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:

**«Каталог товаров»**

БГУИР КП 1-40 04 01 61 ПЗ

|  |  |
| --- | --- |
|  | Выполнил студент группы 153503  Кончик Денис Сергеевич  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись студента) |
|  | Курсовой проект представлен  на проверку \_\_\_.\_\_\_\_.2023  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись студента) |

Минск 2023

**СОДЕРЖАНИЕ**

[ВВЕДЕНИЕ 2](#_Toc702603608)

[1 Архитектура вычислительной системы 3](#_Toc791370142)

[2 Платформа программного обеспечения 5](#_Toc862657394)

[3 Теоретическое обоснование разработки программного продукта 14](#_Toc776730184)

[4 Проектирование функциональных возможностей программы 20](#_Toc1935942257)

[5 Архитектура разрабатываемой программы 24](#_Toc1512791070)

[ЗАКЛЮЧЕНИЕ 32](#_Toc1655140668)

[СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ 33](#_Toc1117560812)

[ПРИЛОЖЕНИЕ А(обязательное)Исходный код программы 35](#_Toc1034860831)

[ПРИЛОЖЕНИЕ Б(обязательное)Функциональная схема алгоритма 46](#_Toc249836718)

[ПРИЛОЖЕНИЕ В(обязательное)Блок схема программы 47](#_Toc786255223)

[ПРИЛОЖЕНИЕ Г(обязательное)Графический интерфейс 48](#_Toc1794817091)

[ПРИЛОЖЕНИЕ Д(обязательное)Ведомость 49](#_Toc882122453)

****ВВЕДЕНИЕ****

В настоящее время электронная коммерция становится все более популярной и востребованной. Интернет-магазины предлагают удобный способ для покупателей приобрести различные товары и услуги, не выходя из дома. Для успешного функционирования интернет-магазина необходимо иметь эффективную систему управления каталогом товаров, которая позволяет добавлять, редактировать и удалять товары, а также обеспечивает удобный интерфейс для покупателей.

Целью данного курсового проекта является разработка каталога товаров для интернет-магазина с использованием технологии ASP.NET MVC. Проект направлен на создание функционального и удобного веб-приложения, которое позволяет покупателям осуществлять поиск и выбор товаров, добавлять их в корзину и осуществлять покупку. Продавцы, в свою очередь, должны иметь возможность добавлять, удалять и редактировать товары, относящиеся к их магазинам. Администратор должен обладать полным доступом ко всем функциональным возможностям системы, включая управление пользователями и магазинами.

Задачи курсового проекта:

1. Изучение принципов объектно-ориентированного программирования и архитектурного подхода MVC (Model-View-Controller).
2. Разработка архитектуры системы, включающей разделение на уровни данных (DAL), бизнес-логики (BLL) и веб-приложения (WEB).
3. Создание моделей данных и определение их связей, чтобы представлять товары, пользователей, магазины и другие сущности в системе.
4. Реализация системы ролей и прав доступа, интерфейса для каждой из ролей.
5. Применение технологии Entity Framework для взаимодействия с базой данных, хранящей информацию о сущностях программы.
6. Использование языка разметки HTML, шаблонизатора Razor и фреймворка Bootstrap для создания удобного пользовательского интерфейса приложения.

В результате выполнения курсового проекта будет разработан функциональный и удобный в использовании каталог товаров для интернет-магазина на основе технологии ASP.NET MVC. Пользователи смогут легко находить и приобретать товары, а администраторы и продавцы получат удобные инструменты для управления системой.

# Анализ используемых источников

*"CLR via C#" - Джеффри Рихтер*. Эта книга является одним из авторитетных руководств по разработке на языке C# и платформе .NET. Автор освещает внутреннее устройство и работу среды CLR (Common Language Runtime) и предлагает глубокое понимание основных концепций и возможностей языка C#.

*Metanit*. Данный веб-сайт предлагает обширные обучающие материалы по различным аспектам разработки на языке C#. Здесь можно найти подробные руководства, примеры кода и объяснения различных тем, включая основы языка, работу с базами данных, создание веб-приложений и другие.

*Документация Bootstrap*. Bootstrap является популярным фреймворком для разработки веб-интерфейсов. Официальная документация Bootstrap предоставляет подробное описание компонентов, классов, шаблонов и инструментов, которые помогут вам создавать красивые и отзывчивые веб-страницы.

*Документация Microsoft по ASP.NET MVC*. Официальная документация Microsoft по ASP.NET MVC содержит подробное описание архитектуры, основных компонентов и функциональности фреймворка. Она также включает примеры кода и руководства для создания мощных веб-приложений.

*"Язык программирования C# 7 и платформы .NET и .NET Core"*. Эндрю Троелсен. Эта книга является исчерпывающим руководством по разработке на языке C# 7 и платформе .NET и .NET Core. Автор предлагает подробное объяснение основных концепций языка и платформы, включая типы данных, синтаксис, обработку исключений, работу с файлами и другие важные аспекты программирования на C#.

*Документация Microsoft по Entity Framework*. Entity Framework (EF) является фреймворком для доступа к данным в приложениях на платформе .NET. Официальная документация Entity Framework от Microsoft предоставляет всестороннюю информацию о возможностях и функциональности фреймворка, а также руководства по его использованию.

# Теоретическое обоснование разработки

В данном разделе представлено теоретическое обоснование разработки проекта, которое включает в себя описание основных технологий и инструментов, используемых в процессе разработки. Для данного проекта были выбраны следующие основные компоненты: язык программирования C#, платформа .NET, база данных SQLite, ASP.NET и Entity Framework. Для создания удобного пользовательского интерфейса представлений был использован язык разметки HTML и язык JavaScript, шаблонизатор Razor и фреймворк Bootstrap.

C# (C Sharp) является объектно-ориентированным языком программирования, разработанным компанией Microsoft. Он был представлен в 2000 году в рамках платформы .NET Framework и с тех пор стал одним из основных языков для разработки приложений на этой платформе.

Основные характеристики C#:

1. Объектно-ориентированная парадигма: C# поддерживает основные принципы объектно-ориентированного программирования, такие как наследование, полиморфизм и инкапсуляция. Это позволяет разработчикам создавать модульные, гибкие и расширяемые приложения. Классы являются основными строительными блоками программы, а объекты представляют экземпляры этих классов.
2. Безопасность типов: C# является языком со строгой типизацией, что означает, что каждая переменная и выражение имеют определенный тип данных. Это помогает предотвратить ошибки типов во время компиляции, улучшает надежность программы и повышает безопасность выполнения приложений.
3. Управление памятью: В C# используется автоматическое управление памятью с помощью сборки мусора. Это означает, что разработчику не нужно явно освобождать память после использования объектов, так как сборщик мусора автоматически определяет, когда объекты больше не используются и освобождает память, занимаемую ими. Это упрощает разработку, уменьшает количество ошибок, связанных с управлением памятью, и повышает производительность приложений.
4. Многоязыковая поддержка: C# может взаимодействовать с другими языками, поддерживаемыми платформой .NET, такими как Visual Basic .NET и F#. Это позволяет разработчикам использовать различные языки в рамках одного приложения в зависимости от их предпочтений и требований проекта. Также это облегчает переиспользование кода и интеграцию с существующими проектами на разных языках.
5. Богатая стандартная библиотека: C# имеет обширную стандартную библиотеку классов, которая предоставляет различные функциональные возможности для работы с файлами, сетью, базами данных, графикой и другими аспектами разработки приложений. Благодаря этому разработчикам необходимо создавать множество функций "с нуля", а они могут использовать готовые классы и методы из стандартной библиотеки. Это существенно упрощает разработку приложений и повышает производительность разработчика.
6. Поддержка асинхронного программирования: C# предоставляет инструменты для работы с асинхронными операциями. Ключевые слова async и await позволяют создавать асинхронные методы, которые могут выполняться параллельно и не блокируют главный поток выполнения. Это особенно полезно для разработки реактивных и отзывчивых приложений, а также для улучшения производительности и отзывчивости пользовательского интерфейса.
7. Мощные инструменты разработки: Для разработки приложений на C# существует множество интегрированных сред разработки (IDE), таких как Microsoft Visual Studio и Visual Studio Code, которые предоставляют широкий набор инструментов для отладки, автодополнения кода, управления проектами и других задач разработки. Эти инструменты значительно упрощают разработку и улучшают производительность разработчика.
8. Поддержка различных платформ: C# поддерживает разработку приложений для разных платформ, включая Windows, macOS и Linux. Благодаря использованию платформы .NET Core, разработчики могут создавать переносимый код, который может быть запущен на разных операционных системах без необходимости переписывания значительной части приложения.

Язык программирования C# является мощным инструментом для разработки разнообразных приложений, обладает удобным синтаксисом, богатыми возможностями и широкой поддержкой со стороны Microsoft и сообщества разработчиков.

.NET (от англ. "dot net") - это кроссплатформенная платформа для разработки программного обеспечения, разработанная компанией Microsoft. Она предоставляет средства и инфраструктуру для создания разнообразных типов приложений, включая настольные приложения, веб-приложения, мобильные приложения, игры, облачные сервисы и IoT-решения.

Платформа .NET была представлена в 2002 году Microsoft как инновационное решение для разработки приложений под Windows. Она включала в себя .NET Framework, среду выполнения CLR (Common Language Runtime) и языки программирования, такие как C# и Visual Basic .NET. .NET Framework был предназначен для разработки приложений под Windows и был основным фреймворком для работы с платформой .NET в течение многих лет.

. NET Framework является классической реализацией платформы .NET. Он включает в себя обширную стандартную библиотеку классов (Base Class Library, BCL), которая предоставляет различные функциональные возможности для разработки приложений, такие как работа с файлами, сетью, базами данных, графикой и другими аспектами разработки. .NET Framework также включает CLR, которая обеспечивает управление памятью, компиляцию JIT (Just-In-Time) и другие важные функции исполнения кода. .NET Framework был основным фреймворком для разработки Windows-приложений до появления .NET Core.

.NET Core является новой платформой разработки приложений, которая была представлена Microsoft в 2016 году. Он является модульным и кроссплатформенным фреймворком, который можно использовать для создания приложений под Windows, Linux и macOS. .NET Core предлагает легковесный и быстрый подход к разработке приложений, а также предоставляет новые возможности, такие как поддержка асинхронного программирования и встроенная поддержка микросервисной архитектуры. .NET Core также включает в себя свою собственную версию стандартной библиотеки классов, называемую "CoreFX".

Объединение в .NET: В 2019 году Microsoft объединила .NET Framework и .NET Core в единую платформу, называемую просто .NET. Это объединение произошло в рамках релиза .NET 5.0, и оно призвано упростить разработку приложений и сделать платформу .NET более согласованной и единообразной. Теперь .NET включает в себя общую стандартную библиотеку классов (Base Class Library), общий CLR и общие инструменты разработки. Это позволяет разработчикам использовать одни и те же навыки и инструменты для создания приложений под разные операционные системы и устройства.

Платформа .NET поддерживает несколько языков программирования, включая C#, Visual Basic .NET, F# и другие. C# является основным языком для разработки приложений под платформу .NET. Он предоставляет современные возможности объектно-ориентированного программирования, сильную типизацию, поддержку асинхронного программирования и многое другое. Visual Basic .NET является еще одним популярным языком, особенно для разработки приложений с использованием графического интерфейса. F# - функциональный язык программирования, который также поддерживается платформой .NET и обладает сильными возможностями для анализа данных и параллельного программирования.

Для разработки приложений на платформе .NET доступно множество интегрированных сред разработки (IDE). Microsoft Visual Studio является наиболее популярным и мощным инструментом разработки, предоставляющим широкий набор функций для создания, отладки и развертывания приложений на платформе .NET. Однако также существуют и другие популярные IDE, такие как Visual Studio Code, которые обладают легковесным и гибким подходом к разработке на .NET.

Платформа .NET является мощным инструментом для разработки программного обеспечения. Сочетание языков программирования, CLR, библиотек классов и инструментов разработки обеспечивает разработчикам широкие возможности для создания качественных и масштабируемых приложений на разных операционных системах и устройствах.

ASP.NET - это фреймворк для разработки веб-приложений, разработанный компанией Microsoft. Он предоставляет разработчикам мощные инструменты и функциональные возможности для создания динамических, масштабируемых и безопасных веб-приложений.

Вот некоторые ключевые особенности и компоненты ASP.NET:

1. Модель программирования MVC: ASP.NET предлагает модель программирования Model-View-Controller (MVC), которая позволяет разделить приложение на три основных компонента: модель (Model), представление (View) и контроллер (Controller). Это обеспечивает логическую разделенность кода и улучшает его поддерживаемость и расширяемость.
2. Языки программирования: ASP.NET поддерживает несколько языков программирования, включая C#, Visual Basic .NET, F# и другие. Выбор языка зависит от предпочтений разработчика и требований проекта.
3. Серверная часть: ASP.NET позволяет разрабатывать серверную часть веб-приложения с использованием C# (или других языков программирования) и .NET-фреймворка. Вы можете создавать классы, методы, модели и сервисы для обработки запросов, взаимодействия с базой данных, бизнес-логики и других задач.
4. Шаблонизатор Razor: ASP.NET использует шаблонизатор Razor, который позволяет встраивать код C# (или других языков программирования) непосредственно в HTML-разметку представлений. Это упрощает создание динамических и интерактивных веб-страниц.
5. Библиотеки классов: ASP.NET предоставляет обширные библиотеки классов, которые содержат множество функциональных возможностей для разработки веб-приложений. Например, есть классы и методы для работы с сетью, базами данных, безопасностью, управлением сеансами и другими аспектами разработки.
6. Безопасность: ASP.NET обладает встроенными механизмами безопасности, такими как аутентификация, авторизация, обработка CSRF-атак и другие. Он также обеспечивает защиту от распространенных уязвимостей веб-приложений.
7. Интеграция с другими технологиями: ASP.NET хорошо интегрируется с другими технологиями и инструментами, такими как Entity Framework для работы с базами данных, Web API для создания API-интерфейсов, SignalR для реализации веб-сокетов и другими.

Использование ASP.NET MVC в данном проекте каталога товаров позволило создать структурированное и модульное веб-приложение с логическим разделением на модель, представление и контроллер. Это упрощает разработку, поддержку и тестирование приложения, а также позволяет эффективно обрабатывать запросы, взаимодействовать с базой данных и предоставлять пользователю удобный интерфейс.

Entity Framework - это объектно-ориентированный инструментарий для работы с данными в приложениях .NET. Он предоставляет разработчикам высокоуровневую абстракцию для доступа к данным и управления базами данных, позволяя работать с объектами и сущностями вместо прямых запросов к базе данных.

В контексте данного проекта была использована база данных SQLite, которая является легкой и компактной реляционной базой данных. Entity Framework предоставляет поддержку для SQLite и позволяет работать с ней, используя объектно-ориентированный подход.

Некоторые ключевые особенности и компоненты Entity Framework:

1. Модель данных: Entity Framework позволяет определить модель данных, которая отражает структуру и отношения таблиц в базе данных. Есть возможность использовать атрибуты или Fluent API для определения сущностей (таблиц), свойств и их отношений.
2. Code First и Database First подходы: Entity Framework поддерживает два основных подхода к разработке базы данных. Code First позволяет определить модель данных с помощью классов и атрибутов, а затем сгенерировать схему базы данных из этой модели. Database First подход позволяет сначала создать схему базы данных, а затем сгенерировать модель данных на основе этой схемы.
3. LINQ to Entities: Entity Framework предоставляет LINQ (Language Integrated Query) для создания запросов к данным. LINQ позволяет писать запросы, используя язык C# (или другие языки, поддерживаемые LINQ), что обеспечивает типобезопасные и компилируемые запросы к базе данных.
4. Миграции базы данных: Entity Framework предлагает механизм миграций, который позволяет обновлять схему базы данных по мере развития вашего приложения. Можно создавать и применять миграции, чтобы автоматически изменять структуру базы данных без необходимости вручную вносить изменения.
5. Управление отношениями: Entity Framework позволяет определять различные типы отношений между сущностями, такие как один-к-одному, один-ко-многим и многие-ко-многим. Также можно использовать навигационные свойства для удобной навигации по отношениям и выполнения запросов.
6. Ленивая загрузка и явная загрузка: Entity Framework поддерживает как ленивую загрузку (lazy loading), когда связанные данные загружаются по требованию, так и явную загрузку (eager loading), когда явно указывается, какие связанные данные должны быть загружены.

Использование Entity Framework с базой данных SQLite в данном проекте дало возможность эффективно работать с данными, использовать ORM-подход для доступа к базе данных и упростить взаимодействие с данными в приложении.

Когда мы говорим о создании веб-страниц, существует популярная комбинация инструментов, которая включает в себя HTML, JavaScript, cshtml (Razor) и Bootstrap. Давайте подробнее рассмотрим каждый из этих компонентов и их взаимодействие друг с другом.

HTML (HyperText Markup Language) является языком разметки, который служит основой для создания структуры и содержимого веб-страниц. Это означает, что с помощью HTML мы можем определить различные элементы на странице, такие как заголовки, абзацы, списки, таблицы, формы и многие другие. HTML определяет, как каждый из этих элементов будет отображаться на странице и как они будут взаимодействовать друг с другом.

JavaScript является языком программирования, который добавляет интерактивность и динамическое поведение на веб-страницах. Он предоставляет разработчикам возможность создавать веб-приложения, которые могут реагировать на действия пользователей и обмениваться данными с сервером. JavaScript может выполнять различные действия на странице, обрабатывать события, взаимодействовать с элементами страницы и многое другое. Он дополняет функциональность HTML, позволяя создавать более динамичный и интерактивный пользовательский опыт.

Razor (cshtml) является разметочным языком, который позволяет встраивать код на C# (или других языках программирования) непосредственно в HTML-разметку представлений. Это дает разработчикам возможность создавать динамические веб-страницы, генерировать контент на основе данных из базы данных, выполнять условные операторы и циклы, а также использовать встроенные помощники для упрощения создания кода. Razor позволяет интегрировать логику программирования в разметку страницы, делая ее более гибкой и мощной.

Bootstrap - это фреймворк для разработки пользовательского интерфейса веб-приложений. Он предоставляет набор предварительно стилизованных компонентов, сеточную систему, CSS-стили, JavaScript-плагины и другие инструменты, которые помогают создавать современные и отзывчивые интерфейсы. Bootstrap также предлагает множество возможностей для настройки и доработки дизайна, включая создание собственных стилей и компонентов. Он упрощает процесс разработки, позволяя разработчикам использовать готовые компоненты и стили, что сокращает время и усилия, необходимые для создания привлекательного пользовательского интерфейса.

Когда все эти компоненты используются вместе, они обеспечивают создание динамических и привлекательных веб-страниц. HTML используется для определения структуры и содержимого страницы, JavaScript добавляет интерактивность и обрабатывает события, cshtml (Razor) позволяет создавать динамический контент на основе данных, а Bootstrap предоставляет готовые стили и компоненты для стилизации и создания отзывчивого дизайна. Эта связка инструментов предоставляет разработчикам мощные средства для создания высококачественных веб-приложений с минимальными усилиями.

# 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 многоразовое 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, многоразового 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 конкркретные реализации фабрики для новых продуктов. Это позволяет легко внедрять изменения в систему без необходимости изменения клиентского кода.

3 Сокрытие 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 клиентскийкод 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.

# Функциональные возможности программы

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

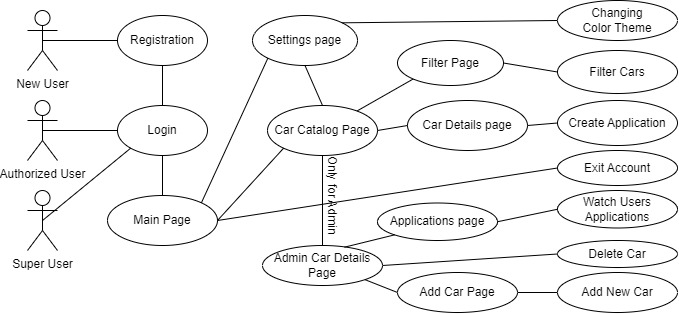
****

Рисунок 1 – Use-case диаграмма приложения

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

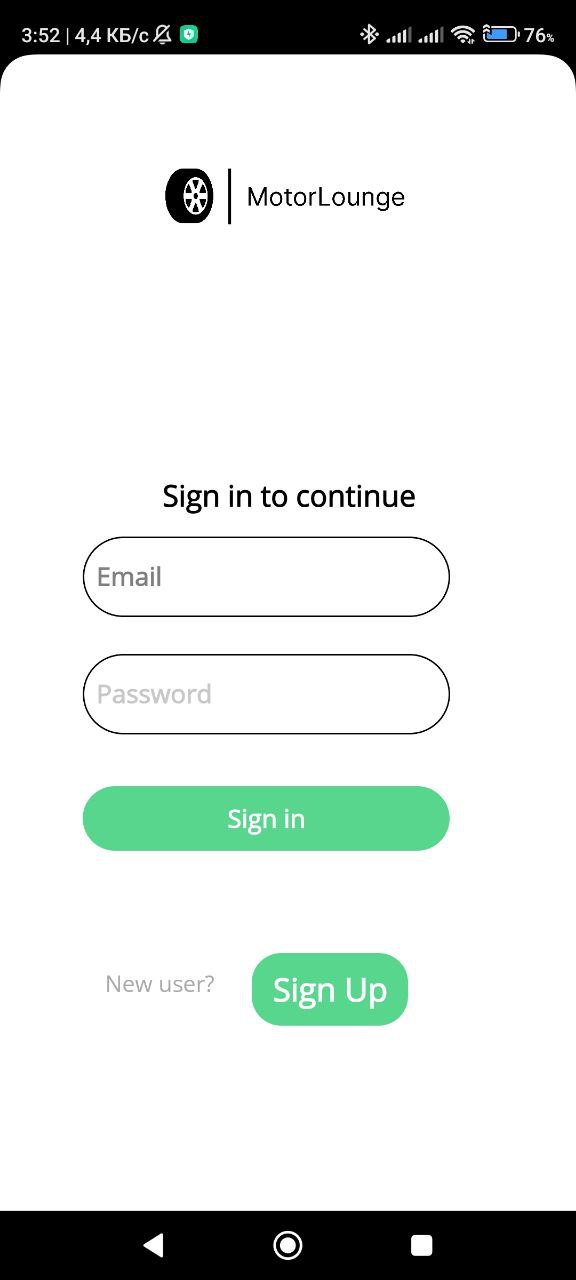


Рисунок 2 – Страница входа

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

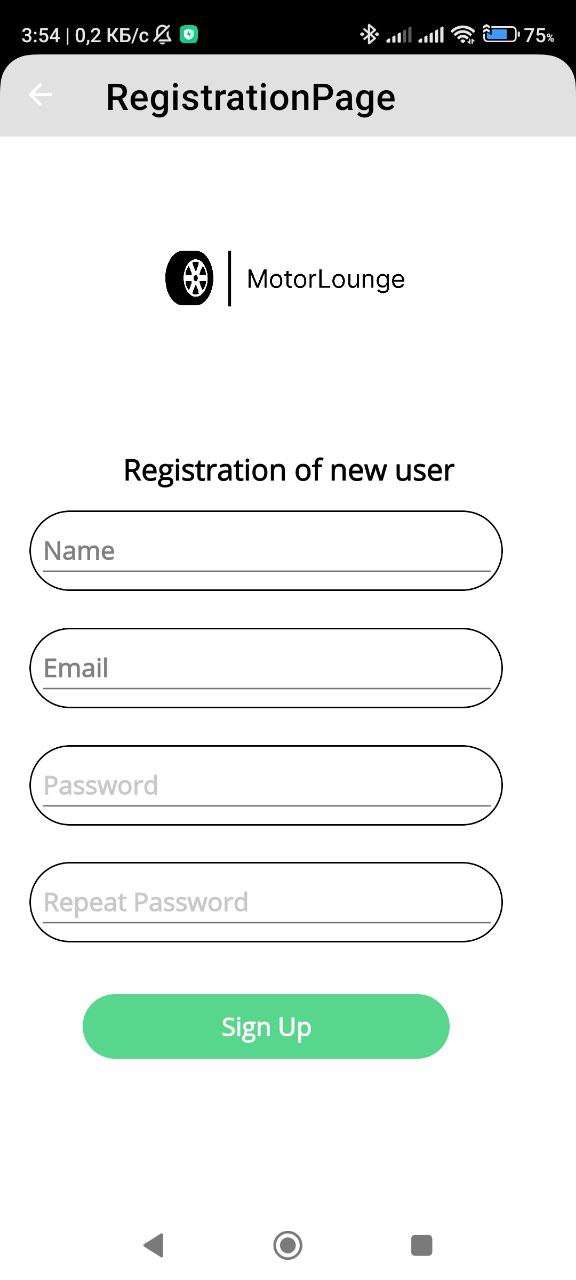


Рисунок 3 – Страница регистрации

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.

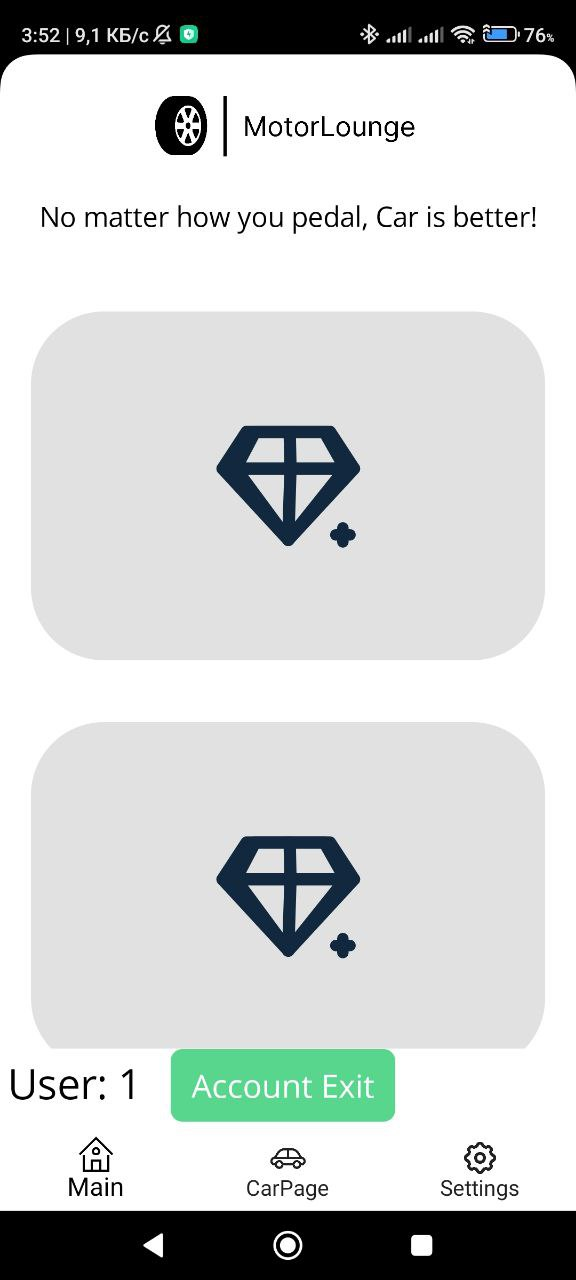


Рисунок 4 – Главная страница приложения

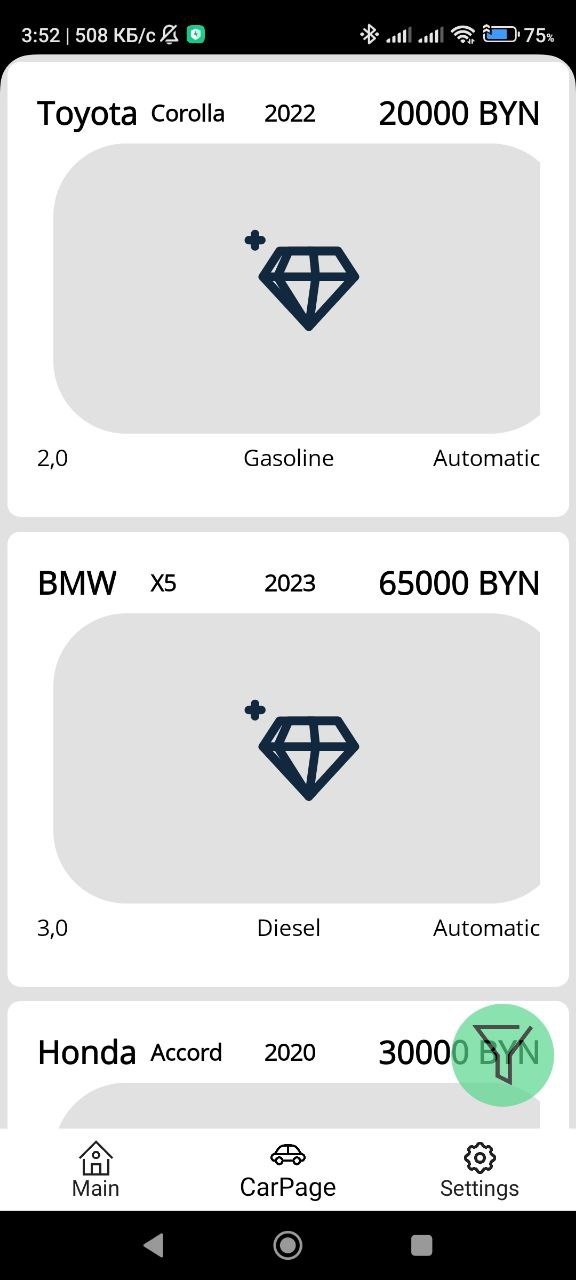


Рисунок 5 – Каталог автомобилей



Рисунок 6 – Страница настроек приложения

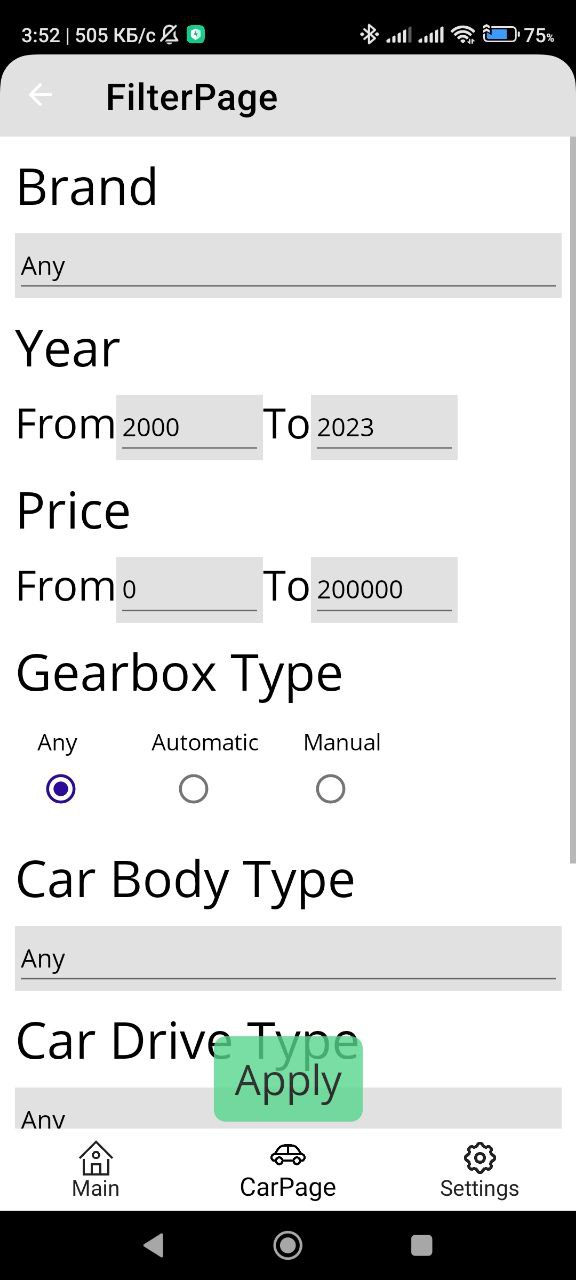


Рисунок 7 – Страница фильтра автомобилей

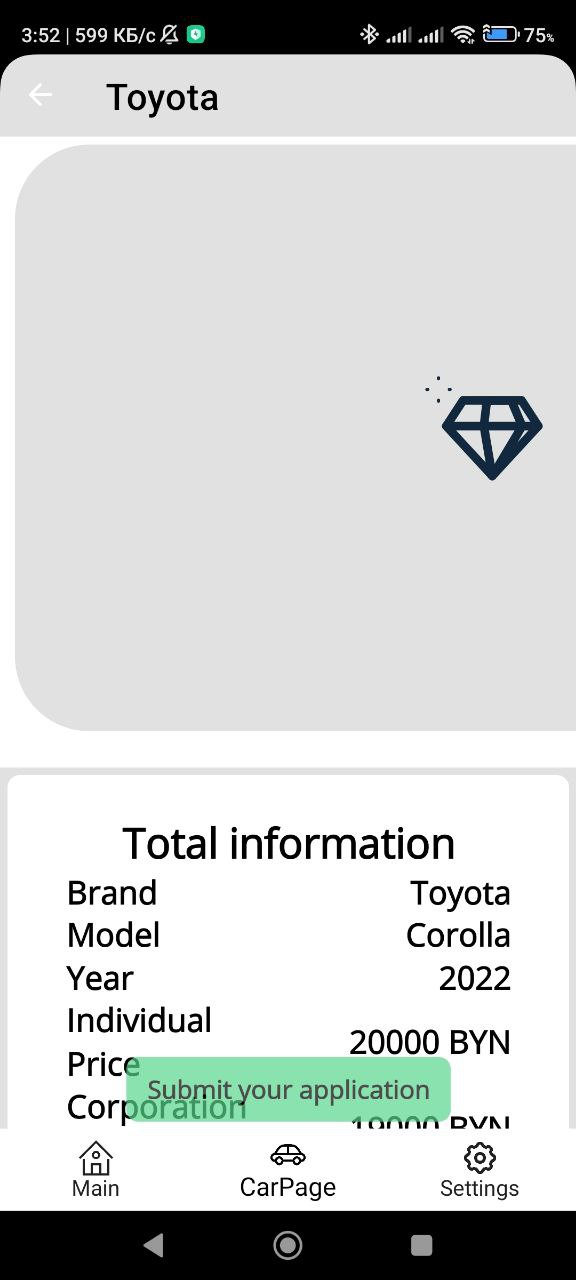


Рисунок 8 – Страница подробной информации

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 включенном темной темой приложения.

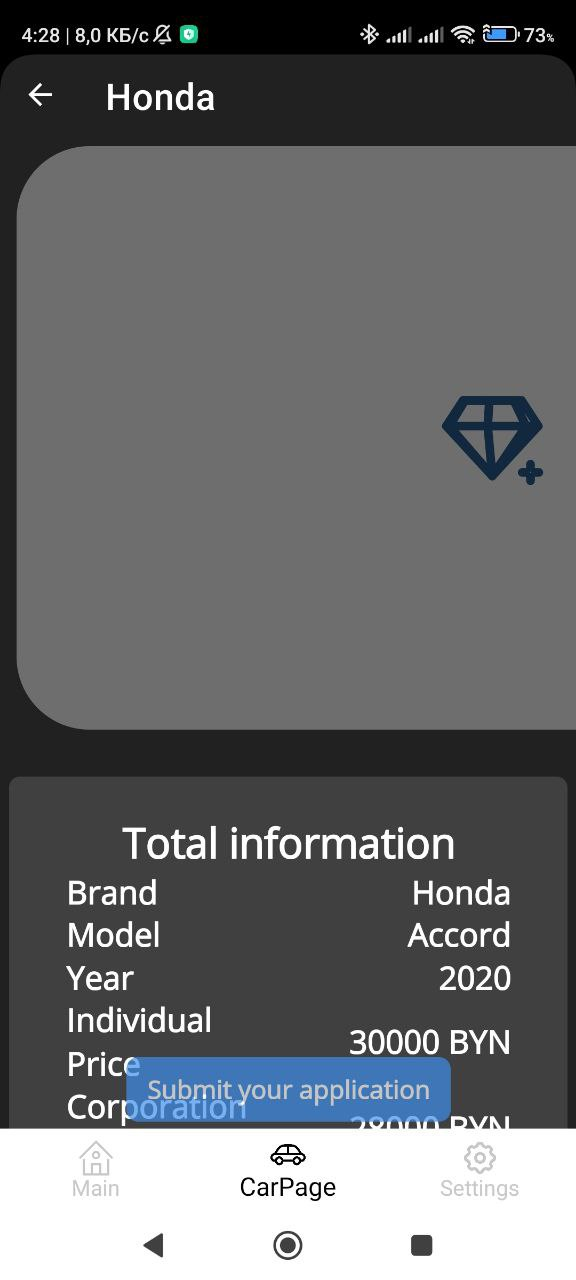


Рисунок 9 – Тёмная тема приложения

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.

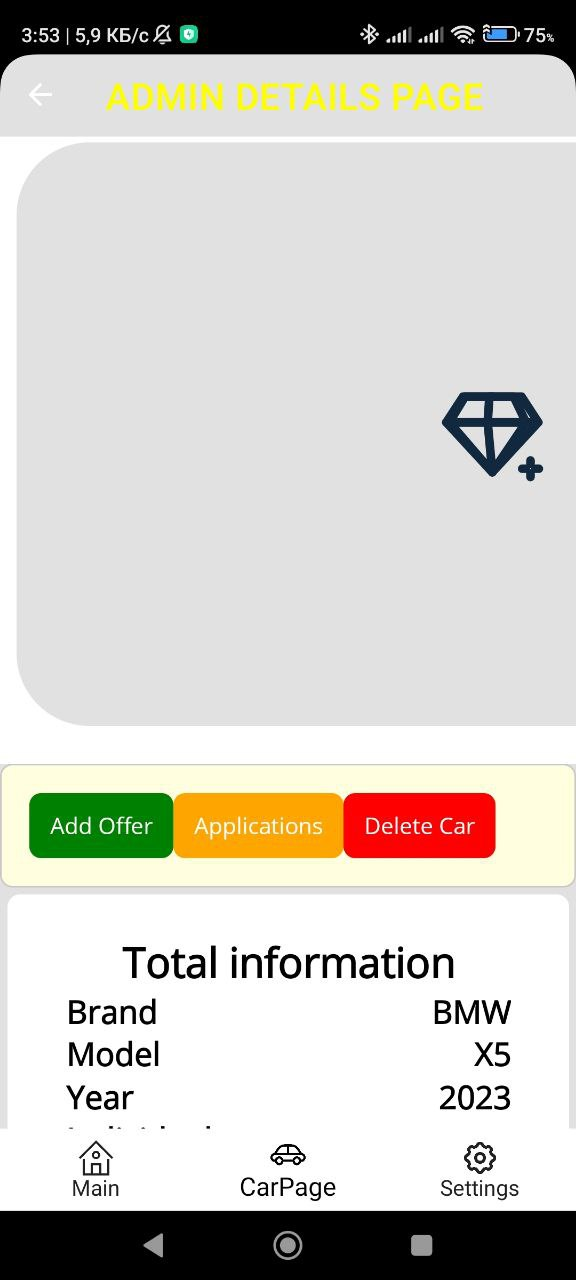


Рисунок 10 – Страница подробной информации глазами админа

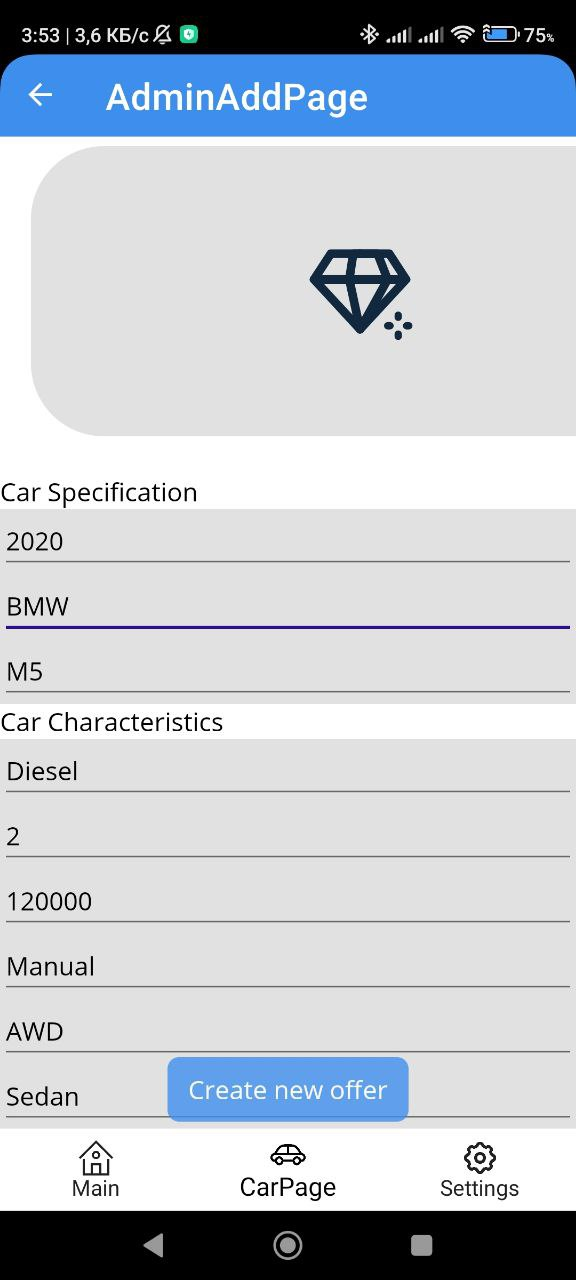


Рисунок 11 – Страница добавления в каталог нового автомобиля

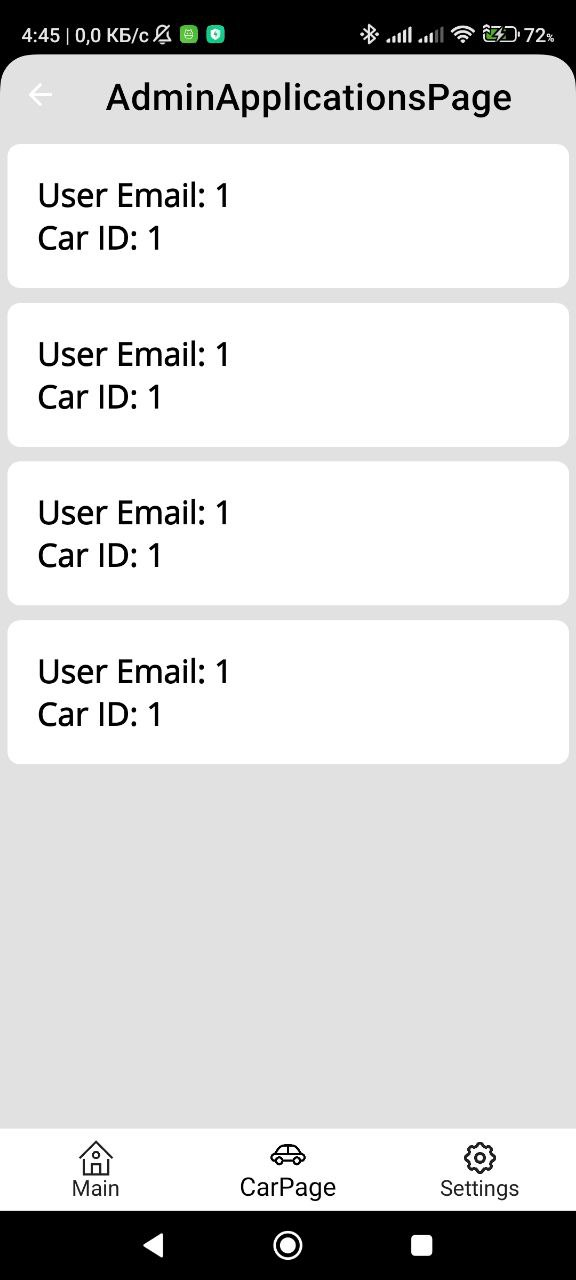


Рисунок 12 – Страница пользовательских заявок

# Архитектура разрабатываемой программы

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

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.

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

ЗАКЛЮЧЕНИЕ

В данной курсовой работе была рассмотрена тема "Кроссплатформенное приложение автосалона". Были рассмотрены основные принципы и подходы к разработке такой системы, а также технологии, используемые для ее создания.

В ходе работы были изучены различные аспекты разработки приложений, включая безопасность, шифрование, установление соединения с базой данных, работу с ролями, а также графический интерфейс пользователя. В процессе работы была разработана кроссплатформенное приложение автосалона, обеспечивающее безопасность и защиту данных пользователей. Все возможности и функционал системы были протестированы и успешно проверены. В целом, разработка кроссплатформенного приложения автосалона является актуальной темой, так как хороших приложений такой направленности мало, и пользователи часто встречаются . Разработка такой системы обмена сообщениями, обеспечивающей безопасность и защиту данных пользователей, является важным шагом в обеспечении безопасности в сфере коммуникаций.

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 ИСПОЛЬЗОВАННЫХ 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.
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 анг. 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

ПРИЛОЖЕНИЕ Б  
(обязательное)  
Функциональная схема алгоритма

ПРИЛОЖЕНИЕ В  
(обязательное)  
Блок схема программы

ПРИЛОЖЕНИЕ Г  
(обязательное)  
Графический интерфейс

ПРИЛОЖЕНИЕ Д  
(обязательное)  
Ведомость