

Министерство образования Республики Беларусь  
Учреждение образования «Белорусский государственный университет  
информатики и радиоэлектроники»

Факультет компьютерных систем и сетей  
Кафедра информатики  
Дисциплина «Объектно-ориентированное программирование»

«К ЗАЩИТЕ ДОПУСТИТЬ»

Руководитель курсового проекта  
ассистент кафедры информатики

\_\_\_\_\_. В.Д.Владымцев  
\_\_\_\_\_. \_\_\_\_\_. 2023

## **ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

к курсовому проекту

на тему:

**«Агрегатор такси»**

БГУИР КП 1-40 04 01 048 ПЗ

Выполнил студент группы 153503  
Кахновский Евгений Сергеевич

\_\_\_\_\_  
(подпись студента)

Курсовой проект представлен  
на проверку \_\_\_\_\_. \_\_\_\_\_. 2023

\_\_\_\_\_  
(подпись студента)

Минск 2023

## СОДЕРЖАНИЕ

ВВЕДЕНИЕ.....	4
1 Анализ используемых источников.....	5
2 Теоретическое обоснование разработки.....	7
3 Паттерны программирования, используемые в разработке приложения ....	17
4 Функциональные возможности программы.....	21
5 Архитектура разрабатываемой программы.....	27
ЗАКЛЮЧЕНИЕ .....	29
СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ .....	30
ПРИЛОЖЕНИЕ А (обязательное) Исходный код программы.....	31
ПРИЛОЖЕНИЕ Б (обязательное) Диаграмма классов .....	37
ПРИЛОЖЕНИЕ В (обязательное) USE-CASE диаграмма.....	41
ПРИЛОЖЕНИЕ Г (обязательное) Графический интерфейс .....	42
ПРИЛОЖЕНИЕ Д (обязательное) Ведомость.....	44

## ВВЕДЕНИЕ

В современном цифровом мире услуги такси стали неотъемлемой частью нашей повседневной жизни. Они предоставляют удобный и эффективный способ перемещения по городу, позволяя пользователям вызывать и заказывать такси с помощью мобильных устройств. Однако разнообразие приложений и платформ для такси создает вызов для пользователей и разработчиков в поиске подходящего агрегатора такси.

В связи с этим, разработка агрегатора такси представляет собой актуальную и перспективную задачу. Агрегатор такси предлагает удобную платформу, объединяющую различные службы такси и предоставляющую пользователям возможность выбора наиболее подходящей и доступной опции такси.

Цели курсового проекта:

- 1 Создать интуитивно понятный пользовательский интерфейс, который позволит пользователям легко находиться в системе и совершать заказы такси без лишних усилий.

- 2 Обеспечить безопасное хранение и обработку личных данных пользователей.

- 3 Реализовать административные функции, которые бы предоставили полный доступ администраторам для эффективного управления системой.

- 4 Обеспечить расширяемость и эффективную поддержку веб-приложения.

Задачи курсового проекта:

- 1 Осуществить разработку агрегатора такси, используя фреймворк ASP.NET и архитектуру MVC, для обеспечения удобного и эффективного веб-приложения.

- 2 Провести анализ требований к системе и установить основные критерии ее функционирования и надежности, с учетом потребностей пользователей и требований к безопасности и производительности.

- 3 Приобрести необходимые теоретические и практические навыки для реализации ролей в проекте, включая функциональности для клиентов и администраторов, с учетом их различных прав и возможностей.

- 4 Реализовать веб-приложение с использованием локальной базы данных SQLite и Entity Framework, обеспечивающее надежное хранение и доступ к данным, а также эффективную обработку запросов пользователей и администраторов.

## 1 АНАЛИЗ ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

Указанные источники предоставляют обширный набор информации, необходимый для изучения различных технологий веб-разработки. Каждый источник представляет собой ценный ресурс, содержащий официальные документации, учебники и руководства, которые позволяют ознакомиться с конкретными аспектами каждой технологии в деталях.

1 "CLR via C#. Программирование на платформе Microsoft .NET Framework 4.5 на языке C#" (Джеффри Рихтер) - это исчерпывающее руководство, которое освещает широкий спектр тем, связанных с платформой .NET и языком C#. Книга предлагает глубокое понимание внутренней работы Common Language Runtime (CLR) и предоставляет инсайты для разработчиков, которые хотят создавать высокопроизводительное и эффективное программное обеспечение на платформе .NET.

2 "Язык программирования C# 7 и платформы .NET и .NET Core" (Эндрю Троелсен, Филипп Джепикс) - данная книга является одним из наиболее уважаемых источников информации по языку C# и платформе .NET. Она предлагает подробное изложение основных возможностей языка, включая объектно-ориентированное программирование, обработку исключений, работу с коллекциями, асинхронное программирование и многое другое. Книга также освещает различия между классической платформой .NET и новой платформой .NET Core.

3 "Совершенный код. Мастер-класс" (Стив Макконнелл) - это авторитетное руководство, которое наставляет разработчиков на создание качественного программного кода. Книга предлагает советы по организации кода, выбору правильных алгоритмов и структур данных, использованию эффективных практик разработки и многое другое. Она помогает повысить производительность, надежность и сопровождаемость кода, что является важным аспектом при разработке сложных веб-приложений.

4 "ASP.Net Core в действии" (Эндрю Лок) - это практическое руководство, которое помогает разработчикам освоить ASP.NET Core и создавать масштабируемые веб-приложения. Книга предлагает обширное покрытие основных концепций и технологий, включая маршрутизацию, модели представления, валидацию данных, аутентификацию и авторизацию, работу с базами данных и многое другое. Она также освещает новейшие тенденции и лучшие практики разработки веб-приложений на платформе ASP.NET Core.

5 "HTML 5, CSS 3 и Web 2.0. Разработка современных Web-сайтов" (Владимир Дронов) - это практическое руководство по разработке современных веб-сайтов с использованием HTML 5, CSS 3 и технологий Web 2.0. Книга предлагает пошаговое руководство по созданию интерактивных и привлекательных веб-интерфейсов, включая работу с медиа-элементами, анимацией, адаптивным дизайном и другими современными возможностями веб-разработки.

6 "Metanit" (<https://metanit.com/>) - это онлайн-ресурс, который предоставляет обширные учебные материалы на русском языке по различным технологиям веб-разработки. Он содержит подробные объяснения, примеры кода и практические упражнения по ASP.NET Core, C#, HTML, CSS, JavaScript и другим технологиям.

7 "Bootstrap - документация" (<https://getbootstrap.com/>) - это официальная документация по Bootstrap, популярному фреймворку для разработки адаптивных и стильных веб-интерфейсов. Документация содержит подробное описание компонентов, классов и функциональности Bootstrap, а также примеры использования.

8 "ASP.NET MVC - Microsoft Learn" (<https://learn.microsoft.com/en-us/aspnet/mvc/>) - это официальный ресурс Microsoft Learn, предоставляющий обучающие материалы и руководства по разработке веб-приложений на платформе ASP.NET MVC. Ресурс включает в себя интерактивные уроки, видеоуроки и практические задания.

9 "Entity Framework documentation" (<https://learn.microsoft.com/en-us/ef/>) - это официальная документация по Entity Framework, инструменту для работы с базами данных в приложениях на платформе .NET. Документация предоставляет подробное описание возможностей Entity Framework, примеры использования и рекомендации по проектированию баз данных.

10 "W3Schools" (<https://www.w3schools.com/>) - это популярный онлайн-ресурс, который предлагает обширную коллекцию учебных материалов по веб-разработке, включая HTML, CSS, JavaScript и другие технологии. Ресурс содержит подробные объяснения, примеры кода и интерактивные упражнения.

Указанные источники позволяют получить глубокое понимание технологий, использованных в описанных вами сценариях, и предлагают полезные материалы для самостоятельного изучения и практического применения этих технологий.

## 2 ТЕОРЕТИЧЕСКОЕ ОБОСНОВАНИЕ РАЗРАБОТКИ

При разработке веб-приложения агрегатор такси были соблюдены все принципы объектно-ориентированного программирования (ООП). Особое внимание было уделено принципам SOLID.

В основной стек технологий используемых при разработке веб-приложения входят: язык программирования C#, платформа .NET, фреймворки ASP.NET и ASP.NET Core MVC, внедрение зависимостей (Dependency Injection, DI), Razor Page, HTML5+CSS3+JS, SQLite, Entity Framework (EF), Bootstrap5, BING Maps API, WEB API. Также были задействованы такие паттерны проектирования как Repository, UnitOfWork, MVC.

Проект реализован с помощью многоуровневой архитектуры (Data Access Layer(DAL), Business Logic Layer(BLL), Domain Layer, Presentation Layer) построения веб-приложений.

Рассмотрим каждую использующуюся технологию по отдельности.

Платформа .NET является мощным инструментарием для разработки разнообразных приложений, включая веб-приложения, на различных языках программирования, таких как C#, VB.NET и F#. Она разработана компанией Microsoft и предоставляет программистам и разработчикам набор инструментов, фреймворков и библиотек для эффективного создания высокопроизводительных и масштабируемых приложений.

Вот несколько ключевых преимуществ платформы .NET, которые привели к выбору этой платформы для разработки агрегатора такси:

1 Многоплатформенность: платформа .NET поддерживает разработку приложений для различных операционных систем, включая Windows, macOS и Linux. Это позволяет агрегатору такси быть доступным для широкой аудитории пользователей, независимо от используемой платформы.

2 Мощный язык программирования: с использованием языка программирования C#, который является одним из основных языков платформы .NET, разработчики получают преимущества типизации, объектно-ориентированного программирования и многопоточности. C# обладает лаконичным синтаксисом, обширной стандартной библиотекой и широкой поддержкой сообщества разработчиков.

3 Большая экосистема инструментов и фреймворков: платформа .NET предоставляет широкий выбор инструментов и фреймворков для разработки веб-приложений. В случае агрегатора такси, были использованы ASP.NET,

ASP.NET Core MVC, Razor Page и WEB API, которые обеспечивают высокую производительность, гибкость разработки, модульность и удобство взаимодействия с клиентами.

4 Безопасность: платформа .NET предоставляет набор механизмов и инструментов для обеспечения безопасности приложений, включая автоматическую проверку типов, обработку исключений, аутентификацию и авторизацию. Это важно для агрегатора такси, где обработка персональных данных и обеспечение безопасности платежей являются критическими аспектами.

5 Поддержка баз данных: платформа .NET обеспечивает простую интеграцию с различными системами управления базами данных. В случае агрегатора такси была использована SQLite для хранения информации о пользователях, заказах такси.

6 Интеграция с внешними сервисами: платформа .NET обладает богатым набором инструментов для интеграции с внешними сервисами, такими как Bing Maps API, который был использован для интеграции карт и маршрутизации в агрегаторе такси.

В целом, выбор платформы .NET для разработки агрегатора такси обусловлен её многоплатформенностью, поддержкой современных языков программирования, мощными инструментами разработки, безопасностью и поддержкой интеграции с внешними сервисами, что позволяет эффективно создавать надежные и высокофункциональные веб-приложения.

ASP.NET Core MVC является фреймворком для разработки веб-приложений на платформе .NET Core. Он предоставляет мощные инструменты для создания веб-интерфейса и бизнес-логики, обеспечивая высокую производительность и гибкость разработки. Вот несколько преимуществ ASP.NET Core MVC и причины выбора этого фреймворка для реализации агрегатора такси:

1 Кроссплатформенность: ASP.NET Core MVC основан на платформе .NET Core, что позволяет разрабатывать приложения, которые могут работать на различных операционных системах, включая Windows, macOS и Linux. Это дает возможность обеспечить доступность агрегатора такси для широкой аудитории пользователей.

2 Высокая производительность: ASP.NET Core MVC предлагает оптимизации производительности, такие как механизмы кэширования, асинхронное выполнение и масштабируемость. Это позволяет обеспечить отзывчивость и быстрое действие приложения, что особенно важно для агрегатора такси, где время отклика имеет большое значение.

3 Модульность и гибкость: ASP.NET Core MVC основан на принципах модульности и разделения ответственности. Он позволяет разделить приложение на логические компоненты (контроллеры, представления, модели), что упрощает поддержку кода и его масштабируемость. Также фреймворк предлагает гибкую конфигурацию маршрутов и позволяет использовать различные стратегии маршрутизации.

4 Богатая экосистема: ASP.NET Core MVC имеет обширную экосистему инструментов, библиотек и расширений, которые облегчают разработку веб-приложений. Это включает в себя ORM-фреймворк Entity Framework для работы с базой данных, библиотеки для обработки входящих запросов, валидации данных, авторизации и аутентификации пользователей и многое другое.

5 Тестирование и отладка: ASP.NET Core MVC предоставляет средства для тестирования и отладки приложений, что упрощает процесс разработки и обеспечивает стабильность и надежность приложения. Разработчики могут проводить модульное, интеграционное и функциональное тестирование своего кода, а также использовать мощные инструменты отладки для идентификации и устранения ошибок.

Итак, выбор ASP.NET Core MVC для реализации агрегатора такси обусловлен его кроссплатформенностью, высокой производительностью, модульностью, богатой экосистемой инструментов и удобством в тестировании и отладке. Этот фреймворк обеспечивает разработчикам мощный инструментарий для создания надежных и масштабируемых веб-приложений, соответствующих требованиям агрегатора такси.

Технология Razor Pages является частью фреймворка ASP.NET Core и предоставляет удобный способ разработки веб-приложений, основанных на модели просмотра и шаблонах Razor. Вот несколько преимуществ и причин выбора технологии Razor Pages для реализации агрегатора такси:

1 Простота и интуитивность: Razor Pages предлагает простую и интуитивно понятную модель программирования, основанную на разделении страниц на разделы (Razor Pages) и соответствующие им модели просмотра (View Models). Это делает процесс разработки более понятным и легким для новых разработчиков.

2 Удобство взаимодействия с данными: Razor Pages обеспечивает прямую интеграцию с моделями просмотра и обработку данных. Разработчики могут легко передавать данные между страницами и моделями просмотра, что упрощает реализацию бизнес-логики и взаимодействие с базой данных.

3 Гибкость и масштабируемость: Razor Pages позволяет создавать



маленькие и самодостаточные страницы, что облегчает разделение и модульность кода. Разработчики могут легко масштабировать приложение, добавляя новые Razor Pages или модели просмотра, не затрагивая другие части приложения.

4 Использование шаблонов Razor: Razor Pages использует синтаксис Razor для создания динамических и гибких шаблонов представлений. Это позволяет разработчикам создавать легко читаемые и поддерживаемые представления, в которых можно использовать код C# для динамической генерации контента.

5 Поддержка асинхронности: Razor Pages предоставляет встроенную поддержку асинхронных операций, что позволяет выполнять длительные операции без блокирования основного потока выполнения. Это повышает отзывчивость приложения и общую производительность.

6 Интеграция с другими компонентами ASP.NET Core: Razor Pages хорошо интегрируется с другими компонентами ASP.NET Core, такими как маршрутизация, инъекция зависимостей, аутентификация и авторизация. Это облегчает взаимодействие с другими частями приложения и использование их функциональности.

Итак, выбор технологии Razor Pages для реализации агрегатора такси основан на её простоте и интуитивности, удобстве взаимодействия с данными, гибкости и масштабируемости, использовании шаблонов Razor, поддержке асинхронности и интеграции с другими компонентами ASP.NET Core. Это позволяет разработчикам эффективно создавать и поддерживать веб-приложение, отвечающее требованиям агрегатора такси.

C# (произносится "си шарп") — это мощный и элегантный объектно-ориентированный язык программирования, разработанный компанией Microsoft. Вот несколько преимуществ и причин выбора языка C# для реализации агрегатора такси:

1 Язык с широким функциональным спектром: C# обладает обширным функциональным набором, включая поддержку сильной типизации, управления памятью, событий, делегатов, асинхронного программирования, LINQ (Language Integrated Query) и многое другое. Это обеспечивает разработчикам гибкость и возможность реализовывать сложные бизнес-логики и функциональности в приложении.

2 Интеграция с платформой .NET: C# является основным языком программирования для разработки на платформе .NET, которая предлагает мощный набор инструментов и библиотек для создания веб-приложений. Благодаря этому, C# обладает преимуществами интеграции с другими

технологиями и компонентами .NET, такими как ASP.NET, Entity Framework, LINQ и многими другими.

3 Широкая поддержка сообщества и ресурсов: C# является одним из самых популярных языков программирования, и за ним стоит большое активное сообщество разработчиков. Это обеспечивает доступность множества ресурсов, форумов, библиотек и инструментов поддержки, которые могут быть использованы для разработки агрегатора такси.

4 Удобство и читаемость кода: C# имеет чистый и понятный синтаксис, что делает код на языке C# легко читаемым и понятным для разработчиков. Это способствует разработке и поддержке приложения, особенно в командной работе, где читаемый и понятный код играет важную роль.

5 Платформонезависимость: C# можно использовать для разработки приложений, которые могут работать на различных платформах, таких как Windows, macOS и Linux. Это обеспечивает гибкость и доступность приложения для широкой аудитории пользователей.

Итак, выбор языка C# для реализации агрегатора такси обусловлен его широким функциональным спектром, интеграцией с платформой .NET, поддержкой сообщества и ресурсов, удобством и читаемостью кода, а также платформонезависимостью. Это обеспечивает эффективную разработку приложения с использованием C#.

SQLite - это компактная, серверная база данных, которая работает без необходимости отдельного сервера. Вот несколько преимуществ и причин выбора технологии SQLite для реализации агрегатора такси:

1 Простота использования: SQLite предоставляет простой и интуитивно понятный способ хранения и управления данными. Она не требует настройки сложной инфраструктуры сервера баз данных и может быть легко интегрирована в приложение. Это упрощает разработку и развертывание приложения, особенно для малых и средних проектов.

2 Компактность и эффективность: SQLite представляет собой самодостаточную библиотеку, которая хранит базу данных в одном файле. Это делает ее компактной и эффективной в использовании ресурсов, что особенно важно для приложений с ограниченными ресурсами, таких как мобильные приложения.

3 Поддержка SQL: SQLite полностью поддерживает стандартный язык структурированных запросов SQL. Это позволяет разработчикам выполнять сложные запросы и манипулировать данными с помощью знакомого и мощного языка. SQLite также поддерживает транзакции, что обеспечивает целостность данных и безопасность операций.

4 Переносимость: SQLite может работать на различных операционных системах, включая Windows, macOS, Linux и другие. Это обеспечивает переносимость приложения, позволяя запускать его на различных платформах без необходимости внесения значительных изменений в код.

5 Надежность: SQLite обеспечивает надежное хранение данных и обеспечивает целостность информации при возникновении сбоев или аварийных ситуаций. Она также предоставляет механизмы резервного копирования и восстановления данных для обеспечения безопасности информации.

Выбор технологии SQLite для реализации агрегатора такси обусловлен ее простотой использования, компактностью, поддержкой SQL, переносимостью и надежностью. SQLite предоставляет эффективное и надежное решение для хранения и управления данными в приложении.

Технологии HTML5, CSS3 и JavaScript (JS) являются основными инструментами для разработки современных веб-интерфейсов. Вот несколько преимуществ и причин выбора этого набора технологий для реализации агрегатора такси:

HTML5: HTML5 представляет собой последнюю версию стандарта HTML, который используется для создания структуры и разметки веб-страниц. Он предлагает множество новых элементов и функций, таких как семантические теги, мультимедиа-элементы, хранилище данных (localStorage) и другие. Это позволяет разработчикам создавать более современные и интерактивные пользовательские интерфейсы.

CSS3: CSS3 является последней версией стандарта CSS и предоставляет богатые возможности для стилизации и визуализации веб-страниц. Он включает в себя новые свойства и модули, такие как гибкая сетка (flexbox), анимации, переходы, тени, градиенты и многое другое. Это позволяет создавать эффектные дизайны и адаптивные пользовательские интерфейсы.

JavaScript: JavaScript является клиентским языком программирования, который позволяет добавлять динамическую функциональность и взаимодействие на веб-страницах. Он поддерживает множество возможностей, таких как манипуляция DOM-деревом, обработка событий, асинхронные запросы к серверу (AJAX), анимации и многое другое. JavaScript является неотъемлемой частью создания интерактивных и динамических пользовательских интерфейсов.

1 Кросс-платформенность: HTML5, CSS3 и JavaScript являются кросс-платформенными технологиями, что означает, что веб-приложение, созданное с использованием этих технологий, может работать на различных платформах

и устройствах, включая компьютеры, мобильные устройства и планшеты. Это обеспечивает гибкость и доступность приложения для широкой аудитории пользователей.

2 Обширное сообщество и ресурсы: HTML5, CSS3 и JavaScript имеют огромное и активное сообщество разработчиков, а также обилие ресурсов, библиотек и инструментов для поддержки разработки. Это обеспечивает доступность множества ресурсов, помощи и решений, которые могут быть использованы в процессе разработки.

Выбор HTML5, CSS3 и JavaScript для реализации агрегатора такси обусловлен их мощными возможностями в создании современных пользовательских интерфейсов, кросс-платформенностью, поддержкой сообщества и обширными ресурсами. Это позволяет разработчикам создавать эстетически привлекательные и интерактивные интерфейсы для удобства пользователей.

Bootstrap 5 - это популярный фреймворк для разработки веб-интерфейсов, основанный на HTML, CSS и JavaScript. Вот несколько преимуществ и причин выбора Bootstrap 5 для реализации агрегатора такси:

1 Готовые компоненты и макеты: Bootstrap 5 предоставляет широкий набор готовых компонентов и макетов, таких как кнопки, формы, навигационные панели, модальные окна, карусели и многое другое. Это позволяет разработчикам быстро и легко создавать стильные и современные интерфейсы без необходимости разработки с нуля.

2 Адаптивный дизайн: Bootstrap 5 обеспечивает адаптивный дизайн, который автоматически адаптируется к различным размерам экранов и устройствам, включая мобильные устройства. Это позволяет создавать веб-интерфейсы, которые выглядят хорошо и функционируют эффективно на любом устройстве.

3 Гибкость и настраиваемость: Bootstrap 5 предоставляет множество настраиваемых опций и переменных, которые позволяют разработчикам легко настроить внешний вид и поведение компонентов в соответствии с требованиями проекта. Это дает возможность создавать уникальные дизайны и адаптировать фреймворк под конкретные потребности приложения.

4 Кросс-браузерная совместимость: Bootstrap 5 обеспечивает совместимость с различными веб-браузерами, включая последние версии Chrome, Firefox, Safari, Edge и других популярных браузеров.

5 Поддержка и активное сообщество: Bootstrap имеет огромную базу пользователей и активное сообщество разработчиков. Это означает, что есть множество ресурсов, документации, тем оформления, плагинов и поддержки,

которые могут помочь в разработке и решении возникающих проблем.

Выбор Bootstrap 5 для реализации агрегатора такси обусловлен его готовыми компонентами, адаптивным дизайном, гибкостью, кросс-браузерной совместимостью и поддержкой сообщества

Entity Framework (EF) - это фреймворк объектно-реляционного отображения (ORM), разработанный для работы с базами данных. Он предоставляет удобный способ взаимодействия с базой данных через объектно-ориентированный подход. Вот несколько преимуществ и причин выбора EF вместе с SQLite для реализации агрегатора такси:

1 Упрощенное взаимодействие с базой данных: EF предоставляет простой и интуитивно понятный API для работы с базой данных. Он позволяет разработчикам работать с данными в виде объектов и коллекций, а не напрямую с SQL-запросами и таблицами. Это делает процесс взаимодействия с базой данных более удобным и эффективным.

2 Автоматическое отображение объектов на таблицы: EF обеспечивает автоматическое отображение объектов приложения на соответствующие таблицы в базе данных. Разработчикам не нужно писать SQL-запросы или создавать схему базы данных вручную. Это упрощает и ускоряет процесс разработки и поддержки приложения.

3 Работа с различными базами данных: EF поддерживает несколько провайдеров баз данных, включая SQLite, SQL Server, MySQL, PostgreSQL и другие. Это позволяет разработчикам использовать EF с разными базами данных, в зависимости от требований проекта. В случае выбора SQLite в качестве базы данных для агрегатора такси, использование EF позволяет удобно и эффективно работать с этой легкой и компактной базой данных.

4 Управление связями между объектами: EF позволяет управлять связями между объектами в приложении, предоставляя механизмы для определения отношений "один-ко-многим", "многие-ко-многим" и других типов связей. Это облегчает работу с данными, требующими связи между различными таблицами.

5 Повышение производительности и безопасности: EF предлагает возможности кэширования данных, оптимизации запросов и управления транзакциями, что способствует повышению производительности приложения. Он также предоставляет механизмы защиты от атак типа SQL-инъекций и других уязвимостей, обеспечивая безопасность взаимодействия с базой данных.

Выбор EF вместе с SQLite для агрегатора такси обусловлен удобством и простотой работы с данными, автоматическим отображением объектов на

таблицы, поддержкой различных баз данных и возможностью управления связями между объектами. Кроме того, EF предоставляет механизмы для повышения производительности и обеспечения безопасности взаимодействия с базой данных.

Технология внедрения зависимостей (Dependency Injection, DI) — это подход в разработке программного обеспечения, который позволяет управлять зависимостями между компонентами приложения. Вместо того чтобы явно создавать экземпляры зависимых объектов внутри класса, эти зависимости предоставляются извне.

Вот несколько преимуществ использования технологии внедрения зависимостей:

1 Разрешение зависимостей: DI позволяет автоматически разрешать зависимости компонентов приложения. Вместо того, чтобы каждый раз создавать экземпляр зависимого объекта внутри класса, DI-контейнер берет на себя задачу создания и управления зависимостями. Это позволяет упростить код, улучшить его читаемость и снизить связанность между классами.

2 Гибкость и переиспользование: DI способствует гибкости и переиспользованию компонентов приложения. Зависимости могут быть легко заменены или изменены, без внесения изменений в код классов, которые их используют. Это позволяет создавать более модульные и расширяемые системы.

3 Тестирование: DI упрощает тестирование приложения. Зависимости могут быть заменены на моки или заглушки для проведения юнит-тестирования без необходимости запуска всей системы. Такой подход позволяет легко изолировать и тестировать отдельные компоненты приложения.

4 Разделение ответственностей: Использование DI способствует разделению ответственностей в коде. Каждый компонент отвечает только за свою специфическую функциональность, а внедрение зависимостей обеспечивает связывание этих компонентов воедино. Это способствует созданию более чистой и модульной архитектуры приложения.

5 Управление жизненным циклом: DI позволяет управлять жизненным циклом объектов. Контейнер внедрения зависимостей может создавать объекты в единственном экземпляре (singleton) или создавать новый экземпляр при каждом запросе. Это дает контроль над временем жизни объектов и может помочь в управлении ресурсами.

При выборе технологии внедрения зависимостей, одним из популярных вариантов является использование фреймворков, таких как ASP.NET Core,

который предоставляет встроенную поддержку DI и который был как раз использован для написания агрегатора такси. Выбор DI позволяет создавать более гибкие, расширяемые и тестируемые приложения, улучшает их архитектуру и облегчает поддержку в долгосрочной перспективе.

Bing Maps API - это сервис картографии и геопространственной аналитики, предоставляемый Microsoft. Он предлагает набор инструментов и функций для встраивания интерактивных карт и геопространственных данных в веб-приложения.

История зарождения Bing Maps API уходит в корни Virtual Earth, предшественника Bing Maps, который был запущен в 2005 году. Он был разработан компанией Microsoft в ответ на растущую потребность в картографических сервисах и пространственной информации.

Bing Maps API обладает несколькими преимуществами, которые делают его привлекательным для разработчиков и бизнес-пользователей:

1 Широкий функционал: Bing Maps API предоставляет множество функций для работы с картами и геопространственными данными. Это включает в себя отображение карт, нанесение маркеров, поиск местоположений, маршрутизацию, анализ геоданных и многое другое. Богатый набор функций позволяет создавать разнообразные приложения, включая агрегаторы такси.

2 Интеграция с другими сервисами Microsoft: Bing Maps API интегрируется с другими сервисами Microsoft, такими как Azure, Power BI и Office 365.

3 Гибкость в настройке и настройки: Bing Maps API предлагает гибкие настройки и настраиваемые параметры, позволяя разработчикам адаптировать карты и функциональность под конкретные требования приложений. Это включает выбор стилей карты, настройку маркеров, добавление пользовательских данных и другие возможности.

4 Разнообразие платформ и языков: Bing Maps API поддерживает различные платформы разработки, включая .NET, JavaScript, Android, iOS и другие. Он также предоставляет SDK и библиотеки для удобной интеграции с различными языками программирования.

В сравнении с другими картографическими сервисами, Bing Maps API обладает удобным интерфейсом, широкими функциональными возможностями и интеграцией с экосистемой Microsoft. Он является конкурентоспособным решением для разработки приложений, связанных с картографией и геопространственными данными.

### **3 ПАТТЕРНЫ ПРОГРАММИРОВАНИЯ, ИСПОЛЬЗУЕМЫЕ В РАЗРАБОТКЕ ПРИЛОЖЕНИЯ**

Технология MVC (Model-View-Controller) является архитектурным шаблоном разработки программного обеспечения, который был применен в фреймворках ASP.NET и ASP.NET Core для создания веб-приложений. Он предлагает разделение бизнес-логики, пользовательского интерфейса и управления состоянием приложения на три компонента: модель (Model), представление (View) и контроллер (Controller).

1 Модель (Model) представляет бизнес-логику и данные приложения. Она отвечает за обработку и хранение данных, а также за выполнение операций, связанных с бизнес-процессами приложения. Модель может включать классы, объекты, сервисы доступа к данным, валидацию данных и другие элементы, необходимые для обработки бизнес-логики.

2 Представление (View) отвечает за отображение данных пользователю. Оно представляет собой пользовательский интерфейс приложения, который может быть представлен в виде веб-страницы, шаблона или другого способа визуализации информации. Представление использует данные, полученные от модели, для формирования и отображения пользовательского интерфейса. Оно обычно содержит HTML, CSS и другие элементы для создания визуального представления данных.

3 Контроллер (Controller) обрабатывает входящие запросы от пользователя, взаимодействует с моделью и определяет, какое представление должно быть отображено в ответ на запрос. Он является посредником между пользователем, моделью и представлением. Контроллер обрабатывает данные, полученные от пользователя, вызывает соответствующие методы модели для обработки данных и определяет, какое представление должно быть отображено в результате выполнения запроса.

Вот несколько преимуществ и причин выбора технологии MVC для реализации агрегатора такси:

1 Разделение ответственности: MVC разделяет компоненты приложения на отдельные слои, каждый из которых отвечает за свои функциональности. Модель отвечает за бизнес-логику и обработку данных, представление отображает информацию пользователю, а контроллер обрабатывает входящие запросы и координирует взаимодействие между моделью и представлением. Это делает код более понятным, поддерживаемым и масштабируемым.

2 Гибкость и расширяемость: MVC позволяет легко добавлять новые функциональности и изменять существующую логику без влияния на другие



компоненты приложения. Каждый компонент имеет свою собственную задачу и может быть заменен или модифицирован независимо от других компонентов. Это упрощает разработку и поддержку приложения, особенно при необходимости внесения изменений или добавления новых функций.

3 Улучшенная тестируемость: Модель, представление и контроллер могут быть легко тестируемыми независимо друг от друга. Таким образом, разработчики могут проводить модульное тестирование каждого компонента для обеспечения его правильной работы. Это способствует созданию надежного и стабильного приложения.

4 Поддержка множества клиентских платформ: MVC позволяет создавать веб-приложения, которые могут быть использованы на различных клиентских платформах, включая компьютеры, мобильные устройства и планшеты. Это обеспечивает гибкость и доступность приложения для широкой аудитории пользователей.

5 Четкое разделение ролей: MVC позволяет разделить роли разработчиков на различные задачи. Например, дизайнеры могут работать над представлениями, бизнес-логика может быть реализована разработчиками модели, а контроллеры могут быть созданы разработчиками, ответственными за обработку запросов и управление потоком данных. Это улучшает эффективность работы команды разработки.

Таким образом, выбор технологии MVC для реализации агрегатора такси обусловлен ее способностью к разделению ответственности, гибкостью и расширяемостью, улучшенной тестируемостью, поддержкой множества клиентских платформ и четким разделением ролей в команде разработки. Это обеспечивает эффективность и надежность разработки приложения.

Паттерн Repository (Репозиторий) является одним из популярных паттернов проектирования, который применяется при разработке приложений на ASP.NET и других платформах. Он используется для абстрагирования доступа к данным, обеспечивая единый интерфейс для работы с различными источниками данных, такими как базы данных, файловые системы, веб-сервисы и другие.

Основные преимущества паттерна Repository включают:

1 Разделение бизнес-логики от доступа к данным: Паттерн Repository помогает отделить бизнес-логику вашего приложения от деталей доступа к данным. Он предоставляет единый интерфейс для работы с данными, скрывая сложности и специфику конкретного источника данных. Это упрощает тестирование, поддержку и изменение кода вашего приложения.

2 Централизованное управление доступом к данным: Repository

предоставляет централизованный механизм для выполнения операций с данными, таких как создание, чтение, обновление и удаление (CRUD). Он инкапсулирует логику доступа к данным внутри репозитория, что делает его удобным и управляемым местом для работы с данными.

3 Улучшенная тестируемость: Паттерн Repository облегчает тестирование бизнес-логики вашего приложения путем предоставления заменяемой реализации репозитория. Вы можете создать заглушки (mock) или подставные объекты для тестирования, что делает тестирование легким и изолированным от зависимостей от реальных источников данных.

4 Повторное использование кода: Repository позволяет использовать один и тот же интерфейс доступа к данным в разных частях вашего приложения. Вы можете повторно использовать репозиторий в различных контекстах, что способствует сокращению дублирования кода и улучшает общую поддерживаемость приложения.

Паттерн Unit of Work (Единица работы) является одним из популярных паттернов проектирования, который используется при разработке приложений на ASP.NET и других платформах. Он используется для управления транзакциями и координирования операций записи/чтения в базу данных в рамках одной логической операции.

Основная цель паттерна Unit of Work заключается в следующем:

1 Управление транзакциями: Unit of Work позволяет группировать операции записи/чтения в базу данных в рамках одной транзакции. Это обеспечивает атомарность операций, что означает, что либо все операции успешно выполняются и сохраняются, либо ни одна из них не выполняется. Такой подход гарантирует целостность данных и предотвращает возникновение неконсистентных состояний.

2 Единый контекст работы с данными: Unit of Work предоставляет единый контекст, в рамках которого выполняются операции записи/чтения. Это означает, что все изменения данных, выполненные в рамках Unit of Work, будут сохранены или отменены вместе. Это упрощает кодирование бизнес-логики и уменьшает вероятность ошибок.

3 Управление жизненным циклом объектов: Unit of Work отслеживает изменения, внесенные в объекты данных в рамках текущей операции. Он автоматически обновляет состояние объектов и выполняет соответствующие операции сохранения или обновления в базе данных при вызове метода сохранения (Save).

4 Улучшенная производительность: Unit of Work позволяет сократить количество операций чтения/записи в базу данных. Он группирует изменения

данных и выполняет операцию сохранения в базу данных одним запросом, что может значительно повысить производительность приложения.

Преимущество использования паттернов Repository и Unit of Work вместе заключается в том, что они взаимодополняют друг друга и обеспечивают более гибкую и эффективную работу с данными в приложении. Вот несколько основных преимуществ их совместного использования:

1 Разделение ответственностей: Паттерн Repository отвечает за абстрагирование доступа к данным и предоставляет единый интерфейс для работы с различными источниками данных. Он скрывает сложность и специфику работы с данными, что позволяет более удобно разрабатывать и поддерживать код приложения. Паттерн Unit of Work, в свою очередь, управляет транзакциями и координирует операции записи/чтения данных в рамках единого контекста. Вместе они позволяют разделить ответственность между уровнями доступа к данным и управлением транзакциями.

2 Упрощенная работа с данными: Комбинированное использование паттернов Repository и Unit of Work обеспечивает единый и удобный способ работы с данными в приложении. Repository предоставляет методы для выполнения операций чтения/записи сущностей, а Unit of Work контролирует сохранение изменений и выполнение операций в рамках транзакции. Это упрощает разработку, тестирование и поддержку кода приложения.

3 Целостность данных: Паттерн Unit of Work позволяет группировать операции записи/чтения в рамках одной транзакции. Это обеспечивает целостность данных и гарантирует, что либо все изменения будут успешно сохранены, либо ни одно из них не будет применено. Такой подход предотвращает возникновение неконсистентных состояний в базе данных и обеспечивает атомарность операций.

4 Гибкость и расширяемость: Использование паттернов Repository и Unit of Work позволяет легко изменять и заменять источники данных или реализацию доступа к данным без влияния на бизнес-логику приложения. Вы можете легко добавить новый репозиторий или внести изменения в Unit of Work, не затрагивая другие компоненты приложения. Это делает приложение более гибким и расширяемым.

В целом, совместное использование паттернов Repository и Unit of Work способствует лучшей структурированности кода, улучшению производительности, упрощению работы с данными и обеспечению целостности данных в приложении агрегатора такси на платформе ASP.NET.

## 4 ФУНКЦИОНАЛЬНЫЕ ВОЗМОЖНОСТИ ПРОГРАММЫ

Все возможные варианты использования функций проекта агрегатор такси пользователем в зависимости от его роли (администратор, клиент) продемонстрированы на use-case диаграмме (приложение В).

При первом использовании агрегатора такси, незарегистрированный пользователь попадает на главную страницу (рисунок 1), где у него есть возможность перейти на страницу регистрации или страницу входа в аккаунт, если такой существует. Также есть возможность использовать контакты (рисунок 2), которые указаны внизу страниц веб-приложения.

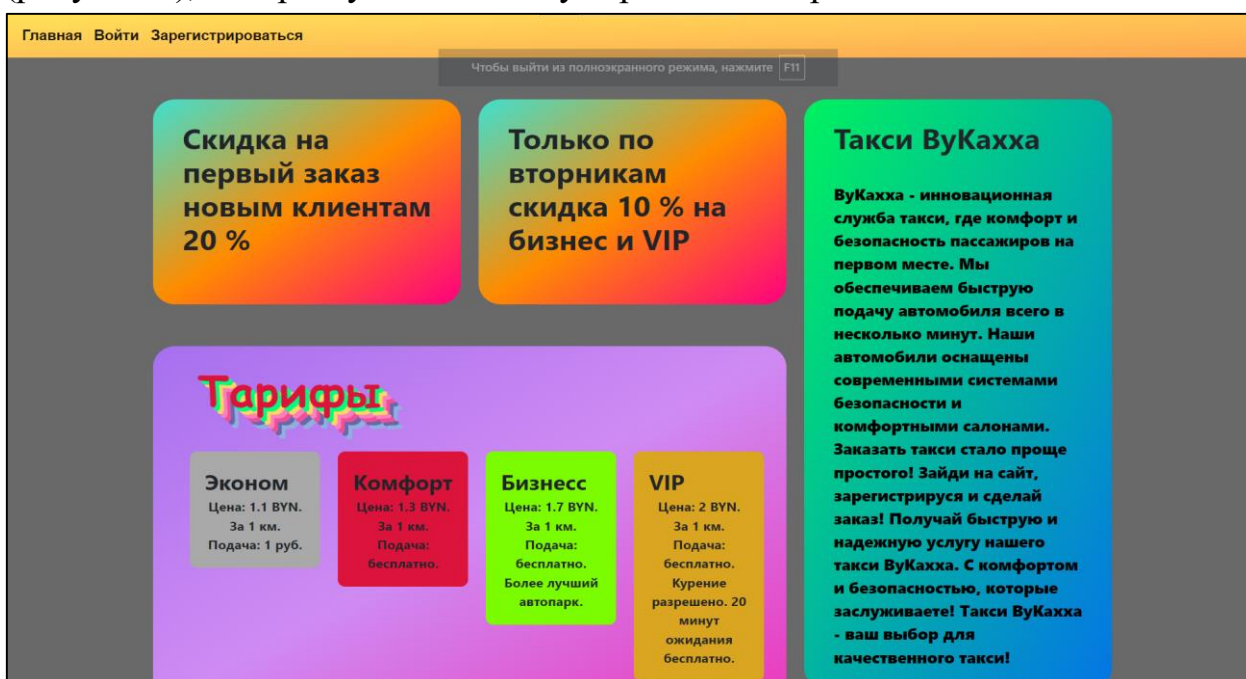


Рисунок 1 – Главная страница веб-приложения

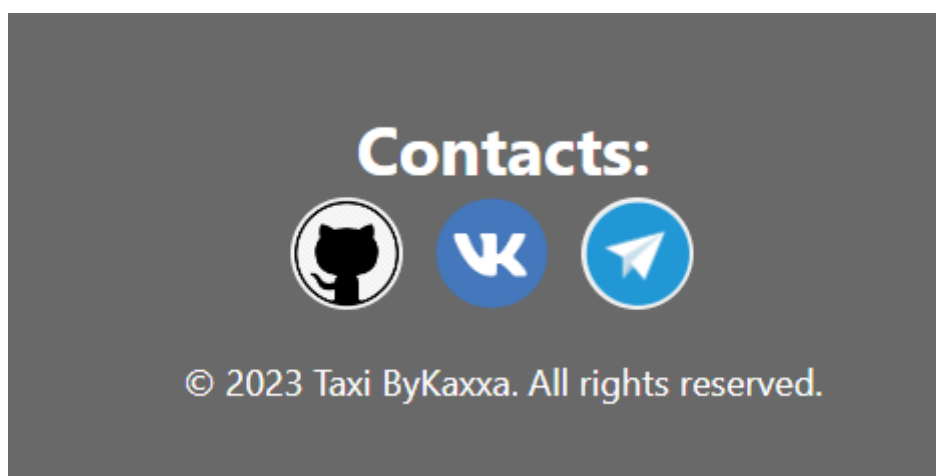


Рисунок 2 – Контакты

При нажатии на кнопку логина, пользователю будет предоставлена страница авторизации (рисунок 3).

Рисунок 3 – Страница авторизации

При нажатии на кнопку зарегистрироваться или при нажатии кнопки Sign up на страницу авторизации (рисунок 3), пользователя перекинет на страницу регистрации (рисунок 4).

Рисунок 4 – Страница регистрации

После успешной регистрации или входа в аккаунт (за клиента), пользователь может совершить заказ такси (рисунок 5).

Рисунок 5 – Страница оформления заказа такси

После успешного оформления заказа, пользователя перекинет на страницу подтверждения заказа (рисунок 6).

Рисунок 6 – Страница подтверждения заказа

На странице подтверждения заказа пользователь увидит всю дополнительную информацию о заказе, такую как цена, расстояние, время, карту маршрута. Так же пользователю будет предложено подтвердить заказ или отменить его. Если пользователь решит отменить заказ, то его перекинет снова на страницу оформления заказа (рисунок 5), а если пользователь подтвердит заказ, то его перекинет на главную страницу (рисунок 1) и заказ будет успешно принят.

Пользователю также доступна кнопка просмотра всех его совершенных заказов (рисунок 7).

Главная

Сделать заказ

История заказов

Профиль

Выйти


Номер	Описание	Старт	Финиш	Тариф	Цена	Дистанция	Время	Оплата	Кол-во пассажиров	Гендер водителя
1	Поездка в Москву бизнес класс...	Минск	Москва	Business	1212,16	713,038	07:51:47	CreditCard	4	NoMetter

Рисунок 7 – Страница просмотра всех заказов

При переходе на страницу профиля (рисунок 8), пользователю станут доступны функции редактирования профиля, выбор фото и удаления профиля.

Главная | Сделать заказ | История заказов | Профиль | Выйти

### Изменение профиля



**ИЗОБРАЖЕНИЕ  
ВРЕМЕННО  
ОТСУТСТВУЕТ**

Выберите файл | Файл не выбран

Сохранить фото

Удалить профиль

Name: TestName | Surname: TestSurname

Email: Test@mail.ru

PhoneNumber: +123123123

Age: 22

Login: test

Password: 123

Сохранить

Рисунок 8 – Страница профиля

При нажатии на кнопку выхода, пользователя перекинет на страницу на главную страницу (рисунок 1) и он благополучно выйдет из своего аккаунта.

При входе в аккаунт администратора, ему доступны просмотр всех пользователей (рисунок 9), просмотр всех клиентов (рисунок 10), добавление пользователя (рисунок 11), изменения пользователя (рисунок 12) и просмотр всех заказов пользователя (рисунок 7).

Главная										
Список клиентов Админ панель Добавить пользователя Выйти										
Id	Login	Password	Role	Name	Surname	Age	Email	Phone	Delete	Update
2	Jeka	123	Client	Jeka	Каха	22	Каха29@mail.ru	123123123	Delete	Update
21	lera	123	Admin	Лера	Да	50	Lera@mail.ru	123123	Delete	Update
22	test	123	Client	TestName	TestSurname	22	Test@mail.ru	+123123123	Delete	Update

Рисунок 9 – Страница просмотра всех пользователей

Главная									
Список клиентов Админ панель Добавить пользователя Выйти									
Id	Login	Password	Name	Surname	Age	Email	Phone	Заказы	
2	Jeka	123	Jeka	Каха	22	Каха29@mail.ru	123123123	Заказы	
22	test	123	TestName	TestSurname	22	Test@mail.ru	+123123123	Заказы	

Рисунок 10 – Страница просмотра всех клиентов



The screenshot shows a web application interface with a yellow header bar containing navigation links: 'Главная', 'Список клиентов', 'Админ панель', 'Добавить пользователя', and 'Выйти'. The main content area has a dark gray background with a central light blue rounded rectangle titled 'Создание пользователя'. Inside this rectangle, there are several input fields: 'Name' and 'Surname' (side-by-side), 'Email', 'PhoneNumber', 'Age', 'Login', 'Password', and 'PasswordConfirm' (stacked vertically). Below these is a dropdown menu for 'Роль' with 'Пользователь' selected. At the bottom of the form is a gray button labeled 'Добавить пользователя'.

Рисунок 11 – Страница добавления пользователя

The screenshot shows a web application interface similar to the previous one, with the same yellow header bar. The main content area has a dark gray background with a central light blue rounded rectangle titled 'Изменение пользователя'. Inside this rectangle, the input fields are pre-filled with test data: 'Name' (TestName), 'Surname' (TestSurname), 'Email' (Test@mail.ru), 'PhoneNumber' (+123123123), 'Age' (22), 'Login' (test), and 'Password' (123). The 'Роль' dropdown menu also has 'Пользователь' selected. At the bottom of the form is a gray button labeled 'Обновить данные'.

Рисунок 12 – Страница изменения данных пользователя

## 5 АРХИТЕКТУРА РАЗРАБАТЫВАЕМОЙ ПРОГРАММЫ

Как отмечалось ранее, в основе веб-приложения агрегатор такси лежат основные принципы объектно-ориентированного программирования (ООП), принципы SOLID.

Архитектура разрабатываемой программы агрегатора такси будет основана на многоуровневой архитектуре, которая включает следующие уровни: DAL (Data Access Layer), Domain, WEB (Presentation Layer) и BLL (Business Logic Layer). Каждый уровень выполняет определенные функции и отвечает за определенные аспекты разработки.

1 Уровень доступа к данным (Data Access Layer, DAL): Этот уровень отвечает за взаимодействие с базой данных или другими источниками данных. Здесь располагаются компоненты, отвечающие за выполнение операций чтения и записи данных. DAL предоставляет абстракцию доступа к данным и скрывает детали работы с конкретными источниками данных. Он включает в себя репозитории, которые предоставляют методы для выполнения операций CRUD (Create, Read, Update, Delete) над сущностями приложения.

2 Уровень бизнес-логики (Business Logic Layer, BLL): На этом уровне содержится бизнес-логика приложения, которая определяет правила и процессы работы с данными. Здесь располагаются компоненты, отвечающие за обработку и манипуляцию данными, а также за принятие бизнес-решений. BLL обычно содержит сервисы или менеджеры, которые предоставляют методы для выполнения сложных операций, комбинируя функциональность DAL.

3 Уровень доменной модели (Domain Layer): Этот уровень отражает структуру и поведение основных сущностей предметной области приложения. Здесь определены классы, представляющие объекты, с которыми работает приложение (например, таксисты, клиенты, заказы и т.д.). Доменная модель отражает основные понятия и связи между ними в предметной области и может включать в себя валидацию данных и бизнес-правила.

4 Уровень веб-интерфейса (Web Layer): Это уровень отвечает за представление данных и взаимодействие с пользователем. Здесь располагаются компоненты, связанные с веб-интерфейсом приложения, такие как контроллеры (для обработки запросов пользователя), представления (для отображения данных) и модели (для передачи данных между представлением и контроллером). Web Layer связывает пользовательский интерфейс с бизнес-логикой, обрабатывает входящие запросы и отображает данные пользователю.

## Преимущества многоуровневой архитектуры:

1 Разделение ответственностей: Многоуровневая архитектура позволяет ясно разделить функциональность приложения на логические компоненты и слои, что облегчает понимание и поддержку кода. Каждый уровень имеет свою специфическую задачу и ответственность.

2 Повторное использование кода: Хорошо структурированная многоуровневая архитектура способствует повторному использованию кода. Например, DAL может быть использован в различных частях приложения для доступа к данным, а BLL может предоставлять повторно используемую бизнес-логику для разных сценариев.

3 Тестирование и поддержка: Разделение кода на уровни упрощает тестирование отдельных компонентов и модулей приложения. Каждый уровень может быть протестирован отдельно, что способствует обнаружению и исправлению ошибок. Кроме того, при поддержке и расширении приложения, изменения могут быть внесены в отдельные уровни без затрагивания других.

4 Масштабируемость: Многоуровневая архитектура обеспечивает гибкость и масштабируемость приложения. Каждый уровень может быть масштабирован и оптимизирован отдельно, что позволяет легко адаптировать приложение к растущим требованиям и нагрузке.

В целом, многоуровневая архитектура способствует чистоте кода, улучшению модульности, упрощению разработки и поддержки приложения агрегатора такси на платформе ASP.NET. Она помогает разделить ответственности между различными компонентами и обеспечить более гибкую и эффективную работу приложения.

## ЗАКЛЮЧЕНИЕ

В ходе разработки агрегатора такси на платформе ASP.NET были использованы различные технологии и подходы, которые сыграли важную роль в создании мощного и функционального веб-приложения.

ASP.NET MVC стал основой разработки, обеспечивая разделение задач между моделью, представлением и контроллером. Это позволило создать структурированное приложение, где каждый компонент отвечает за свои функции, что способствует легкости разработки, поддержки и масштабируемости.

Entity Framework был использован для работы с базой данных, позволяя эффективно взаимодействовать с данными и обеспечивая удобство работы с базой данных SQLite. Благодаря Entity Framework, операции чтения и записи данных были упрощены и оптимизированы.

JavaScript, HTML и CSS были использованы для создания интерактивного пользовательского интерфейса. Эти языки разметки и стилей позволили создать привлекательный и отзывчивый дизайн, обеспечивая удобство использования и хороший пользовательский опыт.

Использование Razor Page позволило создавать динамические и гибкие представления, где логика и разметка объединены в одном файле. Это упростило процесс разработки и поддержки представлений.

Bootstrap5 предоставил готовые компоненты и стили, которые обеспечили адаптивность и современный внешний вид приложения. Это позволило создать приятный пользовательский интерфейс, который легко адаптируется к различным устройствам и разрешениям экранов.

N-Layer architecture, в сочетании с паттернами Unit of Work и Repository, способствовали созданию модульной и гибкой структуры приложения. Разделение слоев приложения позволило легко масштабировать и модифицировать компоненты, облегчая их тестирование и поддержку.

В результате использования этих технологий и подходов был создан функциональный и удобный агрегатор такси, который позволяет клиентам с легкостью заказывать такси и администраторам эффективно управлять информацией и пользователями. Проект соответствует требованиям объектно-ориентированного программирования и демонстрирует применение современных технологий в разработке веб-приложений.

## СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

- [1] Рихтер Дж. CLR via C#. Программирование на платформе Microsoft .NET Framework 4.5 на языке C#. 4-е изд. — СПб.: Питер, 2013. — 896 с.: ил. — (Серия «Мастер-класс»).
- [2] Троелсен, Эндрю, Джепикс, Филипп. Язык программирования C# 7 и платформы .NET и .NET Core, 8-е изд. : Пер. с англ. — СПб. : ООО «Диалектика», 2018 — 1328 с.
- [3] Макконнелл С. Совершенный код. Мастер-класс / Пер. с англ. — М. : Издательство «Русская редакция», 2010. — 896 стр. : ил.
- [4] Лок Э. ASP.Net Core в действии / пер. с англ. Д. А. Беликова. — М. : ДМК Пресс, 2021. – 906 с.: ил.
- [5] Дронов В. А. Д75 HTML 5, CSS 3 и Web 2.0. Разработка современных Web-сайтов. — СПб. : БХВ-Петербург, 2011. — 416 с.: ил. — (Профессиональное программирование)
- [6] Metanit [Электронный ресурс]. – Режим доступа : <https://metanit.com/>
- [7] Bootstrap – документация [Электронный ресурс]. – Режим доступа : <https://getbootstrap.com/>.
- [8] ASP.NET MVC - Microsoft Learn [Электронный ресурс]. – Режим доступа : <https://learn.microsoft.com/en-us/aspnet/mvc/>.
- [9] Entity Framework documentation [Электронный ресурс]. – Режим доступа : <https://learn.microsoft.com/en-us/ef/>.
- [10] W3Schools [Электронный ресурс]. – Режим доступа : <https://www.w3schools.com/>.

## ПРИЛОЖЕНИЕ А

### (обязательное)

### Исходный код программы

#### Листинг 1 – Контроллер администратора

```
[Area("Admin")]
public class AdminController : Controller
{
    private readonly IClientService _clientService;
    private readonly ITaxiOrderService _taxiOrderService;

    public AdminController(IClientService clientService, ITaxiOrderService
taxiOrderService)
    {
        _clientService = clientService;
        _taxiOrderService = taxiOrderService;
    }

    #region HttpGet
    [HttpGet]
    public IActionResult Index()
    {
        return View(_clientService.GetAllClients().Result.Data);
    }

    [HttpGet]
    public IActionResult AddClient() { return View(); }

    [HttpGet]
    public IActionResult UpdateClient(int id)
    {
        return View(_clientService.FirstOrDefault(x => x.Id ==
id).Result.Data);
    }

    [HttpGet]
    public IActionResult DeleteClient() { return View(); }

    [HttpGet]
    public IActionResult AdminPanel() { return
View(_clientService.GetAllClients().Result.Data); }

    [HttpGet]
    public IActionResult FindClient() { return View(); }

    [HttpGet]
    public IActionResult GetTaxiOrdersForAdmin(int userId)
    {
        return
View(_taxiOrderService.GetAllClientTaxiOrders(userId).Result.Data);
    }
    #endregion

    #region HttpPost
    [HttpPost]
    public async Task<IActionResult>
AddClient(TaxiApplication.Domain.Entity.Client client)
    {
        if (ModelState.IsValid)
```

```

        {
            var response = await _clientService.AddClient(client);

            if (response.StatusCode == Domain.Enum.StatusCode.OK)
            {
                return RedirectToAction("Index");
            }

            ModelState.AddModelError("", response.Description);
        }
        return View(client);
    }
    [HttpPost]
    public async Task<IActionResult> DeleteClient(int id)
    {
        var response = await _clientService.DeleteClient(id);

        if (response.StatusCode == Domain.Enum.StatusCode.OK)
        {
            return RedirectToAction("Index");
        }
        return View();
    }
    [HttpPost]
    public async Task<IActionResult> DeleteOrder(int id)
    {
        var response = await _taxiOrderService.DeleteTaxiOrder(id);

        if (response.StatusCode == Domain.Enum.StatusCode.OK)
        {
            return RedirectToAction("Index");
        }
        return RedirectToAction("Index", "Home", new { area = "" });
    }
    [HttpPost]
    public async Task<IActionResult>
    UpdateClient(TaxiApplication.Domain.Entity.Client client)
    {
        var response = await _clientService.UpdateClient(client);

        if (response.StatusCode == Domain.Enum.StatusCode.OK)
        {
            return RedirectToAction("AdminPanel");
        }
        return View();
    }
}

```

## Листинг 2 – Контроллер клиента

```

[Area("Client")]
public class ClientController : Controller
{
    ITaxiOrderService _taxiOrderService { get; set; }
    IMapService _mapService { get; set; }
    IClientService _clientService { get; set; }
    public ClientController(ITaxiOrderService taxiOrderService, IMapService
    mapService, IClientService clientService)
    {
        _taxiOrderService = taxiOrderService;
        _mapService = mapService;
        _clientService = clientService;
    }
    private bool CheckUserStatus()
    {
        int ID =
        int.Parse(User.FindFirst(ClaimTypes.NameIdentifier)?.Value!);
    }
}

```

```

        var All_Clients = (_clientService.GetAllClients()).Result.Data;
        var client = All_Clients!.FirstOrDefault(c => c.Id == ID);

        return client is null ? true : false;
    }
    [HttpGet]
    public async Task<IActionResult> ProfileEdit()
    {
        if (CheckUserStatus() is true) return RedirectToAction("Logout",
"Account", new { area = "" });
        var client = (await
_clientService.GetClient(int.Parse(User.FindFirst(ClaimTypes.NameIdentifier)?
.Value!))).Data;
        return View(client);
    }
    [HttpPost]
    public async Task<IActionResult> ProfileEdit(Client client)
    {
        if (CheckUserStatus() is true) return RedirectToAction("Logout",
"Account", new { area = "" });

        client.Profile.Photo = (await
_clientService.GetClient(int.Parse(User.FindFirst(ClaimTypes.NameIdentifier)?
.Value!))).Data!.Profile.Photo;
        var response = await _clientService.UpdateClient(client);

        if (response.StatusCode == Domain.Enum.StatusCode.OK)
        {
            return View(client);
        }
        return View(client/*Страница ошибки*/);
    }
    [HttpPost]
    public async Task<IActionResult> UploadPhoto(IFormFile photoFile)
    {
        if (CheckUserStatus() is true) return RedirectToAction("Logout",
"Account", new { area = "" });
        try
        {
            if (photoFile != null && photoFile.Length > 0)
            {
                byte[] photoBytes;
                using (var memoryStream = new MemoryStream())
                {
                    await photoFile.CopyToAsync(memoryStream);
                    photoBytes = memoryStream.ToArray();
                }
                var client = (await
_clientService.GetClient(int.Parse(User.FindFirst(ClaimTypes.NameIdentifier)?
.Value!))).Data;

                client!.Profile.Photo = photoBytes;
                await _clientService.UpdateClient(client);
            }

            return RedirectToAction("ProfileEdit");
        }
        catch (Exception ex)
        {
            await
Console.Out.WriteLineAsync($"[ClientController.UploadPhoto] error:
{ex.Message}");
            return RedirectToAction("ProfileEdit");
        }
    }

```



```

    }
}

[HttpPost]
public async Task<IActionResult> DeleteProfile(Client client)
{
    if (CheckUserStatus() is true) return RedirectToAction("Logout",
"Account", new { area = "" });
    try
    {
        await _clientService.DeleteClient(client.Id);
        return RedirectToAction("Logout", "Account", new { area = ""
});
    }
    catch (Exception ex)
    {
        await
Console.Out.WriteLineAsync($"[ClientController.DeleteProfile] error:
{ex.Message}");
        return RedirectToAction("Logout", "Account", new { area = ""
});
    }
}

[HttpGet]
public IActionResult CreateTaxiOrder()
{
    if (CheckUserStatus() is true) return RedirectToAction("Logout",
"Account", new { area = "" });
    return View();
}

[HttpPost]
public async Task<IActionResult> CreateTaxiOrder(TaxiOrderViewModel
taxiOrderViewModel)
{
    if (CheckUserStatus() is true) return RedirectToAction("Logout",
"Account", new { area = "" });
    try
    {
        var request = (await
_mapService.CreateRouteRequest(taxiOrderViewModel)).Data;
        taxiOrderViewModel = (await
_mapService.CollectInfoFromRequest(request, taxiOrderViewModel)).Data!;

        // Проверка на корректность заказа
        if (taxiOrderViewModel.Route.Distance == 0 &&
taxiOrderViewModel.Route.Time.TotalSeconds == 0)
        {return RedirectToAction("CreateTaxiOrder");}
        taxiOrderViewModel.Order.Price = (await
_taxiOrderService.CalculatePrice(taxiOrderViewModel.Order)).Data;
        var json = JsonSerializer.Serialize(taxiOrderViewModel);
        TempData["taxiOrderViewModel_JSON"] = json;
        return RedirectToAction(actionName: "Index", controllerName:
"Map", new { area = "Map" });
    }
    catch (Exception ex)
    {
        await
Console.Out.WriteLineAsync($"[ClientController.CreateTaxiOrder] error:
{ex.Message}");
        return RedirectToAction("CreateTaxiOrder");
    }
}

```

```

    }
    [HttpGet]
    public IActionResult GetTaxiOrders()
    {
        if (CheckUserStatus() is true) return RedirectToAction("Logout",
"Account", new { area = "" });
        return
View(_taxiOrderService.GetAllClientTaxiOrders(int.Parse(User.FindFirst(ClaimT
ypes.NameIdentifier)?.Value!)).Result.Data);
    }
    [HttpPost]
    public async Task<IActionResult> SaveTaxiOrder(string
taxiOrderViewModel)
    {
        if (CheckUserStatus() is true) return RedirectToAction("Logout",
"Account", new { area = "" });
        try
        {
            // Десериализация
            var deserializedObject =
JsonSerializer.Deserialize<TaxiOrderViewModel>(taxiOrderViewModel);

            TaxiOrder taxiOrder = deserializedObject!.Order;
            taxiOrder.CurrentRoute = deserializedObject.Route;
            taxiOrder.ClientId =
int.Parse(User.FindFirst(ClaimTypes.NameIdentifier)?.Value!);

            await _taxiOrderService.AddTaxiOrder(taxiOrder);

            return RedirectToAction("Index", "Home", new { area = "" });
        }
        catch (Exception ex)
        {
            await
Console.Out.WriteLineAsync($"[ClientController.SaveTaxiOrder] error:
{ex.Message}");
            return RedirectToAction("CreateTaxiOrder");
        }
    }
}

```

### Листинг 3 – Контроллер карты

```

[Area("Map")]
public class MapController : Controller
{
    public IActionResult Index()
    {
        try
        {
            // Десериализация TaxiViewModel
            var deserializedViewModel =
JsonSerializer.Deserialize<TaxiOrderViewModel>((string)TempData["taxiOrderVie
wModel_JSON"]);

            //TaxiOrderViewModel viewModel = deserializedObject! as
TaxiOrderViewModel
            TaxiOrderViewModel viewModel = deserializedViewModel!;
            if (viewModel is null) return View();
            return View(viewModel);
        }
        catch (Exception)
        {

```

```

        return View();
    }
}

```

## Листинг 4 – Контроллер аккаунта

```

public class AccountController : Controller
{
    private readonly IAccountService _accountService;

    public AccountController(IAccountService accountService)
    {
        _accountService = accountService;
    }
    #region HttpGet
    [HttpGet]
    public IActionResult Login() => View();
    [HttpGet]
    public IActionResult Registration() => View();
    #endregion
    #region HttpPost
    [HttpPost]
    public async Task<IActionResult> Login(LoginViewModel loginViewModel)
    {
        if (ModelState.IsValid)
        {
            var response = await
            _accountService.LoginAsync(loginViewModel);
            if (response.StatusCode == Domain.Enum.StatusCode.OK)
            {
                await HttpContext.SignInAsync(response.Data!);
                return RedirectToAction("Index", "Home");
            }
            ModelState.AddModelError("", response.Description);
        }
        return View(loginViewModel);
    }

    [HttpPost]
    public async Task<IActionResult> Registration(Client client)
    {
        if (ModelState.IsValid)
        {
            var response = await
            _accountService.RegistrationAsync(client);
            if (response.StatusCode == Domain.Enum.StatusCode.OK)
            {
                await HttpContext.SignInAsync(response.Data!);
                return RedirectToAction("Index", "Home");
            }
            ModelState.AddModelError("", response.Description);
        }
        return View(client);
    }
    #endregion
    [HttpGet]
    public IActionResult AccessDenied() => StatusCode(403, "Forbidden");

    public async Task<IActionResult> Logout()
    {
        await HttpContext.SignOutAsync();
        return RedirectToAction("Index", "Home");
    }
}

```

# ПРИЛОЖЕНИЕ Б (обязательное) Диаграмма классов

ГУИР.153503.048.01

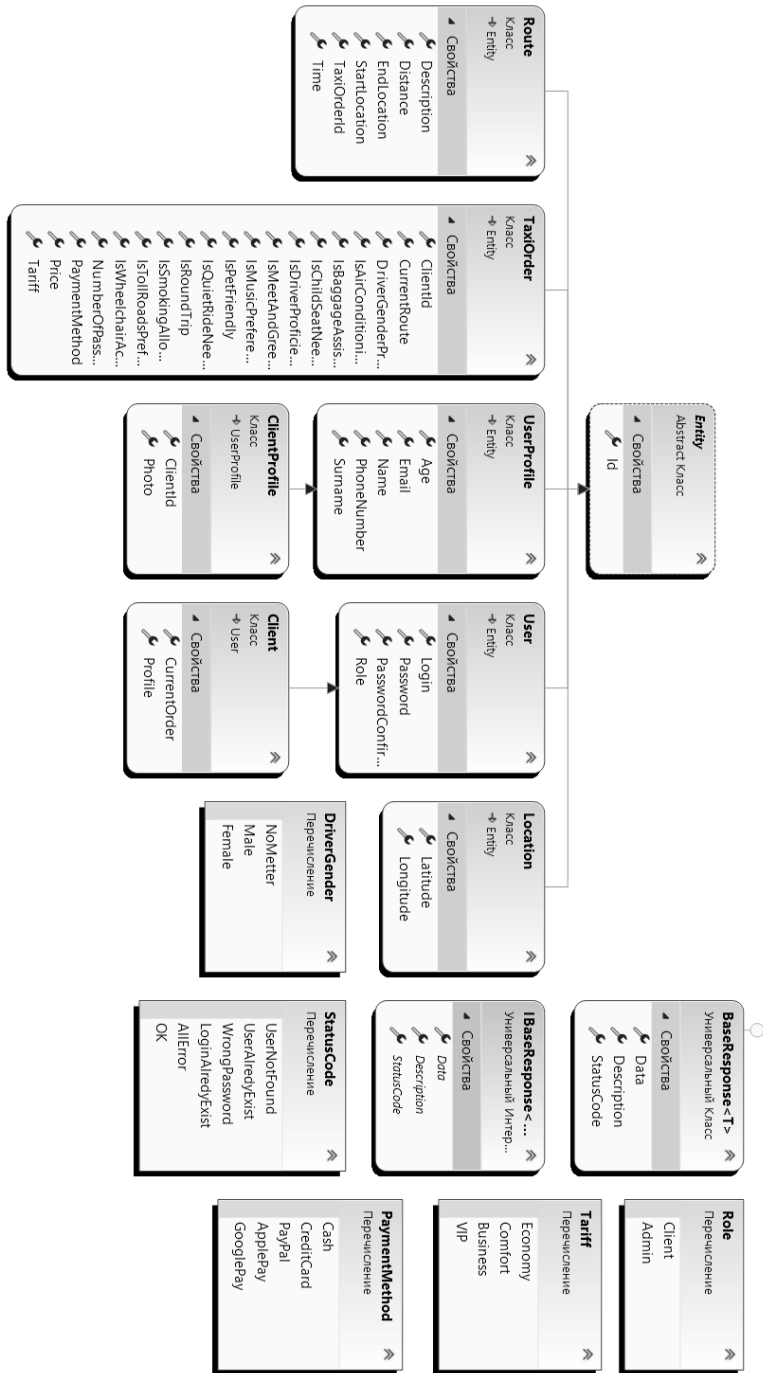


					ГУИР.153503.048.01				
					Диаграмма классов	Лит.	Масса	Масштаб	
Изм.	Лист	№ докум.	Подп.	Дата					
Разраб.		Кахновский				У		1:1	
Прое.		Владимцев							
						Лист 1	Листов 4		
						Кафедра информатики группа 153503			

Формат А4



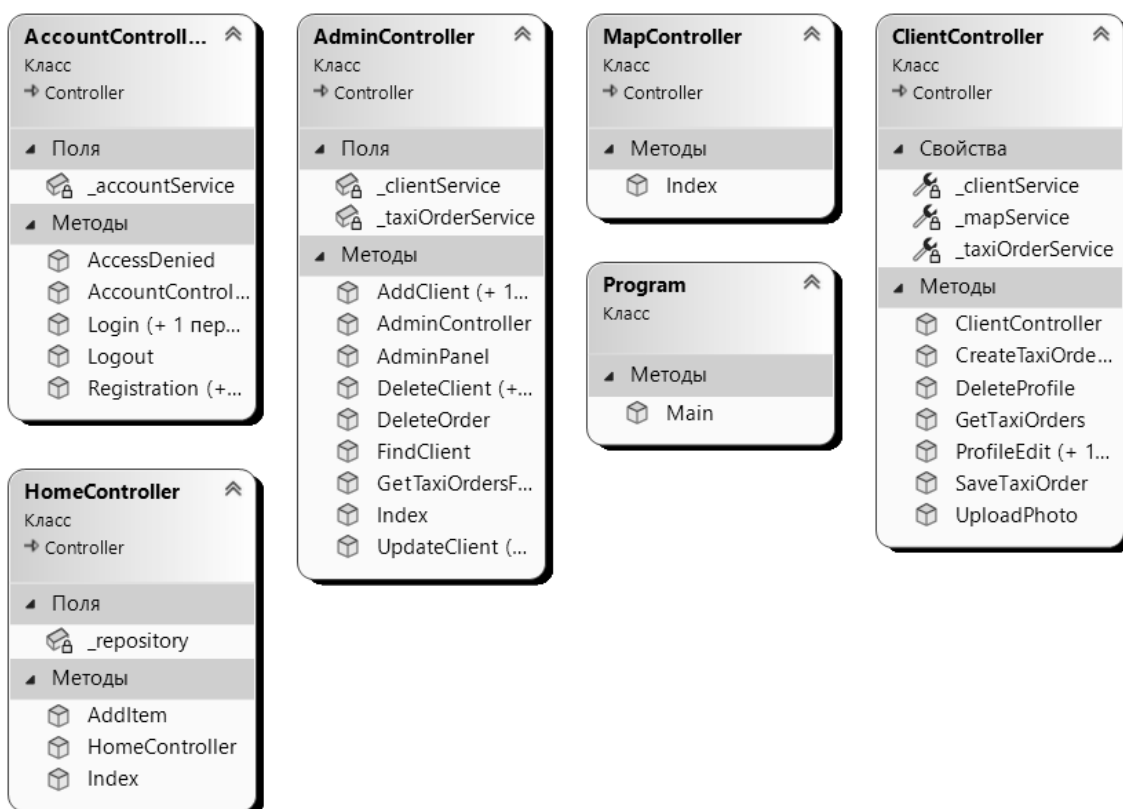
					ГУИР.153503.048.01							
					Диаграмма классов	Лит.		Масса		Масштаб		
Изм.	Лист	№ докум.	Подп.	Дата								
Разраб.		Кахновский				У					1:1	
Пров.		Владимцев										
						Лист 2		Листов 4				
						Кафедра информатики						
						группа 153503						



ГУИР.153503.048.01

Диаграмма классов

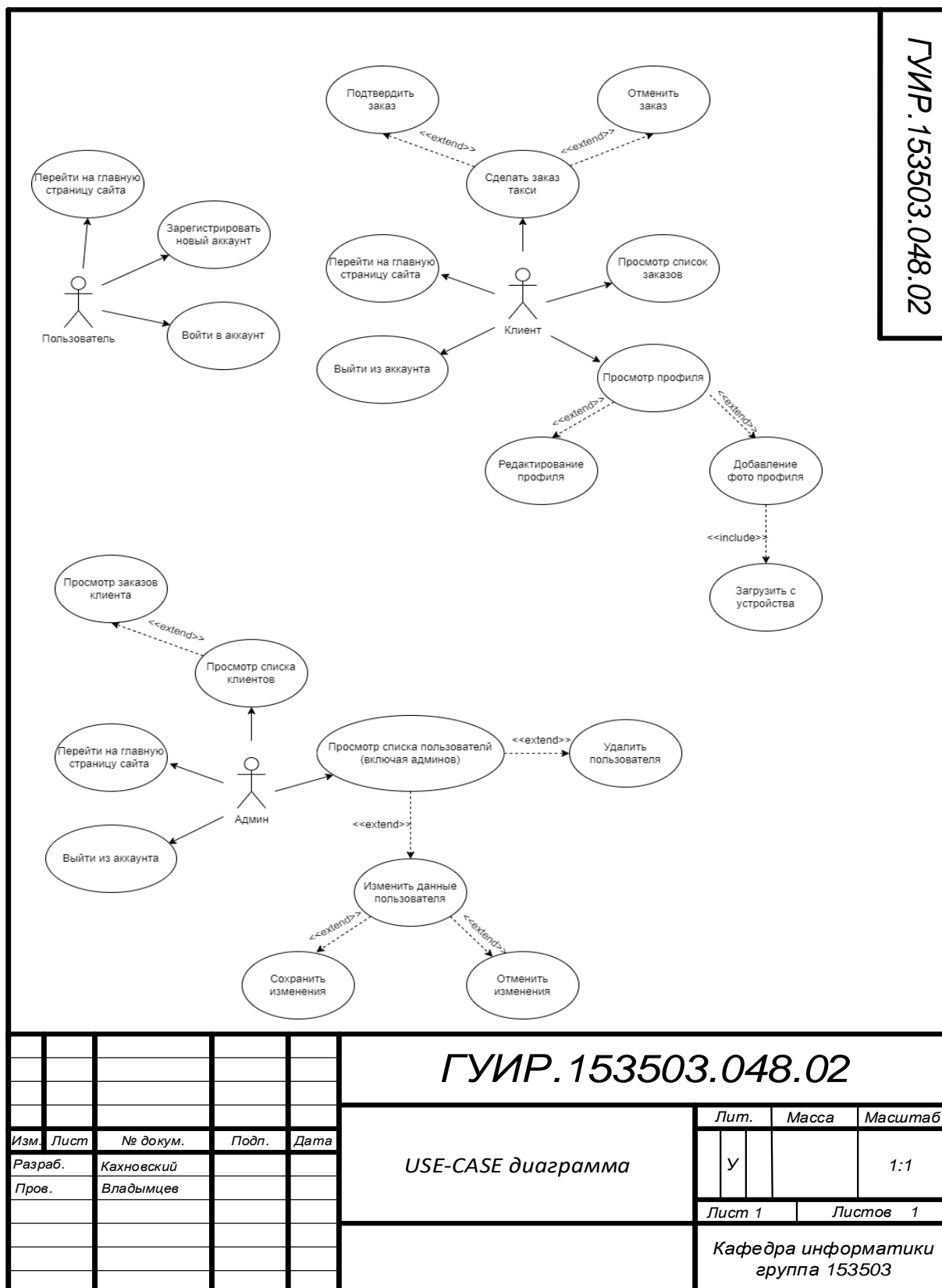
					ГУИР.153503.048.01						
					Диаграмма классов	Лит.		Масса	Масштаб		
Изм.	Лист	№ докум.	Подп.	Дата			У			1:1	
Разраб.		Кахновский									
Пров.		Владымцев									
						Лист 3		Листов 4			
						Кафедра информатики группа 153503					



					ГУИР.153503.048.01				
					Диаграмма классов	Лит.		Масса	Масштаб
Изм.	Лист	№ докум.	Подп.	Дата					
Разраб.		Кахновский				У			1:1
Пров.		Владымцев							
						Лист 4		Листов 4	
						Кафедра информатики группа 153503			

Формат А4

# **ПРИЛОЖЕНИЕ В** **(обязательное)** **USE-CASE диаграмма**



					ГУИР.153503.048.02				
					USE-CASE диаграмма	Лит.	Масса	Масштаб	
Изм.	Лист	№ докум.	Подп.	Дата					
Разраб.		Кахновский				У		1:1	
Пров.		Владынцев							
						Лист 1	Листов 1		
						Кафедра информатики группа 153503			



# ПРИЛОЖЕНИЕ Г (обязательное) Графический интерфейс

[Главная](#) [Сделать заказ](#) [История заказов](#) [Профиль](#) [Выйти](#)

### Заказ:

Начальная точка:

Конечная точка:

Описание:

Тарифы:

Метод оплаты:

Пол водителя:

Кол-во пассажиров:

### Ваши предпочтения:

☐ Домашние животные

☒ Музыка

☐ Детское кресло

☐ Поездка туда и обратно

☒ Возможность курения

☐ Тихая езда

☒ Кондиционер

☐ Помощь с багажом

☐ Платная дорога

☐ Инвалидная коляска

☒ Проф. Английский

☐ Встреча и приветствие

Рисунок 1 –Оформление заказа такси

[Главная](#) [Сделать заказ](#) [История заказов](#) [Профиль](#) [Выйти](#)

### Подтверждение маршрута:

Начальная точка: Минск

Конечная точка: Москва

Тариф заказа: Business

Цена: 1212,16 BYN

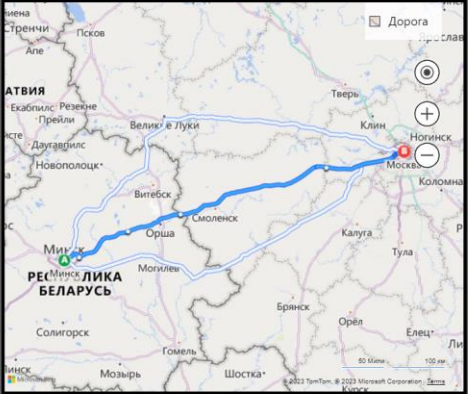
Метод оплаты: CreditCard

Предпочитаемый пол водителя: NoMetter

Дистанция маршрута: 713,038 km

Время на поездку: 07:51:47

Кол-во пассажиров: 4



### Ваши предпочтения:

☐ Домашние животные

☒ Музыка

☐ Детское кресло

☐ Поездка туда и обратно

☒ Возможность курения

☐ Тихая езда

☒ Кондиционер

☐ Помощь с багажом

☐ Платная дорога

☐ Инвалидная коляска

☒ Проф. Английский

☐ Встреча и приветствие

Рисунок 2 – Подтверждения заказа

					<i>ГУИР 153503.048.03 ПЛ</i>			
<i>Изм.</i>	<i>Лист</i>	<i>№ докум</i>	<i>Подпись</i>	<i>Дата</i>	<i>Графический интерфейс</i>	<i>Литера</i>	<i>Лист</i>	<i>Листов</i>
<i>Разраб.</i>		<i>Кахновский</i>				<i>у</i>		
<i>Пров.</i>		<i>Владимцев</i>						
<i>Рец.</i>								
<i>Н.контр.</i>		<i>Владимцев</i>				<i>Кафедра информатики</i>		
<i>Утв.</i>		<i>Владимцев</i>				<i>группа 153503</i>		

**ПРИЛОЖЕНИЕ Д**  
**(обязательное)**  
**Ведомость**

Перв. примен.	Зона	<i>Обозначение</i>	<i>Наименование</i>	<i>Дополнительные сведения</i>
ГСИР. 153503.048 ПЗ			<u>Текстовые документы</u>	
		ГСИР КП 1-40 04 01 048 ПЗ	Пояснительная записка	с.
			<u>Графические документы</u>	
		ГСИР.153503.048.01	Диаграмма классов	Формат А4
Справочный №				
		ГСИР.153503.048.02	USE-CASE диаграмма	Формат А4
		ГСИР.153503.048.03 ПЛ	Графический интерфейс	Формат А4
Подпись и дата				
Инв. № дудл.				
Взам. Инв. №				
Подпись и дата				
Инв. № подл.				

ГСИР КП 1-40 04 01 048 ПЗ				
Изм.	Лист	№ докум.	Подп.	Дата

«Агрегатор такси» Ведомость курсового проекта	<table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td style="width: 33%; text-align: center;">Лит.</td> <td style="width: 33%; text-align: center;">Лист</td> <td style="width: 33%; text-align: center;">Листов</td> </tr> <tr> <td style="text-align: center;">Т</td> <td></td> <td></td> </tr> </table> Кафедра информатики группа 153503	Лит.	Лист	Листов	Т		
Лит.	Лист	Листов					
Т							

Формат А4