

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:

**«Кроссплатформенное приложение автосалона»**

БГУИР КП 1-40 04 01 27 ПЗ

Выполнил студент группы 153503  
Щилов Павел Дмитриевич

\_\_\_\_\_  
(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...	15
4 Функциональные возможности программы.....	21
5 Архитектура разрабатываемой программы.....	29
Заключение .....	31
Список используемых источников.....	32
Приложение А (обязательное) Исходный код программы .....	32
Приложение Б (обязательное) Схема классов программы .....	40
Приложение В(обязательное) Функциональная схема программы.....	41
Приложение Г (обязательное) Графический интерфейс страницы входа .....	42
Приложение Д (обязательное) Темная тема программы .....	43

## ВВЕДЕНИЕ

В современном цифровом мире мобильные устройства стали неотъемлемой частью нашей повседневной жизни. Они предлагают удобный и мощный инструмент для доступа к различным сервисам и информации, где бы мы ни находились. Кроме того, разнообразие операционных систем и платформ, таких как iOS, Android и Windows, делает выбор подходящего приложения для пользователей вызовом для разработчиков.

В сфере автомобильных продаж автосалоны и дилерские центры сталкиваются с задачей эффективного привлечения клиентов и предоставления им удобного способа ознакомления с ассортиментом автомобилей, их характеристиками, ценами и условиями приобретения. Классический подход состоит в создании отдельных приложений для каждой операционной системы, что требует значительных затрат времени и ресурсов на разработку и поддержку.

В связи с этим, разработка кроссплатформенного приложения для автосалона является актуальной и перспективной задачей. Кроссплатформенное приложение предлагает единую базу кода, которая может быть использована для создания приложений, работающих на различных платформах. Такой подход позволяет автосалонам сэкономить время и деньги, предоставляя клиентам возможность использовать приложение независимо от их устройства и операционной системы.

Цели курсового проекта:

- 1 Приобретение теоретических и практических навыков системы ролей.
- 2 Реализация кроссплатформенного приложения с использованием локальной базы данных.

Задачи курсового проекта:

- 1 Изучение существующих технологий хранения данных.
- 2 Анализ требований к системе, установка основных критериев ее функционирования и надежности.
- 3 Разработка архитектуры системы, определение ее основных компонентов и интерфейсов.
- 4 Разработка и реализация кроссплатформенного приложения.

## 1 АНАЛИЗ ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

*Top Programming Languages 2022.* Источник является электронным ресурсом, предоставляющим информацию о самых популярных языках программирования на 2022 год. Источник является доверенным, так как IEEE (Institute of Electrical and Electronics Engineers) является одним из ведущих мировых общественных профессиональных объединений в области техники и информационных технологий. Ссылка на ресурс предоставляет возможность ознакомиться с рейтингом и обоснованиями для каждого языка программирования. Дата доступа указана, что позволяет учесть актуальность информации на момент доступа.

*SQLite.* Источник представляет собой официальный веб-сайт SQLite, компактной встраиваемой реляционной базы данных. SQLite является широко используемым инструментом для разработки приложений и имеет большую популярность в индустрии. Веб-сайт является авторитетным источником, предоставляющим документацию, руководства и другие ресурсы по использованию SQLite. Дата доступа указана, что позволяет учесть актуальность информации на момент доступа.

*Гради Буч. Объектно-ориентированный анализ и проектирование с примерами приложений.* Данный источник представляет собой книгу "Объектно-ориентированный анализ и проектирование с примерами приложений" автора Гради Буча. Книга издана в 2017 году и включает в себя 780 страниц. Гради Буч является известным специалистом в области объектно-ориентированного анализа и проектирования. Для использования в курсовой работе, данная книга может предоставить полезные сведения и примеры в контексте объектно-ориентированной разработки.

*Бертран Мейер. Почувствуй класс. Учимся программировать хорошо с объектами и контрактами.* Данный источник представляет собой книгу "Почувствуй класс. Учимся программировать хорошо с объектами и контрактами" автора Бертрана Мейера. Книга издана в 2018 году и включает в себя 540 страниц. Бертран Мейер является известным ученым в области программирования и объектно-ориентированного дизайна. В книге автор подробно описывает принципы объектно-ориентированного программирования и рассматривает важность контрактов при проектировании программных систем. Этот источник может предоставить полезные материалы и примеры для разработки курсовой работы.

*Metanit. MAUI Framework.* Источник представляет собой электронный ресурс, в котором предоставляется информация о MAUI (Multi-platform App

UI) - фреймворке для разработки кроссплатформенных мобильных приложений с использованием языка программирования C#. Сайт Metanit является надежным источником обучающих материалов по различным технологиям программирования. Дата доступа указана, что позволяет учесть актуальность информации на момент доступа.

*GitHub.* GitHub является популярной платформой разработки программного обеспечения с распределенным контролем версий Git. На GitHub размещены множество открытых проектов и репозиторий, которые предоставляют доступ к исходному коду, документации и примерам различных программных проектов. Поскольку доступ к репозиториям на GitHub открыт для сообщества разработчиков, этот ресурс может быть полезным для изучения кода, поиска решений и исследования существующих проектов, связанных с темой курсовой работы.

*Entity Framework.* Данный источник представляет собой официальную документацию и руководство по использованию Entity Framework - фреймворка для доступа к данным в приложениях на платформе .NET. Ресурс предоставляет информацию о возможностях и функциональности Entity Framework, а также руководства по его использованию. Документация от Microsoft является авторитетным источником информации, так как она создана разработчиками фреймворка и обновляется согласно последним версиям и функциональности.

*Программная платформа .NET.* Источник представляет собой официальную документацию и ресурсы по программной платформе .NET. Ресурс предоставляет информацию о возможностях и компонентах .NET, включая языки программирования, библиотеки, инструменты и среды разработки. Документация от Microsoft является авторитетным источником информации, так как она создана разработчиками платформы и обновляется согласно последним версиям и изменениям.

*What is SQLite.* Данный источник представляет собой руководство и описание SQLite - компактной встраиваемой реляционной базы данных. Ресурс предоставляет информацию о принципах работы с SQLite, ее особенностях и возможностях. Хотя источник не является официальным, Linode является известным провайдером облачных услуг, и их руководства и документация являются достоверными и полезными.

## 2 ТЕОРЕТИЧЕСКОЕ ОБОСНОВАНИЕ РАЗРАБОТКИ

При разработке программного средства были соблюдены все принципы объектно-ориентированного программирования. Особое внимание было уделено наследованию. Основные технологии, используемые для реализации проекта: C#, .NET, Entity Framework, MAUI Framework, SQLite. Рассмотрим каждую технологию в отдельности.

**C#.** C# (C-Sharp) – это объектно-ориентированный язык программирования, разработанный компанией Microsoft. Он является одним из основных языков программирования в экосистеме .NET и широко используется для разработки разнообразных типов приложений, включая настольные приложения, веб-приложения, мобильные приложения и игры.

Вот некоторые ключевые особенности и возможности C#, которые делают его полезным для нашей курсовой работы:

1 **Объектно-ориентированное программирование (ООП):** C# полностью поддерживает принципы ООП, такие как наследование, инкапсуляция, полиморфизм и абстракция. ООП позволяет организовать код в логические сущности, называемые классами, и работать с объектами, взаимодействуя через методы, свойства и события.

2 **Сильная типизация:** C# является статически типизированным языком, что означает, что типы переменных должны быть определены во время компиляции. Это помогает выявлять ошибки на ранних этапах разработки и повышает надежность и безопасность кода.

3 **Сборка мусора:** C# обеспечивает автоматическое управление памятью с помощью механизма сборки мусора. Это позволяет разработчикам не беспокоиться о явном освобождении памяти и избежать утечек памяти.

4 **Богатый набор библиотек и фреймворков:** C# включает в себя обширную библиотеку классов .NET, которая предоставляет множество функций и возможностей для разработки приложений. Кроме того, существует множество сторонних библиотек и фреймворков, которые можно использовать для ускорения разработки и расширения функциональности приложений.

5 **Поддержка асинхронного программирования:** C# предоставляет мощные механизмы для асинхронного программирования с использованием ключевых слов `async` и `await`. Это позволяет создавать отзывчивые и эффективные приложения, которые могут выполнять несколько задач параллельно.

6 **Интеграция с платформой .NET:** C# является ключевым языком для

разработки на платформе .NET. Это означает, что вы Поддержка LINQ: Language Integrated Query (LINQ) – это мощный инструмент, встроенный в C#, который позволяет выполнять запросы и манипулировать данными из различных источников, таких как коллекции объектов, базы данных и XML. LINQ упрощает и улучшает работу с данными, предоставляя выразительные операторы запросов.

7 Многопоточность: C# обеспечивает поддержку многопоточности, что позволяет создавать параллельные и асинхронные приложения для улучшения производительности и отзывчивости. С помощью параллельных задач (Parallel Tasks), многопоточных коллекций (Concurrent Collections) и других конструкций, C# упрощает работу с параллельным и асинхронным кодом.

8 Возможности для разработки мобильных приложений: C# используется вместе с платформой Xamarin для разработки кросс-платформенных мобильных приложений для Android и iOS. Xamarin позволяет использовать общий код на C# для создания приложений, которые могут работать на разных операционных системах.

9 Интеграция с базами данных: C# имеет мощную поддержку для работы с базами данных. Ваша курсовая работа использует Entity Framework, который является объектно-реляционным отображением (ORM) для работы с данными в базах данных. Entity Framework позволяет вам взаимодействовать с базами данных через моделирование данных в виде классов и предоставляет множество функций для управления данными.

10 Расширяемость: C# поддерживает создание расширений и плагинов для приложений. Вы можете использовать механизмы расширений .NET, такие как MEF (Managed Extensibility Framework), чтобы создавать модульные приложения, которые могут динамически загружать и использовать расширения.

В целом, C# обладает мощным набором возможностей, которые делают его эффективным инструментом для разработки разнообразных приложений. Его синтаксис понятен и прост, что делает его доступным для новичков, но в то же время C# предлагает и продвинутые функции для опытных разработчиков.

*SQLite.* SQLite - это легковесная и встраиваемая реляционная база данных, которая предлагает надежное хранение и управление структурированными данными. Она обеспечивает полную функциональность SQL, не требует отдельного сервера баз данных и может быть легко интегрирована в различные приложения.

Вот некоторые ключевые особенности и возможности SQLite, которые

делают ее полезной для нашей курсовой работы:

1 Легковесность: SQLite является компактной базой данных, которая имеет небольшой размер и низкие требования к ресурсам. Она может быть легко встроена в приложение без необходимости настройки и установки дополнительного сервера баз данных.

2 Поддержка SQL: SQLite полностью поддерживает язык SQL (Structured Query Language) и предоставляет мощные возможности для выполнения запросов, создания таблиц, индексов, триггеров и других базовых операций баз данных.

3 Поддержка транзакций: SQLite обеспечивает поддержку ACID-транзакций (Atomicity, Consistency, Isolation, Durability). Вы можете использовать транзакции для группировки операций базы данных в одну логическую единицу работы, обеспечивая целостность данных и возможность отката изменений.

4 Кроссплатформенность: SQLite является кроссплатформенным решением и может работать на различных операционных системах, таких как Windows, macOS, Linux, Android и iOS. Это позволяет вам создавать приложения, которые могут работать на разных платформах с использованием одной и той же базы данных.

5 Малая потребляемая память: SQLite эффективно управляет потребляемой памятью, что делает его идеальным выбором для ограниченных по ресурсам сред и мобильных устройств. Он может работать с большими объемами данных при минимальном использовании оперативной памяти.

6 Расширяемость: SQLite поддерживает расширяемость через использование модулей расширений. Вы можете добавлять пользовательские функции, агрегатные функции и виртуальные таблицы, чтобы расширить возможности SQLite в соответствии со своими потребностями.

7 Безопасность: SQLite предлагает встроенную поддержку шифрования данных. Вы можете защитить свои данные, применяя шифрование на уровне базы данных или на уровне отдельных таблиц и столбцов.

8 Портативность: Файлы баз данных SQLite представляют собой одиночный файл, что делает их легко переносимыми между различными платформами и системами. Вы можете создавать базу данных в одной операционной системе и использовать ее в другой без необходимости вносить изменения в код или схему данных.

9 Широкая поддержка: SQLite имеет широкое сообщество разработчиков и активное сообщество пользователей. Вы найдете множество документации, руководств, форумов и ресурсов, которые помогут вам в



разработке и использовании SQLite.

10 Интеграция с другими языками программирования: SQLite предоставляет интерфейсы API для множества языков программирования, включая C/C++, C#, Java, Python, Ruby и многие другие. Это позволяет вам использовать SQLite в сочетании с предпочитаемым вами языком программирования.

11 Удобный инструментарий: SQLite поставляется с набором инструментов, которые помогают управлять базой данных, выполнить запросы и анализировать данные. Некоторые из популярных инструментов включают командную строку SQLite, SQLite Studio, DB Browser for SQLite и другие.

SQLite является надежным и эффективным решением для хранения данных в различных типах приложений. Он предоставляет простой и гибкий способ работы с реляционными данными, имеет низкие требования к ресурсам и хорошую производительность. Благодаря своей простоте использования и широкой поддержке, SQLite становится популярным выбором для разработчиков при создании приложений с встроенным хранением данных.

*MAUI Framework.* MAUI (Multi-platform App UI) Framework - это кросс-платформенный фреймворк для разработки мобильных и десктопных приложений. Он является эволюцией Xamarin.Forms и предоставляет возможность создавать одну общую кодовую базу для разных платформ, таких как Android, iOS, Windows, macOS и Linux.

Вот некоторые ключевые особенности и возможности MAUI Framework, которые помогли реализовать наш проект:

1 Единая кодовая база: MAUI Framework позволяет вам писать общий код на C# для разных платформ. Вы можете разрабатывать пользовательский интерфейс, бизнес-логику и взаимодействие с данными в одном проекте, а фреймворк обеспечит его работу на разных операционных системах.

2 Кроссплатформенные контролы: MAUI Framework предоставляет набор кроссплатформенных контролов, которые позволяют создавать интерфейс приложения, согласованный с нативным стилем каждой платформы. Это позволяет создавать приложения, которые выглядят и ведут себя единообразно на разных устройствах.

3 Гибкая архитектура: MAUI Framework построен на основе паттерна MVVM (Model-View-ViewModel), что упрощает разделение логики приложения и представления данных. Вы можете легко организовать свой код в соответствии с этим паттерном и обеспечить лучшую отделенность компонентов.

4 Поддержка нативной функциональности: MAUI Framework предоставляет возможность использовать нативные API и возможности каждой платформы. Вы можете взаимодействовать с камерой, геолокацией, уведомлениями, файловой системой и другими возможностями устройств с помощью унифицированного API.

5 Хорошая производительность: MAUI Framework оптимизирован для достижения высокой производительности приложений. Он использует современные техники рендеринга, кэширования и оптимизации, чтобы обеспечить отзывчивость и плавность работы интерфейса.

6 Поддержка расширений и плагинов: MAUI Framework предоставляет механизмы расширений и плагинов, которые позволяют добавлять дополнительную функциональность в приложения. Вы можете использовать существующие расширения или создавать собственные, чтобы расширить возможности фреймворка.

7 Разработка с использованием XAML: MAUI Framework использует язык разметки XAML (eXtensible Application Markup Language) для определения пользовательского интерфейса. XAML позволяет создавать декларативные описания интерфейса, что делает разработку более удобной и позволяет легко разделять внешний вид приложения от логики.

8 Интеграция с экосистемой .NET: MAUI Framework построен на базе платформы .NET и полностью интегрирован с экосистемой инструментов и библиотек .NET. Вы можете использовать широкий спектр инструментов разработки, библиотек и сервисов, которые предоставляются .NET, для ускорения разработки и повышения производительности.

9 Совместимость с Xamarin: MAUI Framework обеспечивает обратную совместимость с Xamarin.Forms, что позволяет переносить существующие проекты на новую платформу. Вы можете использовать существующий код, ресурсы и знания, чтобы упростить переход на MAUI Framework.

10 Активное сообщество: MAUI Framework имеет активное сообщество разработчиков и пользователей. Вы найдете множество руководств, документации, обучающих материалов, форумов и ресурсов, которые помогут вам в разработке приложений с использованием MAUI Framework.

MAUI Framework предоставляет мощные инструменты и возможности для разработки кроссплатформенных приложений с использованием единой кодовой базы. Он позволяет создавать современные и эффективные приложения, которые могут работать на разных операционных системах без необходимости разработки и поддержки отдельных версий для каждой платформы.

*Entity Framework*. Entity Framework (EF) – это объектно-реляционный отображающий (ORM) фреймворк, разработанный компанией Microsoft. Он предоставляет инструменты и функциональность для работы с данными в базе данных с использованием объектно-ориентированной модели.

Вот некоторые ключевые особенности и возможности Entity Framework, из-за которых он был очень полезен для нашей курсовой работы:

1 Отображение объектов на таблицы: Entity Framework позволяет разработчикам работать с данными в базе данных, используя объектно-ориентированную модель. Он автоматически отображает классы и свойства объектов на таблицы и столбцы в базе данных, что упрощает взаимодействие с данными.

2 Язык запросов LINQ: Entity Framework поддерживает использование языка запросов LINQ (Language Integrated Query) для выполнения запросов к базе данных. LINQ предоставляет выразительный синтаксис для формулирования запросов и манипуляции данными, что делает код более понятным и легким для сопровождения.

3 Автоматическая генерация SQL: Entity Framework обеспечивает автоматическую генерацию SQL-запросов на основе операций, выполняемых с объектами. Он позволяет разработчикам работать с данными на уровне объектов, а затем автоматически генерировать соответствующий SQL для выполнения операций в базе данных.

4 Миграции баз данных: Entity Framework предоставляет механизм миграций баз данных, который упрощает обновление схемы базы данных в соответствии с изменениями в объектной модели. Он позволяет автоматически создавать и применять миграции, обеспечивая целостность данных и управление версиями базы данных.

5 Кэширование данных: Entity Framework поддерживает кэширование данных, что позволяет повысить производительность приложения. Он предоставляет механизмы кэширования запросов и результатов запросов, чтобы уменьшить количество обращений к базе данных и улучшить отклик приложения.

6 Поддержка различных провайдеров баз данных: Entity Framework поддерживает различные провайдеры баз данных, включая SQL Server, MySQL, PostgreSQL, Oracle и другие. Это означает, что вы можете использовать Entity Framework для работы с разными типами баз данных, сохраняя при этом единый подход к работе с данными.

7 Удобная интеграция с .NET: Entity Framework интегрируется с платформой .NET, что обеспечивает удобство в использовании и повышает

производительность разработки. Он полностью интегрирован с языком программирования C# и другими компонентами платформы .NET, такими как LINQ, ASP.NET и Windows Presentation Foundation (WPF).

8 Поддержка разных подходов к моделированию данных: Entity Framework предлагает поддержку разных подходов к моделированию данных, включая базовый (Code First), модель базы данных (Database First) и модель разделения (Model Splitting). Это позволяет разработчикам выбрать наиболее подходящий подход для своих потребностей разработки.

9 Инструменты разработки: Entity Framework предоставляет инструменты разработки, которые упрощают процесс работы с данными. Например, он предлагает возможность визуального проектирования модели данных с помощью Entity Designer, а также инструменты для отладки и профилирования запросов.

10 Поддержка транзакций: Entity Framework поддерживает транзакции, что позволяет выполнять операции с данными в рамках одной транзакции. Это обеспечивает целостность данных и возможность отката изменений в случае ошибки или исключительной ситуации.

11 Расширяемость: Entity Framework предоставляет возможность расширения функциональности через плагины и расширения. Вы можете создавать собственные провайдеры баз данных, расширять возможности запросов и добавлять пользовательские функции, чтобы адаптировать Entity Framework под свои уникальные требования.

Entity Framework предоставляет мощные инструменты и функциональность для работы с данными в приложениях на платформе .NET. Он упрощает разработку и обслуживание баз данных, позволяет эффективно работать с объектно-ориентированной моделью данных и предоставляет удобные средства для выполнения запросов, управления транзакциями и улучшения производительности приложений.

Каждая из этих технологий вносит значительный вклад в разработку программного средства и обладает своими уникальными особенностями и преимуществами.

C# является мощным и элегантным языком программирования, который обладает широкими возможностями для создания приложений на платформе .NET. Он обеспечивает высокую производительность, безопасность типов, поддержку объектно-ориентированного программирования и богатый набор инструментов разработки.

.NET 7 представляет собой платформу для разработки приложений, которая включает в себя среду выполнения, набор библиотек классов и

инструменты разработки. Он обеспечивает кроссплатформенность, высокую производительность, поддержку различных языков программирования и интеграцию с другими технологиями, такими как Entity Framework и MAUI Framework.

SQLite является легковесной встроенной базой данных, которая обеспечивает эффективное хранение и управление данными. Он обладает простым в использовании SQL-интерфейсом, надежностью, кроссплатформенностью и поддержкой транзакций, что делает его идеальным выбором для встроенных и мобильных приложений.

MAUI Framework является кроссплатформенным фреймворком для разработки мобильных и десктопных приложений, который позволяет создавать современные и привлекательные пользовательские интерфейсы. Он обладает мощными инструментами разработки, поддержкой XAML и интеграцией с другими технологиями .NET, что позволяет разработчикам создавать высококачественные приложения с использованием единой кодовой базы.

Entity Framework является ORM-фреймворком, который облегчает взаимодействие с данными в базе данных, предоставляя удобные инструменты для работы с объектно-ориентированной моделью. Он поддерживает язык запросов LINQ, автоматическую генерацию SQL, миграции баз данных и интеграцию с различными провайдерами баз данных. В целом, использование этих технологий в проекте обеспечивает разработчикам мощный и современный инструментарий для создания высококачественных программных решений.

C# и .NET 7 предоставляют надежную и эффективную платформу для разработки приложений, обеспечивая высокую производительность, гибкость и масштабируемость.

Entity Framework позволяет разработчикам эффективно работать с данными, предоставляя инструменты для отображения объектов на таблицы базы данных, выполнения запросов с использованием LINQ и автоматической генерации SQL. Он облегчает взаимодействие с базой данных и упрощает процесс работы с данными.

Сочетание всех этих технологий в проекте позволило создать мощные и эффективные программные решения, обеспечивая удобство разработки, гибкость, производительность и надежность. Комбинация языка программирования C#, платформы .NET, базы данных SQLite, фреймворка MAUI и ORM-фреймворка Entity Framework создает прочную основу для разработки современных и инновационных приложений.

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

MVVM. Паттерн MVVM (Model-View-ViewModel) является архитектурным шаблоном проектирования, который широко применяется в разработке пользовательских интерфейсов программных приложений. MVVM разделяет приложение на три основных компонента: модель (Model), представление (View) и модель представления (ViewModel), что позволяет достичь более легкой поддержки и тестирования кода, а также улучшить его читаемость и многоразовое использование.

Вот подробное описание каждого компонента MVVM:

1 Модель (Model). Модель представляет бизнес-логику и данные приложения. Она может включать классы, структуры, интерфейсы и другие компоненты, отвечающие за обработку данных и взаимодействие с внешними источниками данных, такими как базы данных, веб-сервисы и так далее. Модель независима от представления и модели представления, что обеспечивает ее многоразовое использование в других контекстах и упрощает ее тестирование.

2 Представление (View). Представление отвечает за отображение данных модели и взаимодействие с пользователем. Оно представляет собой пользовательский интерфейс приложения, включая элементы управления, макеты, стили и другие компоненты, отображаемые на экране. Представление обычно биндится к свойствам и командам модели представления для отображения и обработки данных.

3 Модель представления (ViewModel). Модель представления служит посредником между моделью и представлением. Она предоставляет данные и команды, необходимые для отображения и взаимодействия с моделью. Модель представления обычно реализует интерфейсы и свойства, которые привязываются к представлению и обеспечивают поток данных между представлением и моделью. Модель представления также может содержать логику обработки событий, валидацию данных, управление навигацией и другие операции, связанные с представлением и моделью.

Основные принципы и преимущества паттерна MVVM:

1 Разделение ответственностей: MVVM разделяет логику приложения на три четко определенных компонента, что упрощает понимание и сопровождение кода.

2 Улучшенная тестируемость: MVVM позволяет легко тестировать каждый компонент независимо друг от друга. Модель может быть

протестирована без необходимости взаимодействия с представлением или моделью представления. Также, модель представления может быть протестирована без необходимости имитации пользовательского интерфейса.

3 Высокая многократное использование: Благодаря разделению логики и данных на различные компоненты, каждый из них может быть повторно использован в других частях приложения. Модель представления, например, может быть использована с различными представлениями для отображения тех же данных.

4 Улучшенная читаемость и поддержка кода: MVVM обеспечивает структурирование кода, что делает его более понятным и удобным для поддержки и расширения. Разделение логики на отдельные компоненты также способствует легкому обновлению и модификации кода без влияния на другие компоненты.

5 Улучшенная разработка интерфейса: MVVM обеспечивает четкое разделение представления и логики, что упрощает разработку пользовательского интерфейса. Дизайнеры могут работать над представлением независимо от разработчиков, а разработчики могут фокусироваться на реализации логики в модели представления без изменения визуальной составляющей.

6 Поддержка связывания данных: MVVM обеспечивает мощный механизм связывания данных между моделью представления и представлением. Это позволяет автоматически обновлять пользовательский интерфейс при изменении данных в модели представления, а также обрабатывать пользовательский ввод и команды.

Однако, при использовании паттерна MVVM необходимо учитывать некоторые аспекты, такие как возможное увеличение сложности кода и необходимость в дополнительной обработке событий и команд в модели представления. Также требуется дополнительное внимание при связывании данных и обработке ошибок, чтобы избежать утечек памяти и проблем с производительностью.

В целом, паттерн MVVM является мощным инструментом для разработки пользовательских интерфейсов, обеспечивающим хорошую организацию кода, легкую тестируемость и высокую производительность. Он позволяет разделить логику приложения на отдельные компоненты, что способствует улучшению поддержки, многократного использования и расширяемости кода. MVVM также облегчает совместную работу дизайнеров и разработчиков, поскольку представление и модель представления могут быть разработаны и изменены независимо друг от друга.

Использование паттерна MVVM в проекте, основанном на C#, .NET 7, MAUI Framework, Entity Framework и SQLite, позволяет создать современное приложение с эффективной архитектурой и отзывчивым пользовательским интерфейсом. Модель представления (ViewModel) взаимодействует с моделью (Model), обеспечивая доступ к данным и бизнес-логике, а также предоставляет связывание данных для представления (View). Это позволяет легко отображать данные из модели в пользовательском интерфейсе и реагировать на взаимодействие пользователя.

В целом, паттерн MVVM является мощным инструментом для разработки современных и гибких пользовательских интерфейсов. Его использование в сочетании с выбранными технологиями позволяет создавать высококачественные и удобные приложения, обладающие хорошей структурированностью, легкостью сопровождения и высокой производительностью.

**Фабрика.** Паттерн Фабрика (Factory) относится к классу порождающих паттернов проектирования и предоставляет механизм для создания объектов без явного указания конкретных классов, используя общий интерфейс или базовый класс. Фабрика инкапсулирует процесс создания объектов, обеспечивая гибкость и упрощение кода.

Основная идея паттерна Фабрика заключается в том, чтобы вынести создание объектов из клиентского кода в отдельный компонент, называемый фабрикой. Фабрика предоставляет методы для создания объектов определенного типа, скрывая детали конкретной реализации. Таким образом, клиентский код взаимодействует только с интерфейсом фабрики, а не с конкретными классами объектов.

В паттерне Фабрика выделяются следующие основные роли:

1 **Продукт (Product).** Это абстрактный класс или интерфейс, представляющий общий интерфейс создаваемых объектов. Конкретные классы продуктов реализуют этот интерфейс и предоставляют специфическую реализацию операций.

2 **Фабрика (Factory).** Это абстрактный класс или интерфейс, определяющий методы создания объектов продуктов. Конкретные фабрики наследуют абстрактную фабрику и реализуют методы создания конкретных продуктов. Фабрика может иметь несколько методов создания, каждый из которых создает объект определенного типа продукта.

3 **Клиент (Client).** Клиентский код взаимодействует с фабрикой через ее интерфейс и не зависит от конкретных классов продуктов. Клиент использует методы фабрики для создания объектов продуктов и выполняет операции над



ними.

Преимущества использования паттерна Фабрика:

1 Упрощение кода: Паттерн Фабрика позволяет вынести сложность создания объектов из клиентского кода, что упрощает его понимание и поддержку. Клиентский код работает с абстракцией фабрики, не заботясь о деталях создания конкретных объектов.

2 Гибкость и расширяемость: Фабрика позволяет легко добавлять новые типы продуктов, расширяя абстрактную фабрику и создавая конкретные реализации фабрики для новых продуктов. Это позволяет легко внедрять изменения в систему без необходимости изменения клиентского кода.

3 Соккрытие деталей реализации: Клиентский код не зависит от конкретных классов продуктов, так как работает только с абстракцией фабрики и интерфейсом продукта. Это позволяет скрыть детали реализации и управлять созданием объектов из одного места.

4 Повышение связности: Паттерн Фабрика способствует повышению связности в системе, так как группирует связанные классы и операции внутри одной фабрики. Это помогает поддерживать целостность и согласованность объектов.

Несмотря на свои преимущества, паттерн Фабрика также имеет некоторые ограничения и недостатки:

1 Увеличение сложности системы: Введение фабрик может увеличить сложность системы из-за введения дополнительных классов и абстракций. Это может быть нежелательным в простых приложениях с небольшим количеством продуктов.

2 Ограниченность вариантов создания объектов: Фабрика определяет только конкретные способы создания объектов, которые были заранее заданы в абстрактной фабрике. Если требуется сложная логика создания объектов, паттерн Фабрика может оказаться недостаточным.

3 Зависимость от фабрик: Клиентский код становится зависимым от фабрик, что может усложнить тестирование и внедрение зависимостей. Это может потребовать использования инверсии управления (IoC) или Dependency Injection (DI) для управления зависимостями и облегчения тестирования.

В целом, паттерн Фабрика является полезным инструментом для управления созданием объектов в приложении. Он способствует упрощению кода, повышает гибкость и расширяемость системы, а также улучшает ее связность и поддержку. Однако, необходимо внимательно оценить его применимость и учитывать возможные недостатки и ограничения в конкретном контексте проекта.

Паттерны Репозиторий (Repository) и Unit Of Work. Паттерн Репозиторий (Repository) и паттерн Unit of Work относятся к классу паттернов проектирования, используемых в разработке программного обеспечения. Эти паттерны связаны с управлением доступом к данным и обеспечивают эффективную работу с источниками данных.

Паттерн Репозиторий (Repository). Паттерн Репозиторий представляет собой прослойку между бизнес-логикой приложения и источниками данных, такими как база данных или веб-сервисы. Репозиторий предоставляет унифицированный интерфейс для работы с данными, скрывая детали конкретной реализации доступа к данным. Он предоставляет операции для создания, чтения, обновления и удаления объектов данных, а также позволяет выполнять запросы и фильтрацию данных. Репозиторий абстрагирует клиентский код от деталей работы с базой данных и упрощает тестирование бизнес-логики без необходимости работать с реальными источниками данных.

Паттерн Unit of Work (Единица работы). Паттерн Unit of Work отвечает за управление жизненным циклом транзакций и обеспечивает целостность изменений, связанных с объектами данных. Единица работы группирует операции над несколькими объектами данных в единый контекст, который может быть подтвержден или откатан как одна транзакция. Он отслеживает изменения в объектах данных, обеспечивает сохранение изменений в источниках данных, а также позволяет отменить несохраненные изменения. Единица работы упрощает управление транзакциями и обеспечивает согласованность изменений в базе данных, предоставляя централизованный механизм для работы с данными.

Преимущества использования паттернов Репозиторий и Unit of Work:

1 Разделение ответственности: Паттерны позволяют разделить ответственность между бизнес-логикой приложения и доступом к данным. Репозиторий абстрагирует клиентский код от деталей работы с базой данных, а Unit of Work обеспечивает централизованное управление изменениями.

2 Упрощение тестирования: Паттерны позволяют легко тестировать бизнес-логику, так как клиентский код не зависит от конкретной реализации доступа к данным. Вместо этого, для тестирования можно использовать заглушки или моки для репозитория и единицы работы, что упрощает создание изолированных тестовых сценариев.

3 Гибкость и расширяемость: Паттерны Репозиторий и Unit of Work позволяют легко заменять или добавлять новые источники данных без внесения изменений в клиентский код. Это упрощает поддержку различных баз данных или других источников данных и обеспечивает гибкость в выборе

технологий.

4 **Согласованность данных:** Единица работы обеспечивает согласованность изменений в базе данных. Изменения, связанные с различными объектами данных, могут быть сохранены или откатаны как одна транзакция, что помогает предотвратить неоднородность данных.

Однако, следует учитывать некоторые аспекты и рекомендации при использовании этих паттернов:

1 **Разумное использование.** Паттерны Репозиторий и Unit of Work целесообразно использовать в приложениях с достаточным уровнем сложности и с большим объемом работы с данными. В простых проектах они могут быть избыточными и усложнить код.

2 **Проектирование интерфейсов.** Репозиторий должен предоставлять только те методы, которые необходимы для выполнения операций с данными. Интерфейсы репозитория должны быть четко определены и сосредоточены на конкретных сущностях.

3 **Управление состоянием.** Единица работы должна управлять состоянием объектов данных и отслеживать их изменения. Для этого может потребоваться использование шаблона "Наблюдатель" или других механизмов для определения изменений.

В целом, паттерны Репозиторий и Unit of Work предоставляют эффективный подход к управлению доступом к данным и обеспечивают гибкость, расширяемость и согласованность в работе с источниками данных. Их использование позволяет разделить ответственность и упростить тестирование и сопровождение приложения.

## 4 ФУНКЦИОНАЛЬНЫЕ ВОЗМОЖНОСТИ ПРОГРАММЫ

Возможности программного средства продемонстрированы с помощью веб-приложения. Техническое изложение предоставляемого функционала можно увидеть на use-case диаграмме (см. рисунок 1):

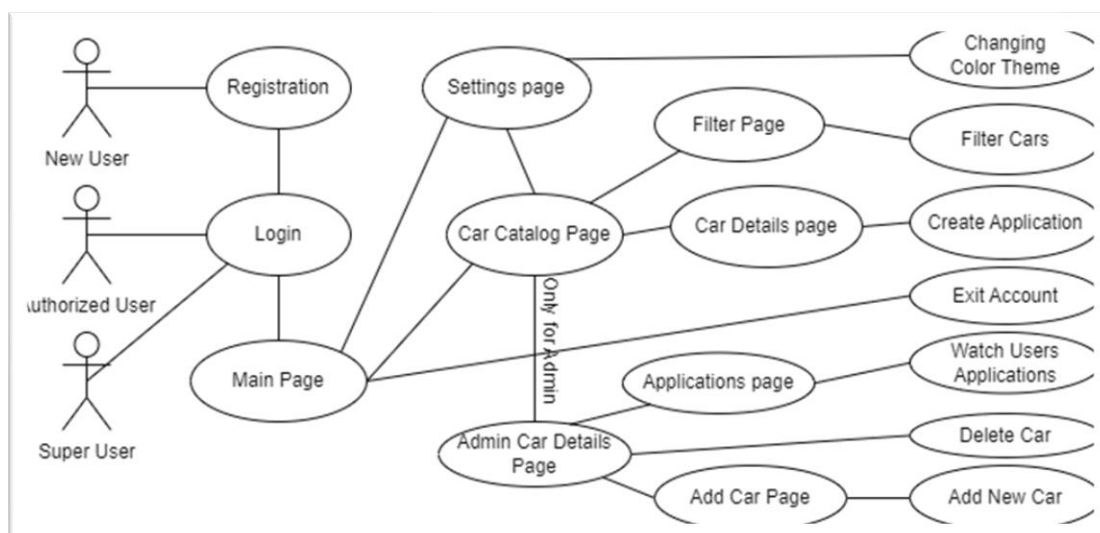


Рисунок 1 – Use-case диаграмма приложения

Чтобы получить доступ к основным функциям приложения, пользователю необходимо пройти регистрацию (см. рисунок 3).

MotorLounge

Sign in to continue

Email

Password

Sign in

New user? Sign Up

Рисунок 2 – Страница входа

Если он уже зарегистрирован, пройти авторизацию (см. рисунок 2), используя данные, которые он указывал при регистрации.

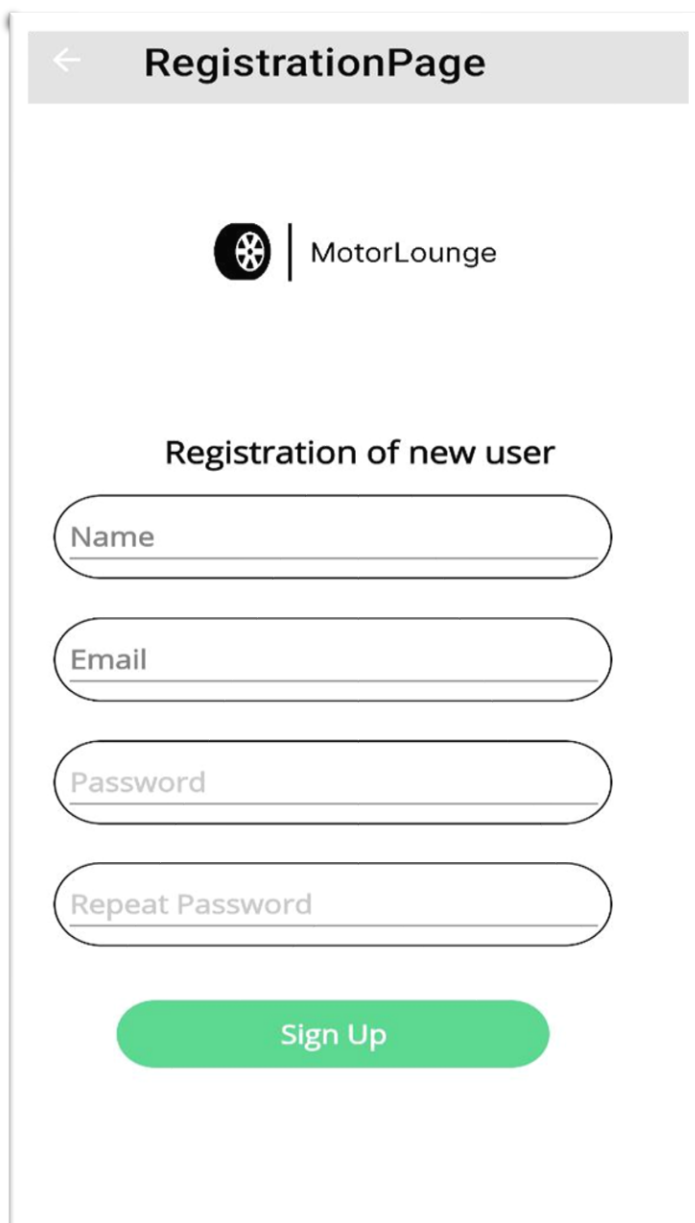
The image shows a mobile application registration screen. At the top, there is a grey header bar with a back arrow on the left and the title "RegistrationPage" in the center. Below the header, there is a logo consisting of a black circle with a white wheel-like icon inside, followed by a vertical line and the text "MotorLounge". Underneath the logo, the text "Registration of new user" is centered. Below this text, there are four rounded rectangular input fields stacked vertically. The first field is labeled "Name", the second "Email", the third "Password", and the fourth "Repeat Password". At the bottom of the form, there is a green rounded rectangular button with the text "Sign Up" in white.

Рисунок 3 – Страница регистрации

После успешной регистрации/авторизации, пользователь получает доступ к перемещению по всем страницам приложения: главная страница приложения (см. рисунок 4), каталог автомобилей (см. рисунок 5), страница настроек (см. рисунок 6), страница фильтра автомобилей (см. рисунок 7), страница подробной информации об автомобиле (см. рисунок 8). Каждая из страниц приложения имеет уникальный внешний вид, отличающийся минимализмом дизайна.

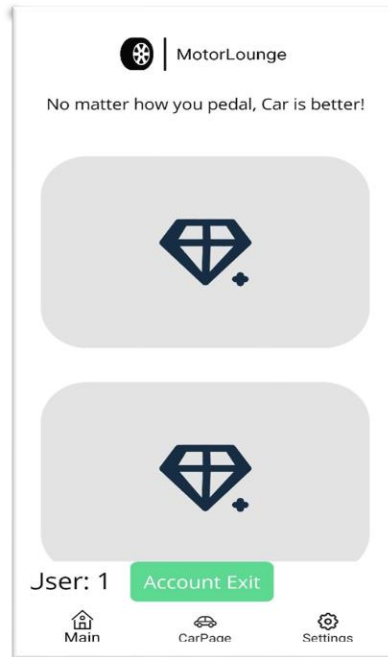


Рисунок 4 – Главная страница приложения

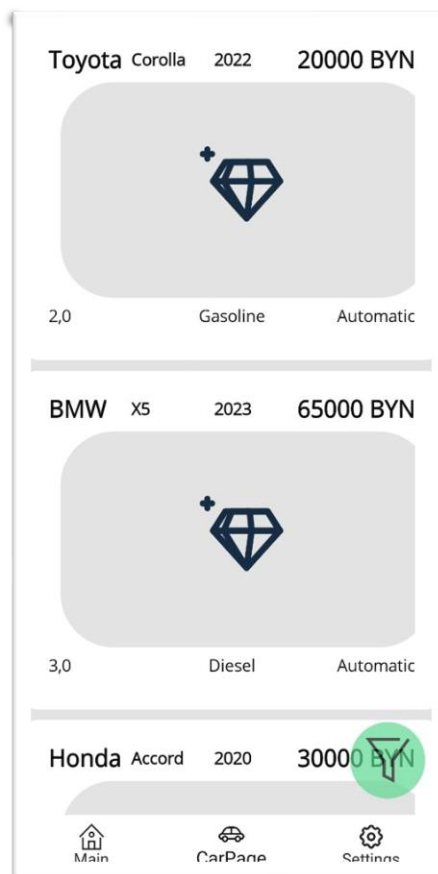


Рисунок 5 – Каталог автомобилей



Рисунок 6 – Страница настроек приложения

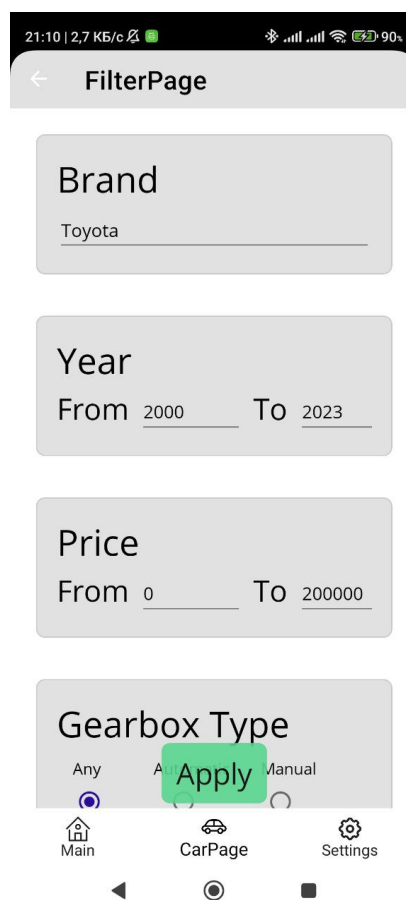


Рисунок 7 – Страница фильтра автомобилей

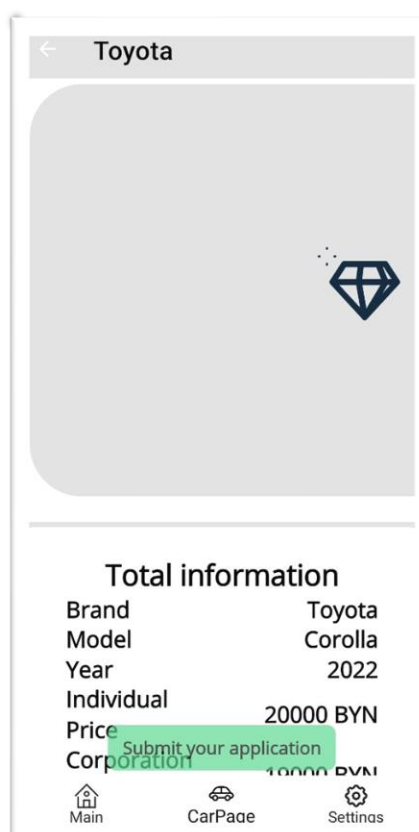


Рисунок 8 – Страница подробной информации

Пользователь может взаимодействовать с главной страницей приложения следующими образами:

- 1 Использовать кнопку выхода из аккаунта для перехода на экран входа в приложение.
- 2 Просматривать содержимое главной страницы.

Пользователь может взаимодействовать со страницей каталога автомобилей приложения следующими образами:

- 1 Просматривать весь каталог автомобилей.
- 2 По кнопке в нижнем правом углу перейти на страницу фильтра автомобилей и отфильтровать весь список автомобилей по заданным параметрам, после чего вернуться на страницу каталога и просматривать отфильтрованные авто.

3 Нажатием на объект автомобиля в списке пользователь перейдет на страницу подробной информации.

Пользователь может взаимодействовать со страницей подробной информации об автомобиле приложения следующими образами:

- 1 Просматривать подробную информацию об автомобиле.
- 2 Нажатием кнопки внизу страницы создать заявку, которую получит



суперпользователь.

На странице настроек пользователь может переключить тему приложения со светлой на темную и обратно. На рисунке 9 можно увидеть интерфейс приложения со включенной темной темой приложения.

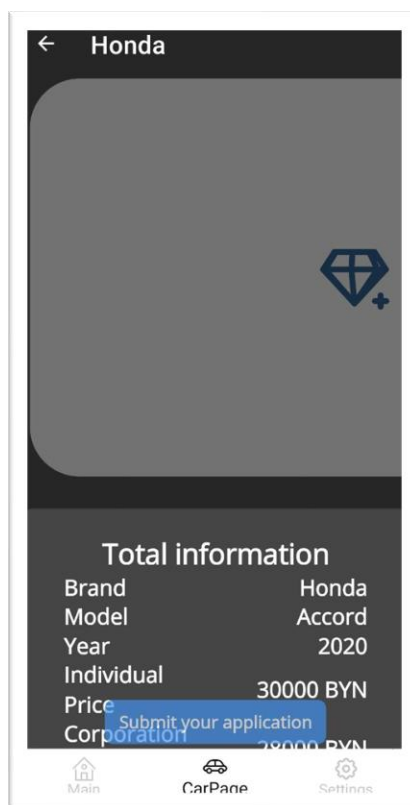


Рисунок 9 – Тёмная тема приложения

Как можно заметить, темная тема приложения заменяет все цвета на более темные, что делает приложение более Суперпользователь, или же админ – это пользователь с расширенными возможностями. Рассмотрим его уникальные права. На рисунке 10 можно увидеть, что страница подробной информации для суперпользователя изменена. Для админа отсутствует кнопка для формирования заявки, однако ниже фотографий автомобиля приведено специальное меню, состоящее из 3 кнопок: добавление новой машины, удаление автомобиля, который сейчас выбран, а также кнопка, приводящая к странице заявок, которые делали обычные пользователи. На рисунке 11 можно увидеть страницу добавления новой машины в каталог, а на рисунке 12 – страницу с заявками пользователей.

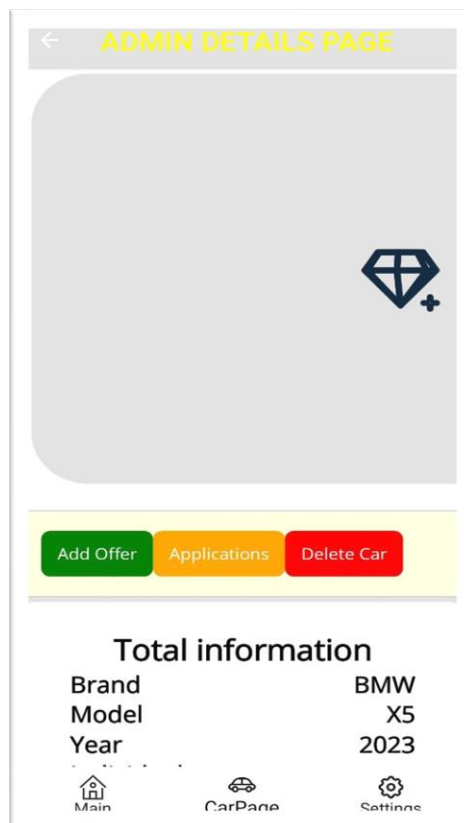


Рисунок 10 – Страница подробной информации глазами админа

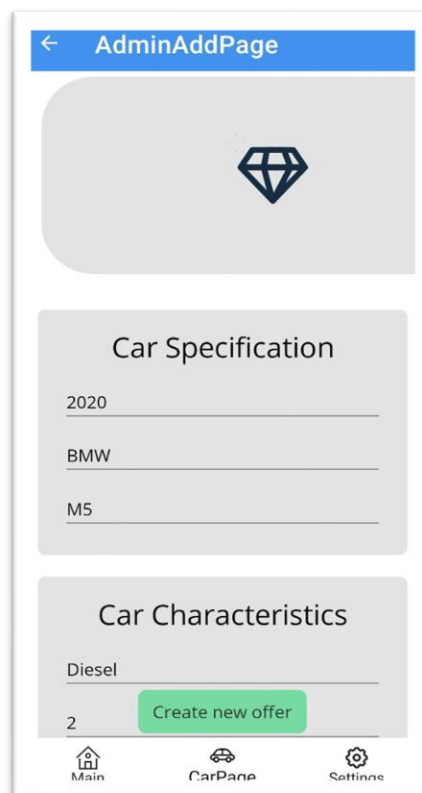


Рисунок 11 – Страница добавления в каталог нового автомобиля

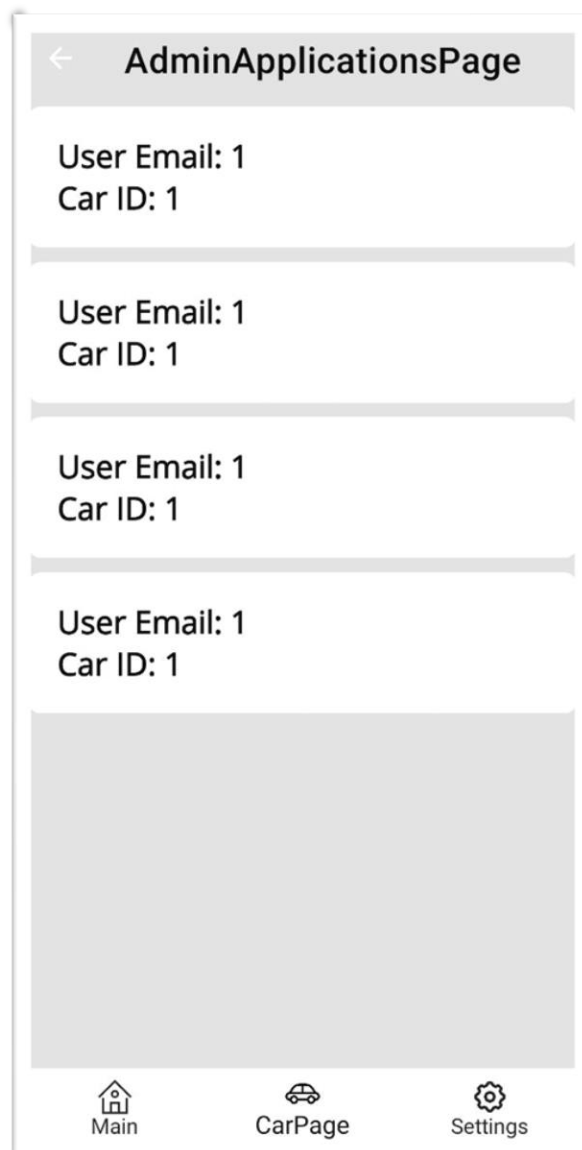


Рисунок 12 – Страница пользовательских заявок

Несложно заметить, что каждая из страниц имеет схожий минималистичный внешний вид, который соответствует определенному стилю. В данном стиле легко рассмотреть главные цвета: белый, серый, светло-синий, а также светло-зеленый. Данные цвета превосходно сочетаются со строгим стилем приложения.

## 5 АРХИТЕКТУРА РАЗРАБАТЫВАЕМОЙ ПРОГРАММЫ

Как уже отмечалось ранее, в основе приложения лежит принцип наследования, а также паттерны проектирования MVVM, «Фабрика», «Единица работы», «Репозиторий».

Наследование также играет важную роль в архитектуре приложения. Детальнее с построением зависимостей между классами можно ознакомиться на UML-схеме (см. рисунок 13).

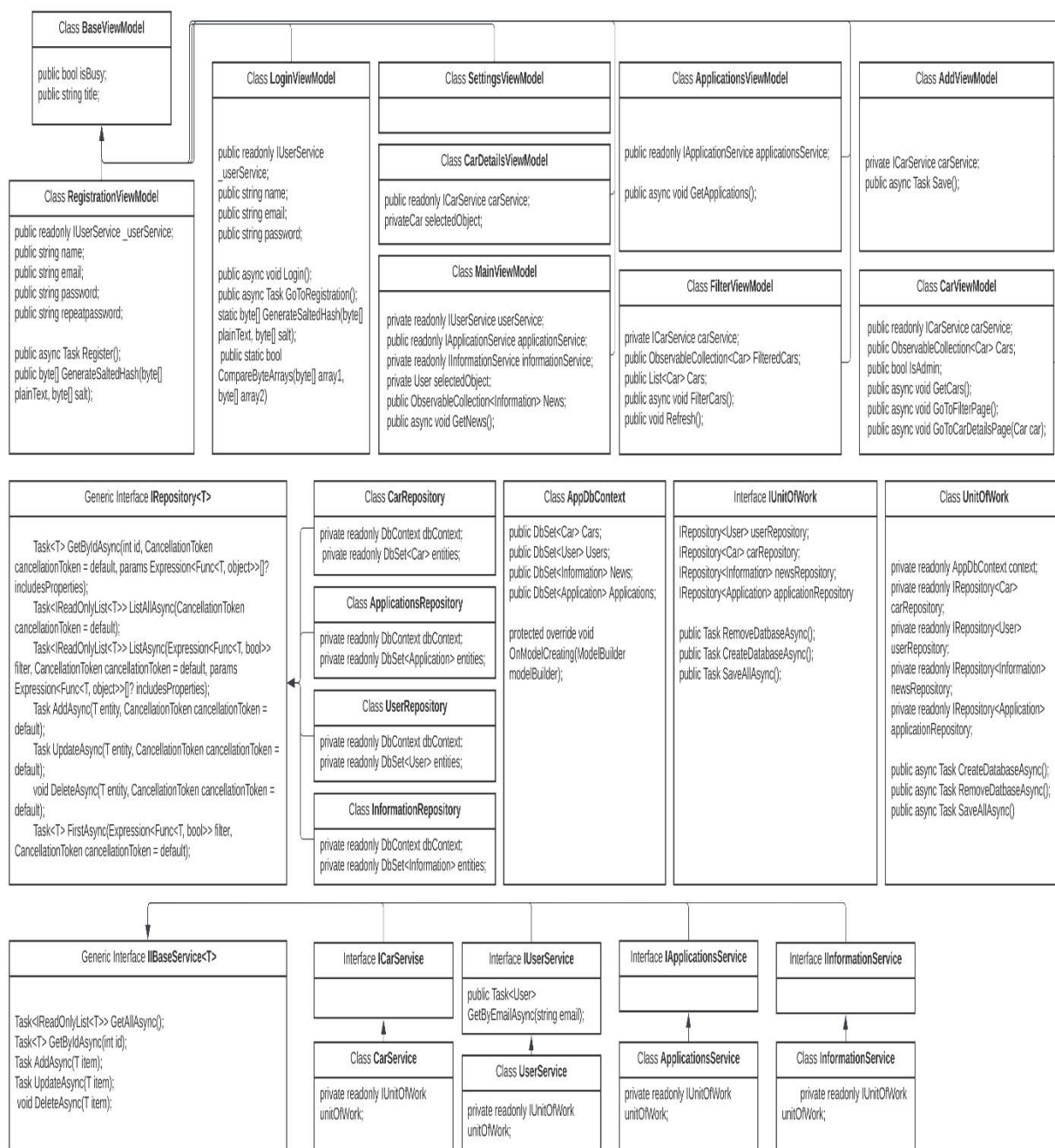


Рисунок 13 – Диаграмма классов приложения

Парадигма объектно-ориентированного программирования повсеместно используется при создании современного программного обеспечения. Модель объектов, заложенная в данную парадигму, способна достаточно точно описывать свойства и возможности сущностей реального мира. Разумеется, эти объекты не существуют обособленно друг от друга, они взаимодействуют друг с другом для достижения какой-то глобальной цели разрабатываемой системы.

Стандартная библиотека некоторого языка программирования – замечательный сборник полезных утилит. Однако разнообразие решаемых программистами задач так велико, что одной только стандартной библиотекой ограничиться не получится. Программисту часто приходится самому создавать необходимый ему набор функциональности. Это можно сделать, создав пакет функций или набор классов.

Создание собственных классов при разработке программы добавляет в проект новый уровень абстракции, который позволяет определить некоторый функционал системы и работать в дальнейшем только с ним. Графическая интерпретация задуманного всегда упрощает реализацию.

Все сущности реального мира, с которыми собирается работать программист, должны быть представлены объектами классов в программе. При этом у каждого класса должно быть только одно назначение и уникально осмысленное имя, которое будет связано с этой целью.

Все эти архитектурные принципы, паттерны и технологии в совокупности обеспечат гибкую, расширяемую и хорошо организованную структуру разрабатываемой программы. Они позволят разделить ответственность между различными компонентами, упростить тестирование и сопровождение кода, а также обеспечить надежность и согласованность данных.

## ЗАКЛЮЧЕНИЕ

В ходе выполнения курсовой работы были изучены и применены различные технологии, паттерны и принципы архитектуры программного средства. Вся архитектура была спроектирована с учетом требований разрабатываемого приложения и целей проекта.

Была выбрана архитектура на основе паттерна MVVM, который обеспечивает разделение бизнес-логики, пользовательского интерфейса и данных. Это позволило упростить разработку и поддержку кода, а также обеспечить гибкость и многократное использование компонентов.

Для управления доступом к данным были использованы паттерны Репозиторий и Unit of Work. Репозиторий абстрагирует доступ к данным и предоставляет интерфейс для работы с различными источниками данных, включая базу данных SQLite. Unit of Work обеспечивает управление жизненным циклом транзакций и согласованность изменений данных.

В разработке приложения были использованы технологии C#, .NET 7, MAUI Framework, Entity Framework и SQLite. C# является мощным языком программирования, а .NET 7 и MAUI Framework предоставляют современные инструменты для разработки кроссплатформенных приложений.

В целом, разработка кроссплатформенного приложения автосалона является актуальной темой, так как хороших приложений такой направленности мало, и пользователи часто встречаются с неприятными ситуациями, связанными с графическим интерфейсом.

Итогом работы можно считать выполненными поставленные ранее цели и задачи:

Цели курсового проекта:

- 1 Приобретение теоретических и практических навыков системы ролей.
- 2 Реализация кроссплатформенного приложения с использованием локальной базы данных.

Задачи курсового проекта:

- 1 Изучение существующих технологий хранения данных.
- 2 Анализ требований к системе, установка основных критериев ее функционирования и надежности.
- 3 Разработка архитектуры системы, определение ее основных компонентов и интерфейсов.
- 4 Разработка и реализация кроссплатформенного приложения.

## СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

- [1] Top Programming Languages 2022 [Электронный ресурс]. – Режим доступа: <https://spectrum.ieee.org/top-programming-languages-2022>. Дата доступа: 01.05.2023
- [2] SQLite [Электронный ресурс]. – Режим доступа: <https://www.sqlite.org> – Дата доступа 01.05.2023.
- [3] Гради Буч Объектно-ориентированный анализ и проектирование с примерами приложений, 2017. – 780 с.
- [4] Бертран Мейер Почувствуй класс. Учимся программировать хорошо с объектами и контрактами, 2018. – 540 с.
- [5] Metanit. MAUI Framework [Электронный ресурс]. – Режим доступа: <https://metanit.com/sharp/maui> – Дата доступа 17.04.2023.
- [6] GitHub [Электронный ресурс]. – Режим доступа: <https://github.com> – Дата доступа: 10.04.2023.
- [7] Entity Framework [Электронный ресурс]. – Режим доступа: <https://learn.microsoft.com/ru-ru/ef> – Дата доступа: 21.04.2023.
- [8] Программная платформа .NET 7 [Электронный ресурс]. – Режим доступа: <https://learn.microsoft.com/ru-ru/dotnet> – Дата доступа: 27.04.2023.
- [9] What is SQLite [Электронный ресурс]. – Режим доступа: <https://www.linode.com/docs/guides/what-is-sqlite> – Дата доступа: 03.05.2023.

## ПРИЛОЖЕНИЕ А

### (обязательное)

### Листинг кода

#### Файл AppDbContext.cs

```
using Microsoft.EntityFrameworkCore;
using Motor_Lounge.Entities.Converters;
using Motor_Lounge.Entities.Cars;
using Motor_Lounge.Entities.Helpers;
using Motor_Lounge.Entities.Users;
using Application = Motor_Lounge.Entities.Users.Application;
namespace Motor_Lounge.Data
{
    public class AppDbContext : DbContext
    {
        public DbSet<Car> Cars => Set<Car>();
        public DbSet<User> Users => Set<User>();
        public DbSet<Information> News => Set<Information>();
        public DbSet<Application> Applications => Set<Application>();
        public AppDbContext(DbContextOptions<AppDbContext> options)
            : base(options)
        {
            Database.EnsureCreated();
        }
        protected override void OnModelCreating(ModelBuilder
modelBuilder)
        {
            modelBuilder.Entity<Car>()
                .Property(c => c.Price)
                .HasConversion(new PriceConverter());
            modelBuilder.Entity<Car>()
                .Property(c => c.Information)
                .HasConversion(new InformationConverter());
            modelBuilder.Entity<Car>()
                .Property(c => c.Photos)
                .HasConversion(new PhotosConverter());
            modelBuilder.Entity<Car>()
                .Property(c => c.Appearance)
                .HasConversion(new AppearanceConverter());
            modelBuilder.Entity<Car>()
                .Property(c => c.Characteristics)
                .HasConversion(new CharacteristicsConverter());
            modelBuilder.Entity<Car>()
                .Property(c => c.Equipment)
                .HasConversion(new EquipmentConverter());
            modelBuilder.Entity<Car>()
                .Property(c => c.Specification)
                .HasConversion(new SpecificationConverter());
            modelBuilder.Entity<Car>()
                .Property(c => c.Specification)
```



```

        .HasConversion(new SpecificationConverter());
        modelBuilder.Entity<Application>().Property(x
x.CarId).IsRequired();
        modelBuilder.Entity<Application>().Property(x
x.UserEmail).IsRequired();
        //for offers on main page
        modelBuilder.Entity<Information>().Property(x
x.Info).IsRequired();
    }
}
}

```

## Файл Repository.cs

```

using Microsoft.EntityFrameworkCore;
using Motor_Lounge.Entities.Cars;
using Motor_Lounge.Entities.Helpers;
using Motor_Lounge.Entities.Users;
using Application = Motor_Lounge.Entities.Users.Application;
namespace Motor_Lounge.Data
{
    public class UserRepository : IRepository<User>
    {
        private readonly DbContext dbContext;
        private readonly DbSet<User> entities;
        public UserRepository(DbContext _dbContext)
        {
            dbContext = _dbContext;
            entities = dbContext.Set<User>();
        }
        public async Task AddAsync(User entity, CancellationToken
cancellationTokен = default)
        {
            await entities.AddAsync(entity, cancellationTokен);
        }
        public void DeleteAsync(User entity, CancellationToken
cancellationTokен = default)
        {
            if (entity == null)
            {
                throw new ArgumentNullException(nameof(entity), "Entity
cannot be null");
            }
            entities.Remove(entity);
        }
        public async Task<User>
FirstAsync(System.Linq.Expressions.Expression<Func<User, bool>> filter,
CancellationToken cancellationTokен = default)
        {
            return await entities.FirstAsync(filter, cancellationTokен);
        }
        public async Task<User> GetByIdAsync(int id, CancellationToken
cancellationTokен = default,
params

```

```

System.Linq.Expressions.Expression<Func<User, object>>[]? includesProperties)
{
    var query = entities.AsQueryable();
    if (includesProperties != null)
    {
        query = includesProperties.Aggregate(query, (current,
includeProperty) => current.Include(includeProperty));
    }
    return await query.FirstAsync(e => e.Id == id,
cancellationTokens);
}

public async Task<User> GetByEmailAsync(string email,
CancellationTokens cancellationTokens = default, params
System.Linq.Expressions.Expression<Func<User, object>>[]? includesProperties)
{
    var query = entities.AsQueryable();
    if (includesProperties != null)
    {
        query = includesProperties.Aggregate(query, (current,
includeProperty) => current.Include(includeProperty));
    }
    return await query.FirstAsync(e => e.Email == email,
cancellationTokens);
}

public async Task<IReadOnlyList<User>>
ListAllAsync(CancellationTokens cancellationTokens = default)
{
    return await entities.ToListAsync(cancellationTokens);
}

public async Task<IReadOnlyList<User>>
ListAsync(System.Linq.Expressions.Expression<Func<User, bool>> filter,
CancellationTokens cancellationTokens = default, params
System.Linq.Expressions.Expression<Func<User, object>>[]? includesProperties)
{
    var query = entities.Where(filter);
    if (includesProperties != null)
    {
        query = includesProperties.Aggregate(query, (current,
includeProperty) => current.Include(includeProperty));
    }
    return await query.ToListAsync(cancellationTokens);
}

public async Task UpdateAsync(User entity, CancellationTokens
cancellationTokens = default)
{
    var old = await GetByIdAsync(entity.Id);
    dbContext.Entry(old).CurrentValues.SetValues(entity);
}
}

public class CarRepository : IRepository<Car>
{
    private readonly DbContext dbContext;

```

```

        private readonly DbSet<Car> entities;
        public CarRepository(DbContext _dbContext)
        {
            dbContext = _dbContext;
            entities = dbContext.Set<Car>();
        }
        public async Task AddAsync(Car entity, CancellationToken
cancellationTokentoken = default)
        {
            await entities.AddAsync(entity, cancellationTokentoken);
        }
        public void DeleteAsync(Car entity, CancellationToken
cancellationTokentoken = default)
        {
            if (entity == null)
            {
                throw new ArgumentNullException(nameof(entity), "Entity
cannot be null");
            }
            entities.Remove(entity);
        }
        public async Task<Car>
FirstAsync(System.Linq.Expressions.Expression<Func<Car, bool>> filter,
CancellationToken cancellationTokentoken = default)
        {
            return await entities.FirstAsync(filter, cancellationTokentoken);
        }
        public async Task<Car> GetByIdAsync(int id, CancellationToken
cancellationTokentoken = default, params
System.Linq.Expressions.Expression<Func<Car, object>>[]? includesProperties)
        {
            var query = entities.AsQueryable();
            if (includesProperties != null)
            {
                query = includesProperties.Aggregate(query, (current,
includeProperty) => current.Include(includeProperty));
            }
            return await query.FirstAsync(e => e.Id == id,
cancellationTokentoken);
        }
        public async Task<IReadOnlyList<Car>>
ListAllAsync(CancellationToken cancellationTokentoken = default)
        {
            return await entities.ToListAsync(cancellationTokentoken);
        }
        public async Task<IReadOnlyList<Car>>
ListAsync(System.Linq.Expressions.Expression<Func<Car, bool>> filter,
CancellationToken cancellationTokentoken = default, params
System.Linq.Expressions.Expression<Func<Car, object>>[]? includesProperties)
        {
            var query = entities.Where(filter);
            if (includesProperties != null)
            {
                query = includesProperties.Aggregate(query, (current,

```

```

includeProperty) => current.Include(includeProperty));
    }
    return await query.ToListAsync(cancellationToken);
}
public async Task UpdateAsync(Car entity, CancellationToken
cancellationToken = default)
{
    var old = await GetByIdAsync(entity.Id);
    dbContext.Entry(old).CurrentValues.SetValues(entity);
}
}
public class InformationRepository : IRepository<Information>
{
    private readonly DbContext dbContext;
    private readonly DbSet<Information> entities;
    public InformationRepository(DbContext _dbContext)
    {
        dbContext = _dbContext;
        entities = dbContext.Set<Information>();
    }
    public async Task AddAsync(Information entity, CancellationToken
cancellationToken = default)
    {
        await entities.AddAsync(entity, cancellationToken);
    }
    public void DeleteAsync(Information entity, CancellationToken
cancellationToken = default)
    {
        if (entity == null)
        {
            throw new ArgumentNullException(nameof(entity), "Entity
cannot be null");
        }
        entities.Remove(entity);
    }
    public async Task<Information>
FirstAsync(System.Linq.Expressions.Expression<Func<Information, bool>> filter,
CancellationToken cancellationToken = default)
    {
        return await entities.FirstAsync(filter, cancellationToken);
    }
    public async Task<Information> GetByIdAsync(int id,
CancellationToken cancellationToken = default, params
System.Linq.Expressions.Expression<Func<Information,
object>>[]?
includesProperties)
    {
        var query = entities.AsQueryable();
        if (includesProperties != null)
        {
            query = includesProperties.Aggregate(query, (current,
includeProperty) => current.Include(includeProperty));
        }
        return await query.FirstAsync(e => e.Id == id,
cancellationToken);
    }
}

```

```

        }
        public async Task<IReadOnlyList<Information>>
ListAllAsync(CancellationToken cancellationToken = default)
        {
            return await entities.ToListAsync(cancellationToken);
        }
        public async Task<IReadOnlyList<Information>>
ListAsync(System.Linq.Expressions.Expression<Func<Information, bool>> filter,
CancellationTokens cancellationToken = default, params
System.Linq.Expressions.Expression<Func<Information, bool>>[]?
includesProperties)
        {
            var query = entities.Where(filter);
            if (includesProperties != null)
            {
                query = includesProperties.Aggregate(query, (current,
includeProperty) => current.Include(includeProperty));
            }
            return await query.ToListAsync(cancellationToken);
        }
        public async Task UpdateAsync(Information entity,
CancellationTokens cancellationToken = default)
        {
            var old = await GetByIdAsync(entity.Id);
            dbContext.Entry(old).CurrentValues.SetValues(entity);
        }
    }
    public class ApplicationRepository : IRepository<Application>
    {
        private readonly DbContext dbContext;
        private readonly DbSet<Application> entities;

        public ApplicationRepository(DbContext _dbContext)
        {
            dbContext = _dbContext;
            entities = dbContext.Set<Application>();
        }
        public async Task AddAsync(Application entity, CancellationTokens
cancellationToken = default)
        {
            await entities.AddAsync(entity, cancellationToken);
        }
        public void DeleteAsync(Application entity, CancellationTokens
cancellationToken = default)
        {
            if (entity == null)
            {
                throw new ArgumentNullException(nameof(entity), "Entity
cannot be null");
            }
            entities.Remove(entity);
        }
        public async Task<Application>
FirstAsync(System.Linq.Expressions.Expression<Func<Application, bool>> filter,

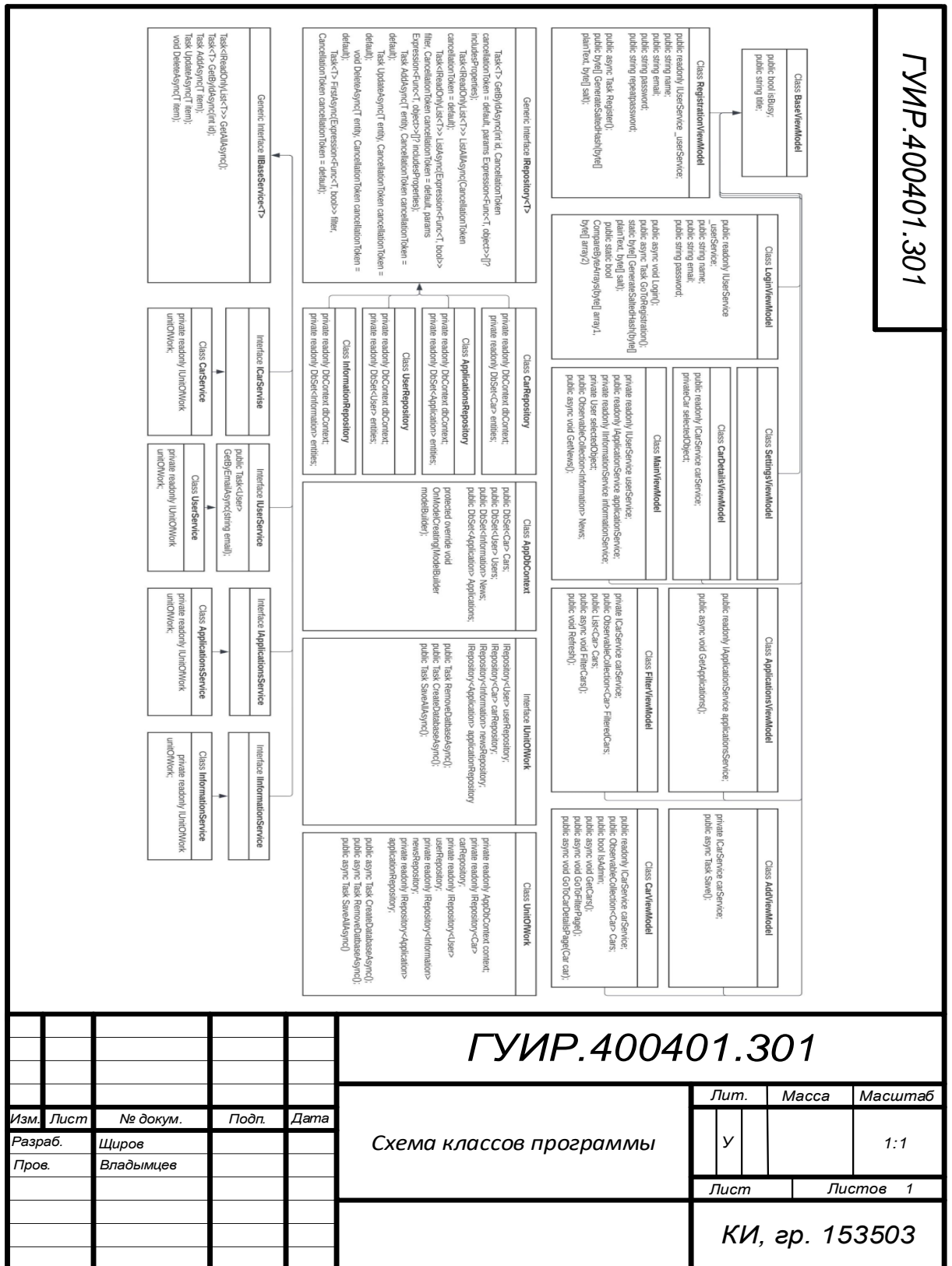
```

```

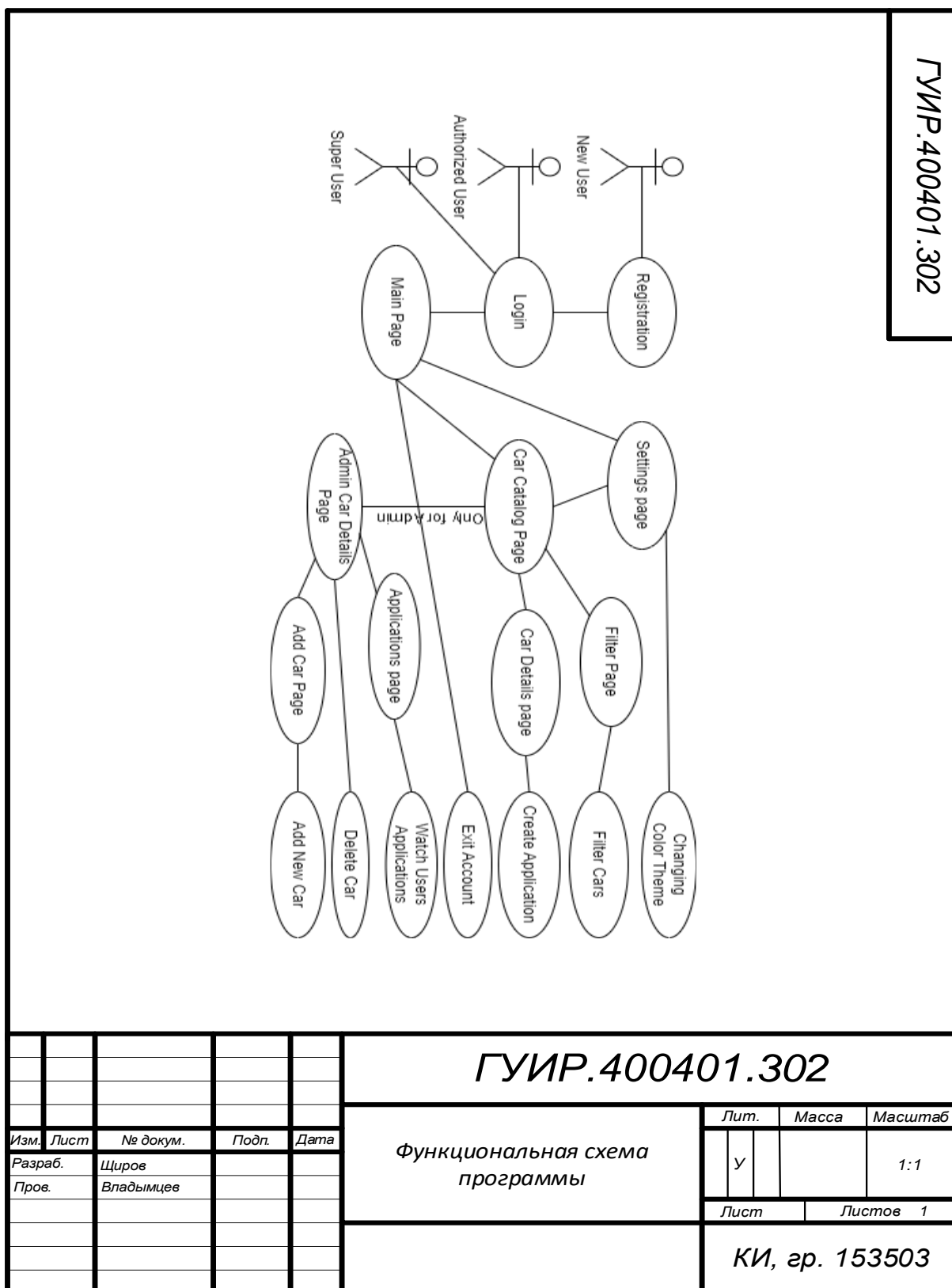
CancellationToken cancellationToken = default)
    {
        return await entities.FirstAsync(filter, cancellationToken);
    }
    public async Task<Application> GetByIdAsync(int id,
CancellationToken cancellationToken = default, params
System.Linq.Expressions.Expression<Func<Application,
object>>[]?
includesProperties)
    {
        var query = entities.AsQueryable();
        if (includesProperties != null)
        {
            query = includesProperties.Aggregate(query, (current,
includeProperty) => current.Include(includeProperty));
        }
        return await query.FirstAsync(e => e.Id == id,
cancellationToken);
    }
    public async Task<IReadOnlyList<Application>>
ListAllAsync(CancellationToken cancellationToken = default)
    {
        return await entities.ToListAsync(cancellationToken);
    }
    public async Task<IReadOnlyList<Application>>
ListAsync(System.Linq.Expressions.Expression<Func<Application, bool>> filter,
CancellationToken cancellationToken = default, params
System.Linq.Expressions.Expression<Func<Application,
object>>[]?
includesProperties)
    {
        var query = entities.Where(filter);
        if (includesProperties != null)
        {
            query = includesProperties.Aggregate(query, (current,
includeProperty) => current.Include(includeProperty));
        }
        return await query.ToListAsync(cancellationToken);
    }
    public async Task UpdateAsync(Application entity,
CancellationToken cancellationToken = default)
    {
        var old = await GetByIdAsync(entity.Id);
        dbContext.Entry(old).CurrentValues.SetValues(entity);
    }
}
}

```

ПРИЛОЖЕНИЕ Б  
(обязательное)  
Схема классов программы




# **ПРИЛОЖЕНИЕ В** **(обязательное)** **Функциональная схема программы**



Формат А4



ГУИР.400401.303

 | MotorLounge

Sign in to continue

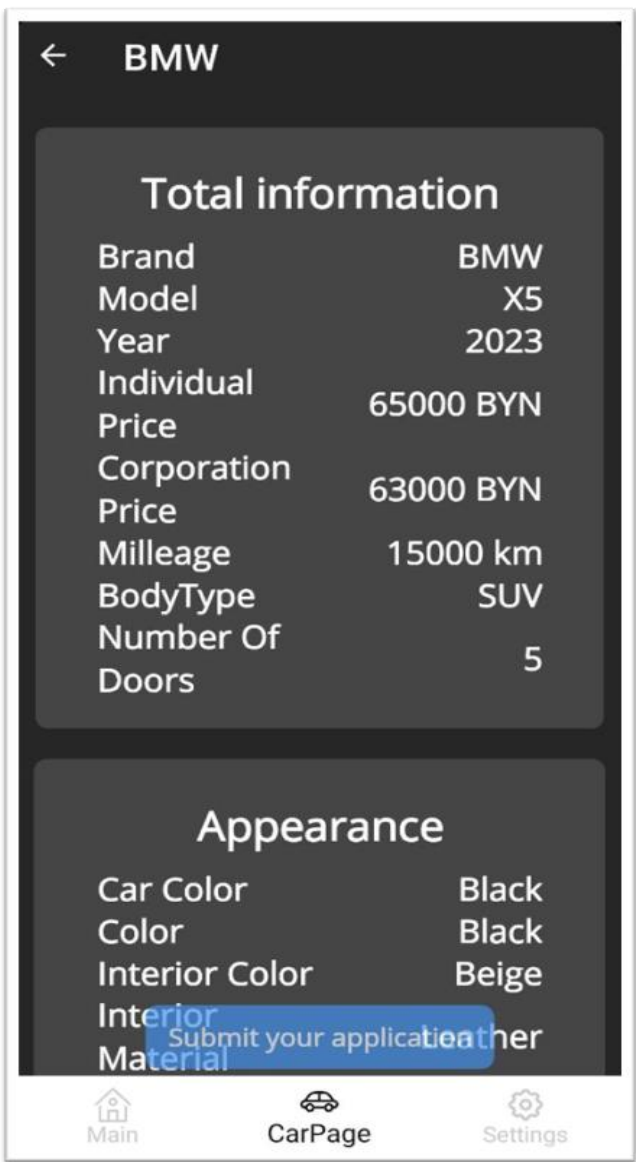



Sign in

New user? [Sign Up](#)

						ГУИР.400401.303							
						Графический интерфейс страницы входа							
											Лит.	Масса	Масштаб
											у		1:1
Изм.	Лист	№ докум.	Подп.	Дата									
Разраб.		Щигов											
Пров.		Владимцев											

Формат А4

**ПРИЛОЖЕНИЕ Д**  
**(обязательное)**  
**Темная тема программы**

ГУИР.400401.304													
													
<div style="display: flex; justify-content: space-around; align-items: center;"> <div style="text-align: center;">  Main         </div> <div style="text-align: center;">  CarPage         </div> <div style="text-align: center;">  Settings         </div> </div>													
ГУИР.400401.304													
Темная тема программы			<table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <th style="width: 10%;">Лит.</th> <th style="width: 10%;">Масса</th> <th style="width: 10%;">Масштаб</th> </tr> <tr> <td style="text-align: center;">У</td> <td></td> <td style="text-align: center;">1:1</td> </tr> <tr> <td colspan="2" style="text-align: center;">Лист</td> <td style="text-align: center;">Листов 1</td> </tr> </table>		Лит.	Масса	Масштаб	У		1:1	Лист		Листов 1
Лит.	Масса	Масштаб											
У		1:1											
Лист		Листов 1											
КИ, гр. 153503													

Формат А4