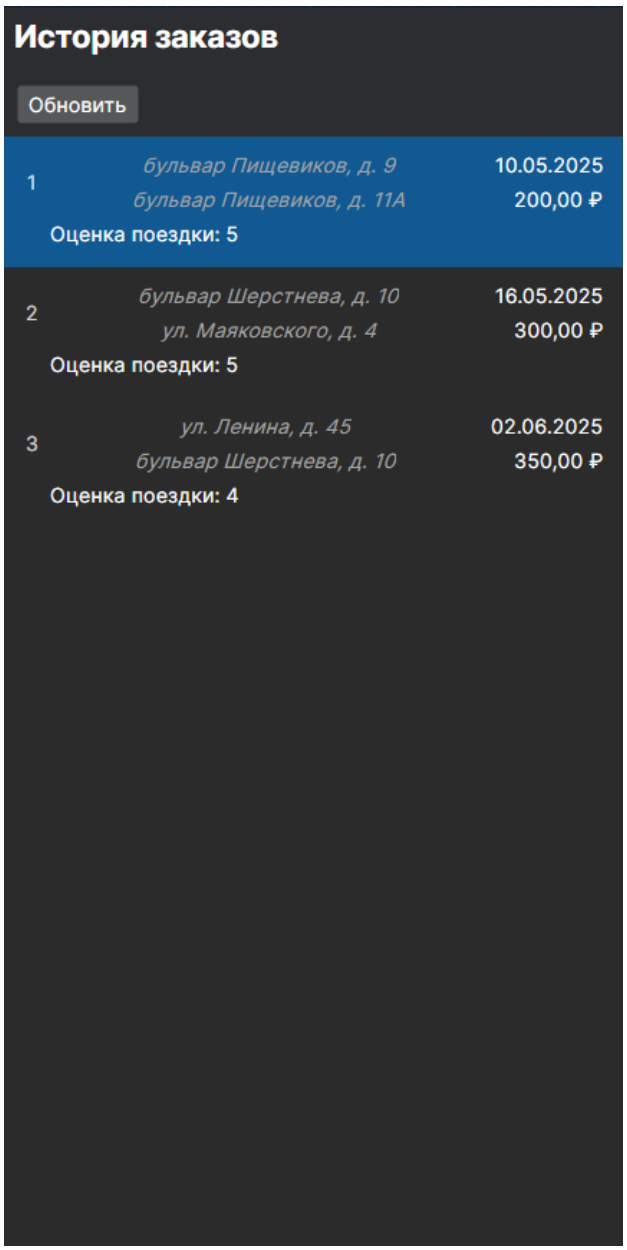

Рисунок 25 – Просмотр истории заказов

Выводы по третьей главе

В данной главе отражена реализация всех этапов работы программного обеспечения – мобильного приложения заказа такси и серверная часть с базой данных.

В ходе тестирования было проверено следующее:

- корректная загрузка и запуск приложения;
- правильность отправки и обработки HTTP запросов;
- надежность выполнения SQL запросов к базе данных;
- правильность навигации и переходов между экранами (регистрация, авторизация, оформление заказа, принятие заказа, выставление оценки и история поездок).

Заключение

В рамках выпускной квалификационной работы по теме «Разработка мобильного приложения для заказа такси» были изучены такие технологии как:

- язык программирования C# «для реализации программного обеспечения;
- платформа ASP.NET для создания Web-API;
- фреймворк Avalonia UI для разработки универсального графического интерфейса, подходящего под различные платформы;
- язык разметки AXAML для создания графического интерфейса в Avalonia UI;
- база данных SQLite» [1] и язык запросов SQL для возможности хранения и валидации информации;
- Яндекс.Геокодер и Яндекс.MapKit – для определения координат по адресам и предоставления картографических данных;
- язык разметки HTML и язык программирования JavaScript для взаимодействия с API Яндекс.Геокодер и Яндекс.MapKit.

В ходе выполнения выпускной квалификационной работы было разработано мобильное и серверное приложение, обеспечивающее доступ пользователям к оформлению заказа такси, просмотру истории и оценок заказов. Доступ водителей к активным заказам для их выполнения. Хранение информации о пользователях, о истории завершенных заказов и их оценках в базе данных SQLite.

«Был решен ряд задач:

- проведен анализ предметной области и моделирование бизнес–процесса, определена технология реализации;
- разработано клиент–серверное приложение с использованием ASP.NET;
- разработана база данных SQLite для хранения информации о пользователях, заказах и их оценках;
- разработано Android–приложение с использованием фреймворка» [5] Avalonia UI;

– проведено автоматическое unit–тестирование и ручное тестирование приложения.

Разработанное решение удовлетворяет базовые требования приложения для заказа такси, имеет регистрацию и авторизацию пользователей. Возможность оформления, приёма, оценки заказов. Присутствует базовая интеграция с картографическими сервисами.

Следующим этапом развития проекта планируется наполнение системы реальными данными, расширение набора платёжных и картографических интеграций и внедрение модуля отзывов и рейтингов клиентов и водителей. Также эта система легко масштабируется и обновляется за счёт модульной архитектурой.

Список используемой литературы

1. Александров, Д. В. Инструментальные средства информационного менеджмента : CASE–технологии и распределенные информационные системы [Текст] : учебное пособие / Д. В. Александров. – М. : Финансы и статистика, 2011 – 224 с.
2. Баратов, И. В. Большой англо–русский и русско–английский компьютерный словарь [Текст] / И. В. Баратов, ред. Н. В. Морозов. – М. : Живой язык, 2010. – 512 с.
3. Буч, Г. Введение в UML от создателей языка. 2–е изд. [Текст] / Г. Буч, Д. Рамбо, И. Якобсон. – М. : ДМК Пресс, 2015. – 496 с.
4. Глухов, И. Н. Теория систем и системный анализ [Текст] : учебное пособие / И. Н. Глухов – М. : Проспект, 2017 – 117 с.
5. Голицына, О. Л. Информационные системы [Текст] : учебное пособие / О. Л. Голицына, Н. В. Максимов, И. И. Попов. – 2–е изд. – М. : ФОРУМ ИНФРА–М, 2016. – 448 с.
6. Горбенко, А. О. Информационные системы в экономике [Текст] / А. О. Горбенко. – М. : БИНОМ. Лаборатория знаний, 2014. – 292 с.
7. Кумскова, И. А. Базы данных [Текст] : учебник / И. А. Кумскова. – 2–е изд., стер. – М. : КНОРУС», 2015. – 488 с.
8. Стратегическое управление информационными системами [Текст] : учебник / под. ред. Г. Н. Калянова. – М. : Институт Информационных Технологий: БИНОМ. Лаборатория знаний, 2017. – 510 с.
9. Сухомлинов, А. И. Разработка информационных систем [Текст] : учебное пособие / А. И. Сухомлинов – М. : Проспект, 2017. – 112 с.
10. Флэнаган, Д. JavaScript [Текст] : карманный справочник, 3–е изд. / Д. Флэнаган. – М. : ООО «И.Д. Вильямс», 2017 – 320 с.
11. Darwen, H. An Introduction to Relational Database Theory : [Электронный ресурс] / H. Darwen. – UK. : [Б.и.] – 2014, 235 p. – Режим доступа: bookboon.com (дата обращения 12.03.2025)

12. SQL Tutorial : на англ. яз. : [Электронный ресурс] / www.w3schools.com : THE WORLD'S LARGEST WEB DEVELOPER SITE – Режим доступа: https://www.w3schools.com/sql/default.asp (дата обращения 13.04.2025)

13. JavaScript Tutorial : на англ. яз. : [Электронный ресурс] / www.w3schools.com : THE WORLD'S LARGEST WEB DEVELOPER SITE – Режим доступа: https://www.w3schools.com/js/default.asp (дата обращения 17.04.2025)

14. A re–introduction to JavaScript (JS tutorial): на англ. яз.: [Электронный ресурс] / MDN web docs: moz://a – Режим доступа: https://developer.mozilla.org/en–US/docs/Web/JavaScript/A_re–introduction_to_JavaScript (дата обращения 19.04.2025)

15. Троелсен Э., Джепикс Ф. Язык программирования C# 10 и платформа .NET 6 : пер. с англ. / Э. Троелсен, Ф. Джепикс. – М. : ДМК Пресс, 2021. – 896 с.

16. Microsoft. Документация по C# [Электронный ресурс]. – Режим доступа: https://learn.microsoft.com/ru–ru/dotnet/csharp/ (дата обращения: 17.04.2025).

17. Hipp D. SQLite Documentation [Электронный ресурс]. – Режим доступа: https://sqlite.org/docs.html (дата обращения: 11.03.2025)

18. Юсов А. Программирование баз данных в SQLite : учеб. пособие. – СПб. : БХВ-Петербург, 2020. – 320 с.

19. Microsoft. Документация по ASP.NET Core [Электронный ресурс]. – Режим доступа: https://learn.microsoft.com/ru–ru/aspnet/core/ (дата обращения: 12.03.2025).

20. Freeman A. Pro ASP.NET Core 6 / A. Freeman, S. Sanderson. – New York : Apress, 2022. – 1152 с.

21. AvaloniaUI. Avalonia UI Documentation [Электронный ресурс]. – Режим доступа: https://docs.avaloniaui.net/ (дата обращения: 13.04.2025).

22. Яндекс.Карты для разработчиков. MapKit SDK для Android и

iOS [Электронный ресурс]. – Режим доступа: https://yandex.ru/dev/maps/mapkit/ (дата обращения: 14.04.2025).

23. Яндекс.Карты API. Геокодер [Электронный ресурс]. – Режим доступа: https://yandex.ru/dev/maps/geocoder/ (дата обращения: 14.04.2025).

24. Калбах К. Интерфейс пользователя: простые правила проектирования / К. Калбах; пер. с англ. – М. : Вильямс, 2019. – 304 с.

25. Nielsen Norman Group. Mobile UX Design [Электронный ресурс]. – Режим доступа: https://www.nngroup.com/topic/mobile–ux/ (дата обращения: 15.03.2025).

26. Соловьёв Д. Разработка мобильных приложений: UI/UX дизайн: учеб. пособие. – СПб. : Питер, 2021. – 256 с.

Приложение А

Файл разметки меню авторизации приложения

```
<UserControl xmlns="https://github.com/avaloniaui"
             xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
             xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
             xmlns:vm="clr-namespace:TomDrive.ViewModels"
             xmlns:mc="http://schemas.openxmlformats.org/markup-
compatibility/2006"
             mc:Ignorable="d" d:DesignWidth="480" d:DesignHeight="960"
             x:Class="TomDrive.Views.LoginView"
             x:DataType="vm:LoginViewModel">

    <Design.DataContext>
        <vm:LoginViewModel />
    </Design.DataContext>

    <UserControl.Styles>
        <Style Selector="Button.login_btns">
            <Setter Property="VerticalAlignment" Value="Bottom" />
            <Setter Property="HorizontalAlignment" Value="Stretch" />
            <Setter Property="HorizontalContentAlignment" Value="Center" />
            <Setter Property="FontSize" Value="16" />
        </Style>
        <Style Selector="Label">
```

```
            <Setter Property="FontSize" Value="16" />
        </Style>
    </UserControl.Styles>

    <StackPanel Spacing="5" VerticalAlignment="Center"
HorizontalAlignment="Center">
        <Label Content="TomDrive"
               HorizontalAlignment="Center"
               FontSize="32"
               FontFamily="Bahnschrift" />
        <Border Background="#575A5D"
                HorizontalAlignment="Center"
                VerticalAlignment="Center"
                Width="300" Height="300"
                CornerRadius="50"
                Padding="25">
            <Border.Effect>
                <DropShadowEffect BlurRadius="10" />
            </Border.Effect>
            <Grid RowDefinitions="Auto,Auto,Auto,*">
                <StackPanel Grid.Row="0">
                    <Label Content="Логин" />
                    <TextBox Text="{Binding Login}" />
                </StackPanel>
                <StackPanel Grid.Row="1">
                    <Label Content="Пароль" />
                    <TextBox Text="{Binding Password}" PasswordChar="*"
RevealPassword="{Binding IsVisible}" />
                </StackPanel>
                <StackPanel Grid.Row="2" VerticalAlignment="Top">
                    <CheckBox Content="Показать пароль" IsChecked="{Binding
IsVisible}" />
                </StackPanel>
                <Grid Grid.Row="3" RowDefinitions="*,*">
                    <Button Grid.Row="0"
                            Classes="login_btns"
                            Content="Войти"
                            Command="{Binding AuthorizeCommand}" />
                    <Button Grid.Row="1"
                            Classes="login_btns"
                            Content="Зарегистрироваться"
                            Command="{Binding RegisterCommand}" />
                </Grid>
            </Grid>
        </Border>
        <Label Content="{Binding ErrorMessage}"
               IsVisible="{Binding IsError}"
               HorizontalAlignment="Center"
               Foreground="Red" />
    </StackPanel>

</UserControl>
```

Приложение Б

Файл разметки основного окна приложения с боковым меню разделов

```
<UserControl xmlns="https://github.com/avaloniaui"
             xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
             xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
             xmlns:vm="clr-namespace:TomDrive.ViewModels.MainMenuViewModels"
             xmlns:mc="http://schemas.openxmlformats.org/markup-
compatibility/2006"
             mc:Ignorable="d" d:DesignWidth="480" d:DesignHeight="960"
             x:Class="TomDrive.Views.MainMenuViews.MainMenuView"
             x:DataType="vm:MainMenuViewModel" FontSize="16">

    <UserControl.Styles>
        <Style Selector="Button">
            <Setter Property="VerticalContentAlignment" Value="Center" />
            <Setter Property="HorizontalContentAlignment" Value="Center" />
            <Setter Property="HorizontalAlignment" Value="Stretch" />
            <Setter Property="Height" Value="50" />
        </Style>
    </UserControl.Styles>

    <DockPanel>
        <Grid DockPanel.Dock="Top"
              ColumnDefinitions="50, *"
              Background="#1A1D20">
            <Button Grid.Column="0" Content="..." Command="{Binding
ToggleMenuCommand}" />
        </Grid>

        <SplitView IsPaneOpen="{Binding IsMenuOpen}"
                   OpenPaneLength="200">
            <SplitView.Pane>
                <ListBox Background="#1A1D20"
                         SelectionMode="Single"
                         SelectedIndex="{Binding CurrentViewId}">
                    <ListBoxItem Content="Профиль" />
                    <ListBoxItem Content="Текущий заказ" />
                    <ListBoxItem Content="История заказов" />
                    <ListBoxItem Content="Режим водителя" />
                </ListBox>
            </SplitView.Pane>
            <ContentControl Content="{Binding CurrentViewModel}" />
        </SplitView>
    </DockPanel>
</UserControl>
```

Приложение В

Код компонента WebView, HTML и JavaScript

```
<UserControl xmlns="https://github.com/avaloniaui"
             xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
             xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
             xmlns:vm="clr-namespace:TomDrive.ViewModels.MainMenuViewModels"
             xmlns:mc="http://schemas.openxmlformats.org/markup-
compatibility/2006"
             mc:Ignorable="d" d:DesignWidth="480" d:DesignHeight="960"
             x:Class="TomDrive.Views.MainMenuViews.CurrentOrderView"
             x:DataType="vm:CurrentOrderViewModel">

    <Design.DataContext>
        <vm:CurrentOrderViewModel />
    </Design.DataContext>

    <UserControl.Styles>
        <Style Selector="StackPanel.order">
            <Setter Property="Margin" Value="5" />
        </Style>
        <Style Selector="Label.propertyTitle">
            <Setter Property="FontSize" Value="18" />
        </Style>
        <Style Selector="Label.propertyValue">
            <Setter Property="FontSize" Value="14" />
            <Setter Property="FontWeight" Value="Light" />
        </Style>
        <Style Selector="Label.timer">
            <Setter Property="FontSize" Value="20" />
            <Setter Property="Foreground" Value="LightGreen" />
        </Style>
    </UserControl.Styles>

    <Grid RowDefinitions="50,Auto,Auto,Auto,Auto,Auto,*,Auto,Auto">
        <Label Grid.Row="0" Content="Текущий заказ" Classes="viewTitle" />
        <StackPanel Grid.Row="1" Classes="order">
            <Label Content="Пункт отправления:" Classes="propertyTitle" />
            <Label Content="{Binding CurrentOrder.address_from}"
                   Classes="propertyValue" />
        </StackPanel>
        <StackPanel Grid.Row="2" Classes="order">
            <Label Content="Пункт назначения:" Classes="propertyTitle" />
            <Label Content="{Binding CurrentOrder.address_to}"
                   Classes="propertyValue" />
        </StackPanel>
        <StackPanel Grid.Row="3" Classes="order">
            <Label Content="Клиент:" Classes="propertyTitle" />
            <Label Content="{Binding CurrentOrder.user.username}"
Classes="propertyValue" />
        </StackPanel>
        <StackPanel Grid.Row="4" Classes="order">
            <Label Content="Стоимость поездки:" Classes="propertyTitle" />
            <Label Content="{Binding CurrentOrder.amount, StringFormat={}{0:0.00
руб}}"
                   Classes="propertyValue" />
        </StackPanel>
        <StackPanel Grid.Row="5" Classes="order">
            <Label Content="Комментарий к заказу:" Classes="propertyTitle" />
            <Label Content="{Binding CurrentOrder.comment}"
Classes="propertyValue" />
        </StackPanel>
        <StackPanel Grid.Row="6" Margin="10">
```

```
        </StackPanel>
        <Label Grid.Row="7"
               Classes="timer"
               Margin="5,0,0,0"
               Content="{Binding StatusString}" />
        <Grid Grid.Row="8" ColumnDefinitions="Auto,*" Margin="5">
            <Label Grid.Column="0" Content="{Binding TimerString}" FontSize="36"
/>
            <Button Grid.Column="1"
                    Content="Оплатить"
                    Command="{Binding PayOrderCommand}"
                    IsVisible="{Binding PayVisibility}"
                    IsEnabled="{Binding PayEnabled}">
            </Button>
            <Button Grid.Column="1"
                    Content="Отменить"
                    Command="{Binding CancelOrderCommand}"
                    IsVisible="{Binding CancelVisibility}">
            </Button>
            <Button Grid.Column="1"
                    Content="Завершить"
                    Command="{Binding FinishOrderCommand}"
                    IsVisible="{Binding FinishVisibility}">
            </Button>
        </Grid>
    </Grid>
</UserControl>
```

Приложение Г

Код модели представления разметки окна «Текущий заказ»

```
using System;
using System.Globalization;
using System.Threading.Tasks;
using System.Timers;
using CommunityToolkit.Mvvm.ComponentModel;
using CommunityToolkit.Mvvm.Input;
using CommunityToolkit.Mvvm.Messaging;
using TomDrive.API;
using TomDrive.Messages;
using TomDrive.Models;

namespace TomDrive.ViewModels.MainMenuViewModels;

// Модель представления активного заказа
public partial class CurrentOrderViewModel : ViewModelBase {
    [ObservableProperty] private Order _currentOrder = null!;
    [ObservableProperty] private string _htmlContent = string.Empty;

    private int _minutes;
    private int _seconds;
    private Timer _orderTimer = null!;
    private Timer _updateTimer = null!;
    [ObservableProperty] private string _statusString = null!;
    [ObservableProperty] private string _timerString = "00:00";
    [ObservableProperty] private bool _cancelVisibility;
    [ObservableProperty] private bool _payVisibility;
    [ObservableProperty] private bool _payEnabled;
    [ObservableProperty] private bool _finishVisibility;

    // Конструктор модели
    public CurrentOrderViewModel(Order order) {
        CurrentOrder = order;
        CurrentOrder.isPayed = false;
        CurrentOrder.isTaken = false;
```

```
        if (Session.IsDriverMode) {
            StatusString = "Удачной поездки!";
        } else {
            StatusString = Utils.GetOrderStatusString(order.status);
        }
        _ = CreateMapAsync();

        StartOrderTimer();
        StartProcessingTimer();
        UpdateButtons();
    }

    // Стандартный конструктор для дизайнера
    public CurrentOrderViewModel() {
    }

    private async Task CreateMapAsync() {
        var coordStringOut = await
CoreAPI.GetCoordinatesFromAddress(CurrentOrder.address_from);
        var coordStringIn = await
CoreAPI.GetCoordinatesFromAddress(CurrentOrder.address_to);

        double coordOutX = double.Parse(coordStringOut.Split(",")[0].Replace(".",
","));
        double coordOutY = double.Parse(coordStringOut.Split(",")[1].Replace(".",
","));
        double coordInX = double.Parse(coordStringIn.Split(",")[0].Replace(".",
","));
        double coordInY = double.Parse(coordStringIn.Split(",")[1].Replace(".",
","));

        double coordCenterX = (coordInX + coordOutX) / 2.0;
        double coordCenterY = (coordInY + coordOutY) / 2.0;
        string coordStringCenter = $"{coordCenterX.ToString("F6",
CultureInfo.InvariantCulture)}, {coordCenterY.ToString("F6",
CultureInfo.InvariantCulture)}";

        double diffX = coordOutX - coordInX;
        double diffY = coordOutY - coordInY;
        double length = Math.Sqrt(diffX * diffX + diffY * diffY) * 100.0;

        // 0.0020 - 15
        // 0.0197 - 13
        // 0.0424 - 12
        // 0.20 - 15
        // 1.97 - 13
        // 4.24 - 12
        int scale = 11;

        if (length <= 0.5) scale = 15;
        else if (length <= 1) scale = 14;
        else if (length <= 2) scale = 13;
        else if (length <= 5) scale = 12;

        HtmlContent = CoreAPI.GetHtmlMapContent(coordStringOut, coordStringIn,
coordStringCenter, scale.ToString());
    }

    private void UpdateButtons() {

        if (Session.IsDriverMode) {
            FinishVisibility = false;
            CancelVisibility = false;
            PayVisibility = false;
```

```
            return;
        }

        switch (CurrentOrder.status) {
            case OrderStatus.Waiting: {
                FinishVisibility = false;
                PayVisibility = false;
                CancelVisibility = true;
                break;
            }
            case OrderStatus.Taken: {
                FinishVisibility = false;
                CancelVisibility = false;
                PayVisibility = true;
                PayEnabled = true;
                break;
            }
            case OrderStatus.Payed: {
                FinishVisibility = false;
                CancelVisibility = false;
                PayVisibility = true;
                PayEnabled = false;
                break;
            }
            case OrderStatus.Finished: {
                break;
            }
        }
    }

    private void StartOrderTimer() {
        _orderTimer = new Timer();
        _orderTimer.Interval = 1000;
        _orderTimer.Elapsed += UpdateOrderTimerAfterElapsed;
        _orderTimer.Start();
    }

    private void StartProcessingTimer() {
        _updateTimer = new Timer();
        _updateTimer.Interval = 5000;
        _updateTimer.Elapsed += UpdateOrderInfoAfterElapsed;
        _updateTimer.Start();
    }

    private void UpdateOrderInfoAfterElapsed(object? sender, EventArgs e) {
        UpdateOrderInfo();
    }

    private void UpdateOrderInfo() {
        var updatedOrder = OrdersCoreApi.GetOrder();
        if (updatedOrder != null) {
            CurrentOrder = updatedOrder;
            StatusString = Utils.GetOrderStatusString(updatedOrder.status);
            UpdateButtons();
        }
    }

    private void UpdateOrderTimerAfterElapsed(object? sender, ElapsedEventArgs
elapsedEventArgs) {
        _seconds++;
        if (_seconds >= 60) {
            _minutes++;
            _seconds = 0;
        }
```

```
        if (_minutes >= 60) {
            _seconds = 0;
            _minutes = 0;
        }

        TimerString = $"{_minutes:D2}:{_seconds:D2}";
    }

    [RelayCommand]
    private async Task CancelOrder() {
        var success = await OrdersCoreApi.CancelOrder();
        if (success) WeakReferenceMessenger.Default.Send<OrderCanceledMessage>();
    }

    [RelayCommand]
    private async Task PayOrder() {
        var success = await OrdersCoreApi.PayOrder();
        if (success) {
            CurrentOrder.isPayed = true;
            CurrentOrder.status = OrderStatus.Payed;
            await OrdersCoreApi.UpdateOrder(CurrentOrder);
            UpdateButtons();
        }
    }

    [RelayCommand]
    private async Task FinishOrder() {
        var success = await OrdersCoreApi.FinishOrder();
        if (success) {
            await OrdersCoreApi.FinishOrder();
            UpdateButtons();
        }
    }
}
```

Приложение Д

Код CoreAPI отвечающий за отображение интерактивной карты

```
using System.Net.Http;
using System.Net.Http.Headers;
using System.Text.Json;
using System.Threading.Tasks;

namespace TomDrive.API;

public class CoreAPI {
    protected static readonly HttpClient client = new();
    private static readonly string baseUrl = "http://83.234.158.17:4444";

    protected static HttpRequestMessage CreateRequest(HttpMethod method, string
url) {
        var request = new HttpRequestMessage(method, baseUrl + url);
        request.Headers.Authorization = new AuthenticationHeaderValue("Bearer",
Session.Token);
        return request;
    }

    protected static HttpRequestMessage CreateRequestWithoutAuth(HttpMethod
method, string url) {
        var request = new HttpRequestMessage(method, baseUrl + url);
        return request;
    }

    public static async Task<string> GetCoordinatesFromAddress(string address) {
        var geocode = $"Воркута {address.Replace(" ", "+")}";

        var stringRequest =
            $"https://geocode-maps.yandex.ru/v1/?apikey=d6cf0434-36f7-4616-ad00-
065c61df4644&geocode={geocode}&lang=ru&format=json";

        var request = new HttpRequestMessage(HttpMethod.Get, stringRequest);
        var response = await client.SendAsync(request);

        var content = await response.Content.ReadAsStringAsync();

        var document = JsonDocument.Parse(content);
        var root = document.RootElement;

        var pos = root
            .GetProperty("response")
            .GetProperty("GeoObjectCollection")
            .GetProperty("featureMember")[0]
            .GetProperty("GeoObject")
            .GetProperty("Point")
```

```
            .GetProperty("pos")
            .GetString();

        if (pos == null) return "";
        var splitTemp = pos.Split(' ');
        var result = $"{splitTemp[1]},{splitTemp[0]}";

        return result;
    }

    public static string GetHtmlMapContent(string address_out, string address_in,
string center_point, string scale) {
        var htmlPage = @"<!DOCTYPE html>
<html>
<head>
  <meta charset=""utf-8"">
  <title>Yandex Map in Avalonia</title>
  <script src=""https://api-maps.yandex.ru/2.1/?apikey=f21f0214-9633-47fc-b39d-
9736b8b598de&lang=ru_RU&load=Map,Placemark""></script>
  <style>html, body, #map { width:100%; height:100%; margin:0; padding:0;
}</style>
</head>
<body>
  <div id=""map""></div>
  <script>
    ymaps.ready(function() {
      var myMap = new ymaps.Map(""map"", {
        center: [CENTER_POINT], // Воркута (6 цифр после запятой)
        zoom: SCALE
      });
      var place_out = new ymaps.Placemark([ADDRESS_OUT],
                      {hintContent: 'Исходная точка', ballonContent: 'Описание'},
                      {preset: 'islands#icon', iconColor: '#0095b6'});
      var place_in = new ymaps.Placemark([ADDRESS_IN],
                      {hintContent: 'Вторичная точка', ballonContent: '...'},
                      {preset: 'islands#icon', iconColor: '#0095b6'});
      myMap.geoObjects.add(place_out);
      myMap.geoObjects.add(place_in);
    });
  </script>
</body>
</html>";

        var result = htmlPage.Replace("ADDRESS_OUT", address_out);
        result = result.Replace("ADDRESS_IN", address_in);
        result = result.Replace("CENTER_POINT", center_point);
        result = result.Replace("SCALE", scale);

        return result;
    }
}
```