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 01 061 ПЗ

Выполнил студент группы 153503
Кончик Денис Сергеевич

_____
(подпись студента)

Курсовой проект представлен
на проверку ___.____.2023

_____
(подпись студента)

Минск 2023

СОДЕРЖАНИЕ

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.

1 АНАЛИЗ ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

"CLR via C#" – книга Джозефа Рихтера, являющаяся авторитетным исчерпывающим источником информации о программировании на языке C# и платформе .NET Framework. Эта книга глубоко анализирует основы языка C# и внутреннее устройство Common Language Runtime (CLR), что позволяет программистам получить полное понимание работы данной платформы. Кроме того, автор предоставляет ценные практические советы и рекомендации по разработке приложений на платформе .NET Framework.

"Язык программирования C# 7 и платформы .NET и .NET Core" – книга, написанная Троелсеном и Джепиксом, представляет собой обширное руководство по C# и платформам .NET и .NET Core. Она охватывает широкий спектр тем, включая основы языка C#, объектно-ориентированное программирование и работу с базами данных. Книга также предоставляет подробную информацию о новых возможностях C# 7 и платформ .NET и .NET Core, что делает ее незаменимым ресурсом для программистов, стремящихся углубить свои знания и навыки в этой области.

"Совершенный код. Мастер-класс" – книга Стива Макконнелла, которая является классическим руководством по написанию качественного и эффективного программного кода. Автор в этой книге предлагает читателям принципы разработки, методы проектирования и советы по организации кода. Он подробно объясняет, как создавать программный код, который легко читается, поддерживается и расширяется, что делает эту книгу неотъемлемой для разработчиков, стремящихся к профессиональному уровню программирования.

"ASP.Net Core в действии" – книга автора Эндрю Лока, которая является незаменимым руководством по разработке веб-приложений на платформе ASP.Net Core. В этой книге подробно рассматриваются основы ASP.Net Core, архитектура MVC и работа с базами данных, что позволяет разработчикам получить необходимые знания для создания эффективных приложений. Книга также содержит множество практических примеров и иллюстраций, которые помогут читателям лучше понять концепции и развить свои навыки в этой области.

"HTML 5, CSS 3 и Web 2.0. Разработка современных Web-сайтов" – книга Владимира Дронова, которая является практическим руководством по разработке современных веб-сайтов с использованием HTML 5, CSS 3 и концепций Web 2.0. Автор в этой книге предоставляет подробное объяснение основных технологий и приемов, необходимых для создания стильных и

интерактивных веб-страниц. Книга является ценным ресурсом для веб-разработчиков, желающих овладеть современными методиками и технологиями в этой области.

Metanit – онлайн-ресурс, предоставляющий обширное собрание учебных материалов по различным технологиям программирования. На данном можно найти информацию о разработке веб-приложений, включая ASP.NET MVC, JavaScript, HTML, CSS и другие. Metanit предлагает подробные статьи, примеры кода и практические упражнения, которые помогут вам углубить свои знания и практические навыки программирования.

Документация Bootstrap – это официальный источник информации о фреймворке Bootstrap, предоставляющем набор готовых компонентов и стилей для разработки адаптивных веб-интерфейсов. На сайте Bootstrap можно найти подробное описание каждого компонента, примеры и рекомендации по их использованию. Это ценный ресурс для разработчиков, стремящихся создавать эстетически привлекательные и отзывчивые пользовательские интерфейсы.

Microsoft Learn – официальный образовательный ресурс компании Microsoft, предоставляющий богатый набор учебных материалов по различным технологиям, включая ASP.NET MVC. Страница ASP.NET MVC на Microsoft Learn содержит учебники, документацию, видеоматериалы и практические задания, помогающие освоить основы и продвинутые концепции. Этот ресурс является незаменимым для разработчиков, стремящихся углубить свои знания и получить официальную информацию от компании-разработчика.

Документация Entity Framework – это официальный источник информации о фреймворке Entity Framework, который используется для работы с базами данных в приложениях .NET. Страница документации Entity Framework предлагает подробное описание возможностей фреймворка, инструкции по его использованию и примеры кода. Этот ресурс является полезным для разработчиков, желающих изучить и использовать Entity Framework в своих проектах.

W3Schools – онлайн-учебник, предоставляющий обширную документацию и учебные материалы по веб-технологиям, включая HTML, CSS, JavaScript и другие. Сайт W3Schools предлагает простые и понятные объяснения основных концепций и примеры кода для практического применения. Он является популярным источником для получения информации о веб-разработке и помогает разработчикам освоить основные принципы и технологии веб-разработки.

2 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. Можно 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.

Рассмотрим подробнее о SQLite, которая была использована в проекте.

SQLite - это легковесная встраиваемая (embedded) реляционная база данных, которая работает без необходимости отдельного сервера баз данных. Она предоставляет набор функций для создания, управления и обработки баз данных, используя SQL (Structured Query Language) для выполнения операций.

Некоторые особенности SQLite:

1 Встраиваемая база данных: SQLite не требует установки отдельного сервера баз данных. Она представляет собой одиночный файл, который может быть интегрирован в ваше приложение. Это упрощает развертывание и управление базой данных.

2 Кроссплатформенность: SQLite поддерживает большое количество операционных систем, включая Windows, macOS, Linux и множество других платформ. Это позволяет вам разрабатывать приложения, которые могут работать на различных операционных системах.

3 Быстрота и эффективность: SQLite обладает небольшим размером, минимальными накладными расходами и высокой производительностью. Она

быстро выполняет запросы к базе данных и обрабатывает большой объем данных.

4 Поддержка SQL: SQLite полностью поддерживает язык SQL, что позволяет использовать привычный синтаксис для создания таблиц, выполнения запросов, фильтрации данных и многого другого. Она поддерживает множество типов данных и операторов SQL.

5 Транзакционная безопасность: SQLite обеспечивает транзакционную безопасность при работе с данными. Она поддерживает ACID-свойства (атомарность, согласованность, изолированность, долговечность), что обеспечивает надежность и целостность данных.

6 Широкая поддержка: SQLite имеет широкую поддержку и активное сообщество разработчиков. Существует множество библиотек, инструментов и документации для работы с SQLite на различных платформах и в различных языках программирования.

В данном проекте использование SQLite как базы данных позволило легко интегрировать базу данных в ваше приложение без дополнительной настройки сервера. SQLite, сочетая простоту, эффективность и мощь SQL, является отличным выбором для малых и средних проектов, где требуется надежное хранение и обработка данных.

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.

3 ПАТТЕРНЫ ПРОЕКТИРОВАНИЯ, ИСПОЛЬЗУЕМЫЕ В РАЗРАБОТКЕ ПРИЛОЖЕНИЯ

В разработке вашего интернет-магазина, реализованного на платформе ASP.NET MVC, были применены несколько паттернов проектирования, которые способствуют эффективному организации кода, повышению его переиспользуемости и улучшению общей архитектуры приложения. Была использована многоуровневая архитектура, разделяющая приложение на уровни доступа к данным (DAL), бизнес-логику (BLL) и пользовательский интерфейс (WEB). Кроме того, был применен паттерн Unit of Work для управления транзакциями и паттерном Repository для абстракции доступа к данным. Однако наиболее фундаментальным и широко применяемым паттерном, используемым в вашем приложении, является MVC (Model-View-Controller), который обеспечивает разделение ответственностей между моделью данных, представлением пользовательского интерфейса и контроллером, и способствует более гибкому и масштабируемому развитию приложения.

Паттерн MVC (Model-View-Controller) является одним из наиболее широко используемых паттернов проектирования в веб-разработке, включая ASP.NET MVC. Он разделяет приложение на три основных компонента: Model (Модель), View (Представление) и Controller (Контроллер). Каждый из этих компонентов имеет свою отдельную ответственность, что способствует улучшению поддерживаемости, расширяемости и переиспользуемости кода.

Подробное описание каждого компонента паттерна MVC:

1 Модель (Model): Модель представляет собой представление данных, которые используются в приложении. Это может быть класс, структура или набор классов, представляющих бизнес-логику и данные приложения. Модель содержит методы для доступа, изменения и обработки данных, а также взаимодействует с базой данных (DAL) для получения и сохранения данных. Она не зависит от представления или контроллера, что обеспечивает ее независимость и повторное использование в различных частях приложения.

2 Представление (View): Представление отвечает за отображение данных пользователю. Оно представляет собой пользовательский интерфейс и может быть HTML-страницей, шаблоном или контроллером, которые отображают данные из модели. Представление не содержит бизнес-логики и должно быть максимально независимым от модели и контроллера. Оно получает данные из модели и отображает их для пользователя. Представление также может обрабатывать ввод и отправлять его контроллеру.

3 Контроллер (Controller): Контроллер является посредником между представлением и моделью. Он получает пользовательский ввод из представления и принимает решения о том, какая модель должна использоваться и какие действия должны быть выполнены. Контроллер обрабатывает запросы пользователя, взаимодействует с моделью для получения данных и обновления состояния модели, а затем выбирает соответствующее представление для отображения этих данных. Контроллер также отвечает за обработку логики взаимодействия между моделью и представлением.

Преимущества паттерна MVC в ASP.NET MVC:

1 Разделение ответственности: Паттерн MVC позволяет четко разделить бизнес-логику, пользовательский интерфейс и взаимодействие с данными. Это упрощает поддержку кода и улучшает его переиспользуемость.

2 Гибкость и масштабируемость: Отдельные компоненты паттерна MVC могут быть изменены или заменены независимо друг от друга, что позволяет легко расширять функциональность приложения.

3 Легкость тестирования: Благодаря разделению ответственности, каждый компонент паттерна MVC может быть протестирован отдельно, что упрощает создание модульных тестов и повышает надежность приложения.

Недостатки паттерна MVC в ASP.NET MVC:

1 Усложнение архитектуры: В некоторых случаях использование паттерна MVC может привести к более сложной архитектуре приложения, особенно при работе с более крупными проектами.

2 Накладные расходы: В связи с разделением ответственности и взаимодействием между компонентами, может возникнуть некоторое увеличение накладных расходов и сложности разработки.

3 Необходимость обучения: Разработчикам, не знакомым с паттерном MVC, может потребоваться некоторое время для изучения и понимания его концепций и принципов.

В целом, паттерн MVC является мощным инструментом для организации кода в ASP.NET MVC приложениях. Он обеспечивает четкую структуру, легкость тестирования и переиспользуемость кода, хотя может потребоваться некоторое время и усилия для его освоения и эффективного применения.

Паттерн Unit of Work представляет собой механизм, который обеспечивает управление транзакциями и группировку операций базы данных в рамках одной логической единицы работы. Он позволяет выполнить несколько операций базы данных, таких как добавление, обновление и

удаление данных, в пределах одной транзакции.

Основные особенности и преимущества паттерна Unit of Work:

1 Управление транзакциями: Паттерн Unit of Work позволяет управлять транзакциями в приложении, обеспечивая атомарность операций. Если какая-либо операция не выполнится успешно, все изменения могут быть отменены (откат транзакции).

2 Улучшение производительности: Группировка операций базы данных в рамках одной транзакции может улучшить производительность, так как минимизируется количество обращений к базе данных.

3 Отложенное сохранение изменений: С помощью паттерна Unit of Work можно отложить сохранение изменений в базу данных до определенного момента, что может быть полезно в некоторых сценариях, таких как массовое обновление данных.

4 Управление состоянием объектов: Паттерн Unit of Work может отслеживать состояние объектов и автоматически применять изменения при сохранении.

Паттерн Repository предоставляет абстракцию для доступа к данным и скрывает детали конкретного источника данных (например, базы данных) от остальной части приложения. Репозиторий обеспечивает единый интерфейс для работы с данными, скрывая сложности доступа к данным.

Основные особенности и преимущества паттерна Repository:

1 Разделение доступа к данным: Репозиторий отделяет бизнес-логику от деталей доступа к данным, что упрощает сопровождение кода и повышает его переиспользуемость.

2 Единый интерфейс: Паттерн Repository предоставляет единый интерфейс для работы с данными, что упрощает взаимодействие между компонентами приложения.

3 Легкость тестирования: Благодаря абстракции, предоставляемой репозиторием, легче проводить модульное тестирование кода, так как можно заменить реальную реализацию репозитория на мок-объекты или заглушки.

Комбинирование паттерна Unit of Work с паттерном Repository и использование Entity Framework позволяет эффективно управлять доступом к данным, обеспечивает гибкость и легкость разработки, а также улучшает сопровождаемость и масштабируемость приложения на платформе ASP.NET MVC.

Также в проекте было использовано внедрение зависимостей. Внедрение зависимостей (Dependency Injection, DI) является ключевым паттерном, который используется для объединения компонентов в проекте и

обеспечения слабой связности между ними. Этот паттерн позволяет инвертировать контроль над созданием и управлением зависимостями, делая систему более гибкой и легко тестируемой.

Основные понятия и принципы внедрения зависимостей:

1 Зависимость: Зависимость - это объект, от которого зависит другой объект для своего функционирования. Например, класс контроллера может зависеть от сервиса для выполнения операций над данными.

2 Инверсия контроля: Инверсия контроля означает, что создание и управление зависимостями осуществляются не самим объектом, который их использует, а внешним компонентом (DI-контейнером).

3 DI-контейнер: DI-контейнер является центральным компонентом, который отвечает за создание и внедрение зависимостей в систему. Он содержит конфигурацию, которая определяет, какие зависимости должны быть созданы и как они должны быть внедрены.

Преимущества внедрения зависимостей:

1 Уменьшение связности: Внедрение зависимостей позволяет снизить связность между компонентами, так как объекты получают свои зависимости извне. Это делает код более гибким, модульным и легко тестируемым.

2 Упрощение тестирования: Благодаря DI можно легко создавать и внедрять заглушки (mock objects) или поддельные реализации зависимостей во время тестирования. Это упрощает юнит-тестирование компонентов и повышает их надежность.

Как работает внедрение зависимостей:

1 Конфигурация DI-контейнера: Сначала происходит определение конфигурации DI-контейнера, в которой указывается, какие зависимости и как они должны быть созданы и внедрены. Это может быть выполнено с использованием кода или конфигурационных файлов.

2 Внедрение зависимостей: Когда компонент требует определенную зависимость, DI-контейнер создает и внедряет ее в объект-получатель. Внедрение может осуществляться через конструктор, методы или свойства.

3 Использование зависимостей: Объекты, получившие свои зависимости, могут использовать их для выполнения своей работы. Зависимости становятся доступными для использования без явного создания или управления ими внутри объекта.

Внедрение зависимостей помогает разделить ответственность и создает более гибкую архитектуру приложения.

4 ФУНКЦИОНАЛЬНЫЕ ВОЗМОЖНОСТИ ПРОГРАММЫ

Функциональные возможности программы в зависимости от роли пользователя представлены в диаграмме вариантов использования (USE-CASE диаграмма) (приложение В).

Неавторизованный пользователь попадает на главную страницу (рисунок 1). Далее у него есть возможность искать товары, а также авторизоваться либо зарегистрироваться.

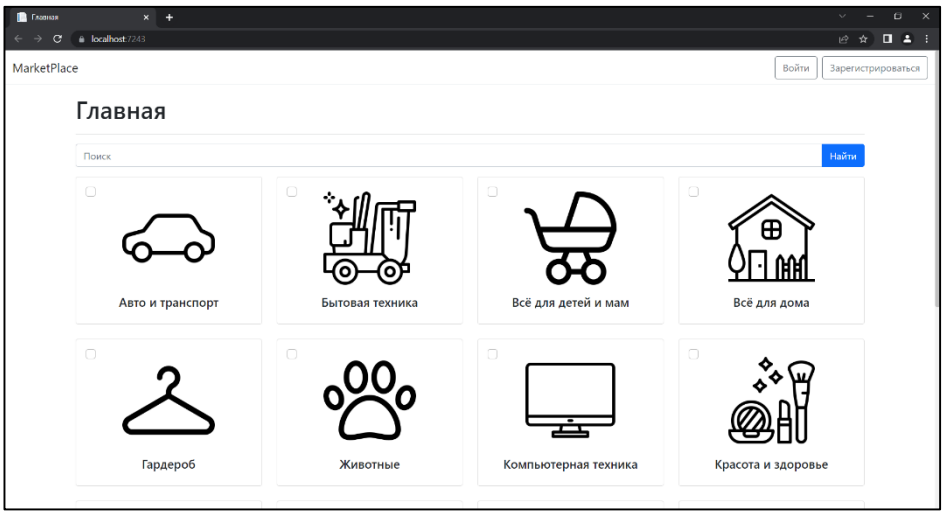


Рисунок 1 – Главная страница

Чтобы получить доступ к функционалу покупателя, необходимо войти (рисунок 3) или зарегистрироваться (рисунок 2).

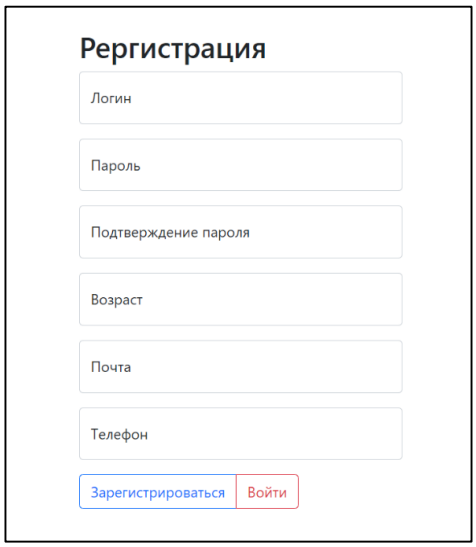


Рисунок 2 – Страница регистрации

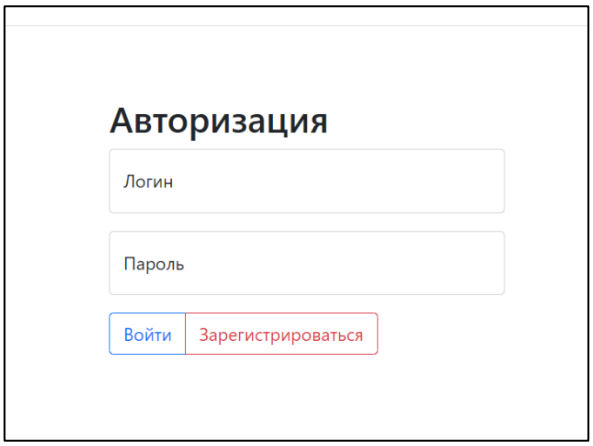


Рисунок 3 – Страница авторизации

Далее покупатель с главной страницы (рисунок 2) может производить поиск товаров и сможет увидеть список найденных товаров (рисунок 4) исходя из своего запроса (строка запроса и выбранные категории).

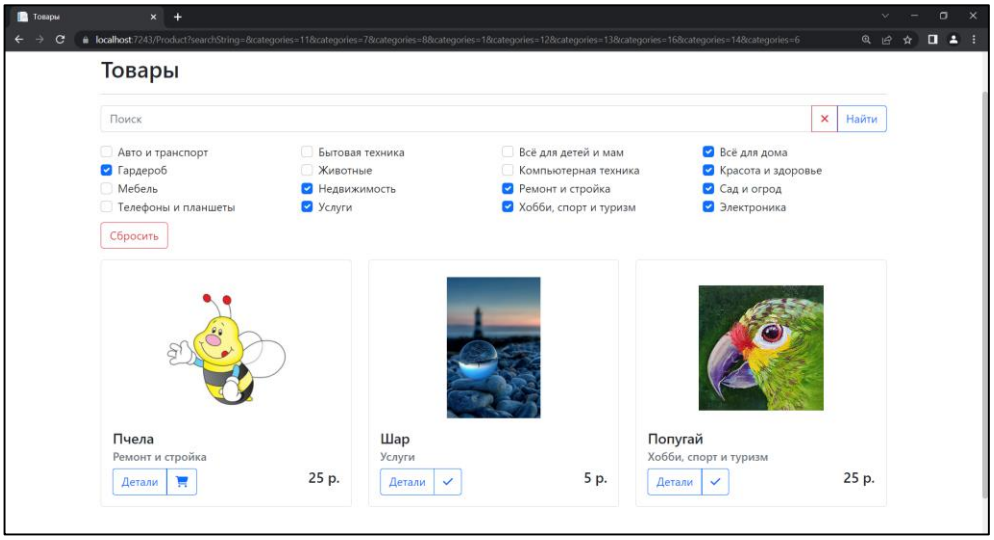


Рисунок 4 – Найденные товары

Затем покупатель может перейти к деталям товара и увидеть более подробную информацию о товаре (рисунок 5), нажав на кнопку «Детали» либо добавить товар в корзину (кнопка «Корзина»).

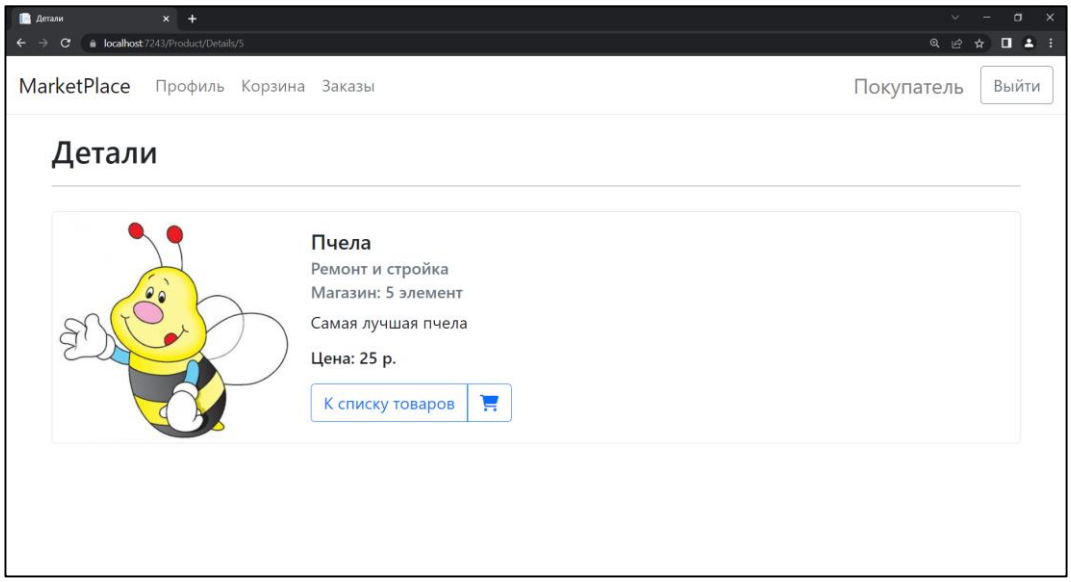


Рисунок 5 – Детали товара

По нажатию на клавишу «Корзина» товар попадает в корзину покупателя (рисунок 6).

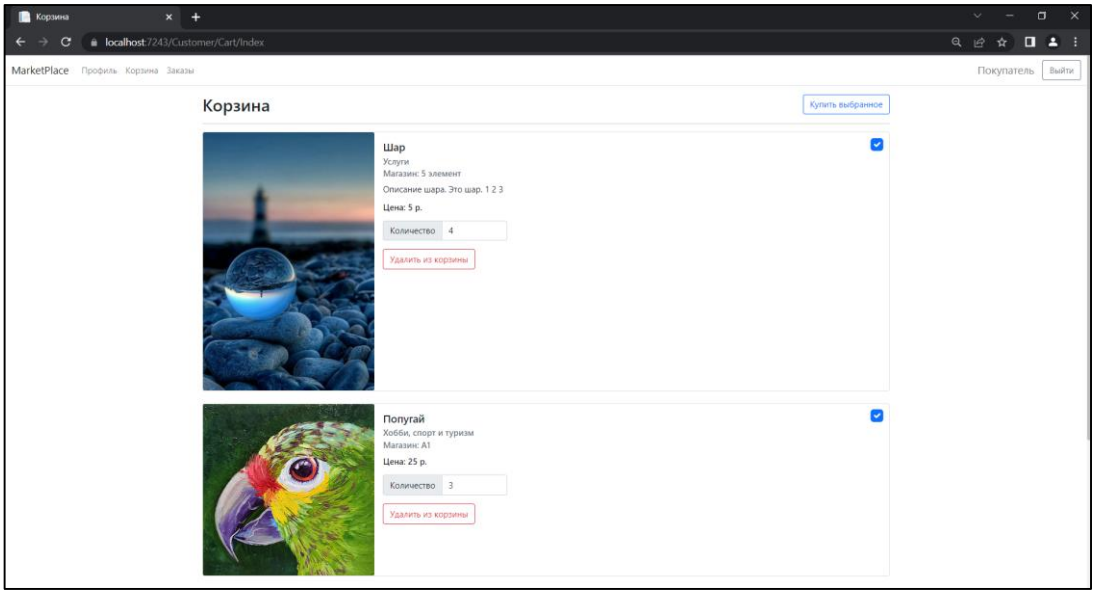


Рисунок 6 – Корзина покупателя

Из корзины можно удалить товар, можно выбрать нужное количество и галочкой выбрать товар к покупке. Если выбран хоть один элемент корзины, то появляется кнопка «Купить выбранное». По нажатию на эту кнопку происходит переадресация на страницу просмотра (подтверждения) заказа (рисунок 7).

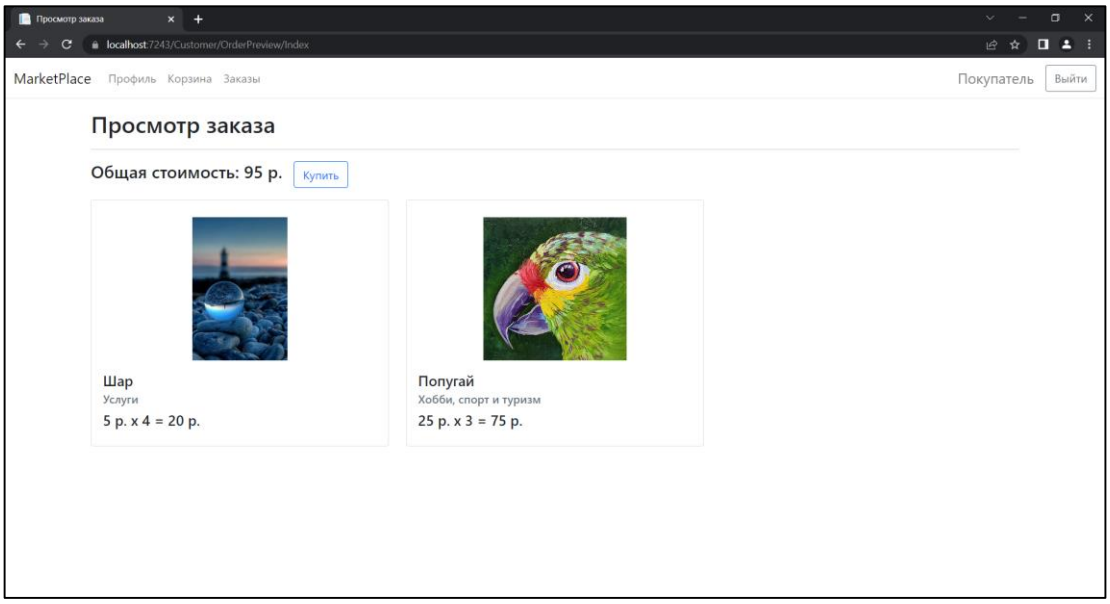


Рисунок 7 – Просмотр заказа

Если покупатель согласен с сформированным заказом, то он нажимает на кнопку «Купить» и происходит покупка. Далее этот заказ можно увидеть в списке заказов покупателя (рисунок 8).

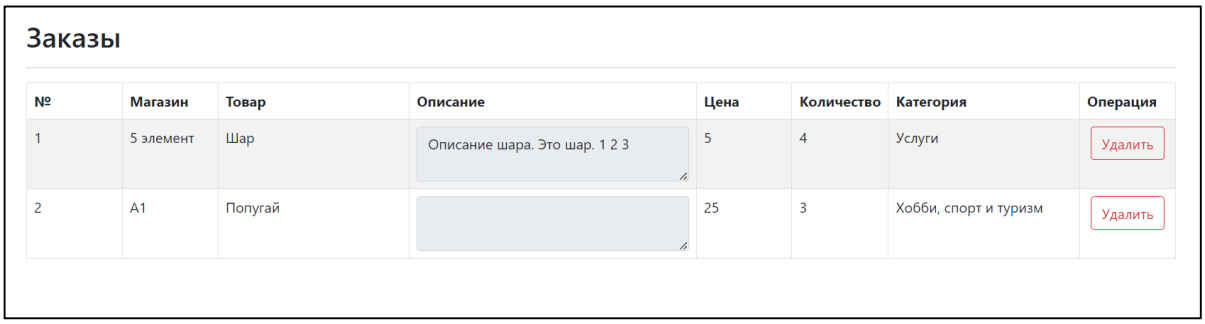


Рисунок 8 – Заказы покупателя

Также у покупателя есть возможность очистить каждый заказ из истории заказов нажатием кнопки «Удалить».

Если пользователь зашел с ролью продавца, то ему доступен функционал по изменению магазина, к которому он относится (рисунок 9).

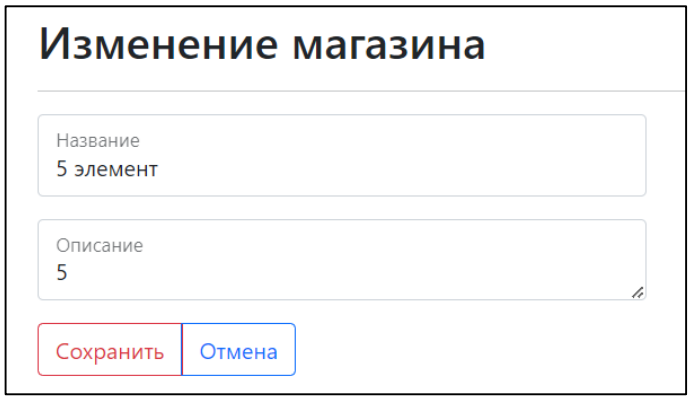


Рисунок 9 – Изменение магазина

Также продавец видит список всех товаров (рисунок 10) магазина, к которому он относится. Он может добавить (рисунок 11), удалить или изменить любой товар.

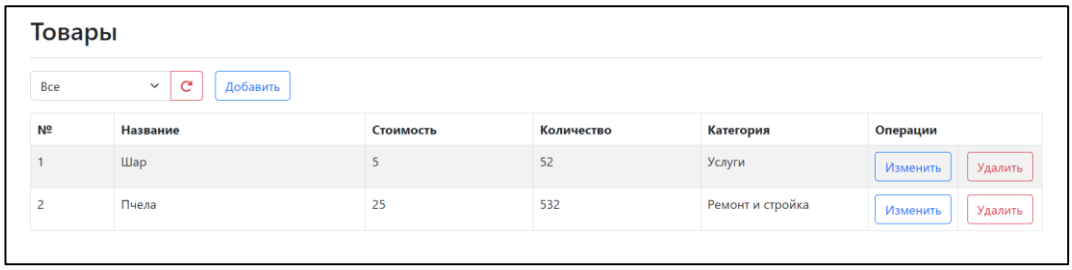


Рисунок 10 – Список товаров магазина

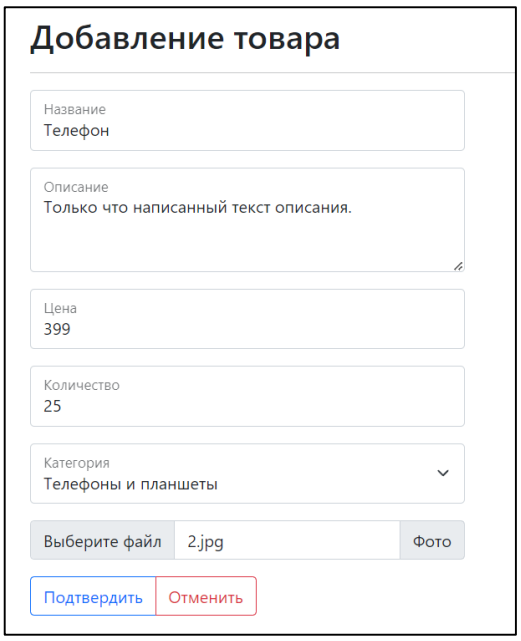


Рисунок 11 – Добавление товара

Если пользователь зашел с ролью администратора, то ему доступен весь функционал по управлению каталогом товаров и пользователями (покупателями и продавцами). Администратор имеет возможность просматривать (рисунок 12), добавлять (рисунок 13) и удалять всех продавцов.

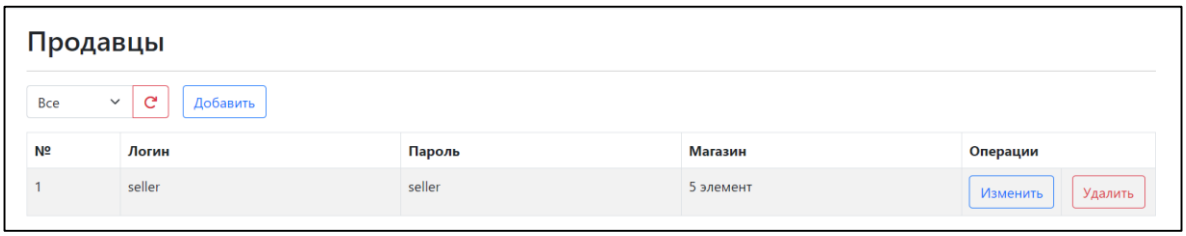


Рисунок 12 – Список продавцов

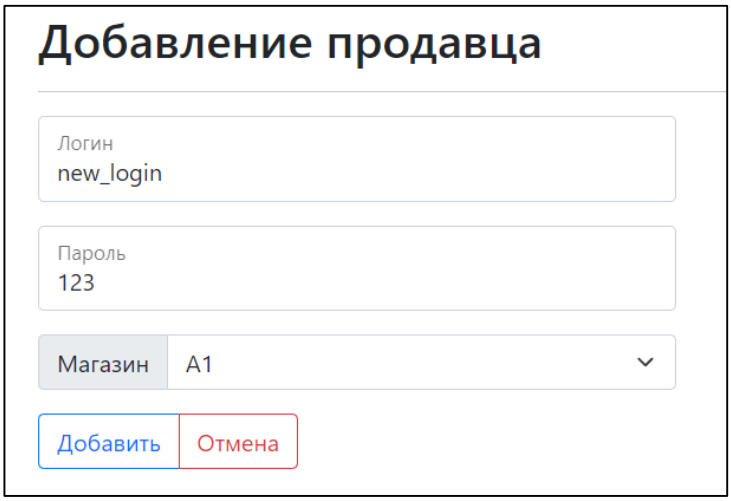


Рисунок 13 – Добавление продавца

Также администратор может просматривать все магазины (рисунок 14), добавлять их и изменять.

Рисунок 14 – Список магазинов

5 АРХИТЕКТУРА РАЗРАБАТЫВАЕМОЙ ПРОГРАММЫ

Архитектура разрабатываемой программы основана на многоуровневой архитектуре. Многоуровневая архитектура (также называемая n-уровневой архитектурой) - это структурная модель разработки ПО, которая разделяет систему на несколько логических уровней. Каждый уровень имеет определенную функциональность и общение с другими уровнями только через определенный интерфейс. Эта архитектура упрощает разработку, тестирование и сопровождение ПО.

В многоуровневой архитектуре веб-приложения, как в случае данного проекта, обычно используются три слоя:

1 Data Access Layer (DAL): DAL отвечает за доступ к данным и работу с источником данных, например, базой данных. В нем находятся классы, которые взаимодействуют с базой данных, выполняют запросы, чтение, запись и обновление данных. DAL использует подход ORM (Object-Relational Mapping) для отображения объектов приложения на таблицы базы данных. В в этом слое используется ORM-фреймворк, например, Entity Framework, для упрощения работы с базой данных. DAL также содержит репозитории, которые предоставляют абстракцию над доступом к данным, скрывая детали взаимодействия с базой данных.

2 Business Logic Layer (BLL): BLL содержит бизнес-логику приложения, которая определяет правила и операции, связанные с предметной областью приложения. В этом слое находятся классы, которые обрабатывают и манипулируют данными, выполняют вычисления, проверки на валидность и другие операции, специфичные для бизнес-правил. BLL использует данные, полученные из DAL, и применяет на них бизнес-правила. Он содержит сервисы, которые обеспечивают интерфейс для взаимодействия с DAL и предоставляют методы для выполнения операций с данными.

3 Presentation Layer (WEB): WEB представляет пользовательский интерфейс (UI) приложения и обрабатывает ввод и вывод данных пользователю. Он включает в себя компоненты, такие как контроллеры и представления (views). Presentation Layer принимает запросы от пользователей, обрабатывает их с помощью контроллеров, которые взаимодействуют с BLL, чтобы получить нужные данные или выполнить операции. Затем контроллеры передают данные представлениям для отображения пользователю.

ЗАКЛЮЧЕНИЕ

В рамках данной работы был разработан каталог товаров с использованием ASP.NET MVC. Проект был построен на основе нескольких ключевых технологий, которые были детально рассмотрены в предыдущих главах.

MVC (Model-View-Controller) предоставил эффективный шаблон проектирования, разделяющий данные, представление и логику обработки приложения. Это позволило легко управлять функциональностью и отображением страниц.

С использованием подхода Services мы смогли разделить бизнес-логику на отдельные слои и создать независимые компоненты для обработки различных задач. Это способствовало повторному использованию кода и обеспечивало более гибкую архитектуру.

Unit of Work и Repository позволили эффективно управлять доступом к данным и абстрагироваться от деталей взаимодействия с базой данных. Это упростило взаимодействие с Entity Framework и сделало работу с данными более удобной и понятной.

Использование Dependency Injection дало возможность создать слабосвязанные компоненты и обеспечить более гибкую конфигурацию приложения.

Применение многоуровневой архитектуры (DAL, BLL, WEB) помогло разделить наше приложение на слои с различными функциональными задачами. Это способствовало поддерживаемости, масштабируемости и облегчило сопровождение приложения.

ASP.NET MVC и Razor предоставили мощный фреймворк для создания динамических и отзывчивых пользовательских интерфейсов. С помощью JavaScript и HTML получилось обеспечить интерактивность и визуально привлекательный внешний вид каталога товаров.

В целом, разработка данного каталога товаров на основе ASP.NET MVC и применение вышеперечисленных технологий позволили создать функциональное, эффективное и удобное в использовании приложение. Этот проект является отличным примером применения современных технологий в разработке интернет-магазинов и демонстрирует преимущества использования MVC-архитектуры, шаблонов проектирования и инструментов разработки веб-приложений.

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. HTML 5, CSS 3 и Web 2.0. Разработка современных Web-сайтов. — СПб. : БХВ-Петербург, 2011. — 416 с.: ил. — (Профессиональное программирование)

[6] Metanit [Электронный ресурс]. – Режим доступа : https://metanit.com/.

[7] Bootstrap – документация [Электронный ресурс]. – Режим доступа : https://getbootstrap.com/.

[8] ASP.NET MVC - Microsoft Learn [Электронный ресурс]. – Режим доступа : https://learn.microsoft.com/en-us/aspnet/mvc/.

[9] Entity Framework documentation [Электронный ресурс]. – Режим доступа : https://learn.microsoft.com/en-us/ef//.

[10] W3Schools [Электронный ресурс]. – Режим доступа : https://www.w3schools.com/.

ПРИЛОЖЕНИЕ А
(обязательное)
Исходный код программы

Листинг 1 – Контроллеры области администратора

```
[Area("Admin")]
[Authorize(Roles = "Admin")]
public class CustomerController : Controller
{
    private ICustomerService _customerService;
    public CustomerController(ICustomerService customerService;

    [HttpGet]
    public async Task<IActionResult> Index();

    [HttpGet]
    public async Task<IActionResult> Save(int id);

    [HttpPost]
    [ValidateAntiForgeryToken]
    public async Task<IActionResult> Save([Bind("Id,Login,Password,Profile")]
DAL.Entities.Customer item);

    [HttpPost]
    [ValidateAntiForgeryToken]
    public async Task<IActionResult> Delete(int id);
}

[Area("Admin")]
[Authorize(Roles = "Admin")]
public class ProductController : Controller
{
    private readonly IProductService _productService;
    private readonly IShopService _shopService;

    public ProductController(IProductService productService, IShopService
shopService);

    public async Task<IActionResult> Index(int shopId, ProductCategory
category, SortOrder order);

    [HttpGet]
    public async Task<IActionResult> Save(int id, int shopId);

    [HttpPost]
    [ValidateAntiForgeryToken]
    public async Task<IActionResult>
Save([Bind("Id,Name,Description,Price,Quantity,Category,ShopId")] Product
item, IFormFile? photo);

    [HttpPost]
    [ValidateAntiForgeryToken]
    public async Task<IActionResult> Delete(int id, int shopId);
```

```
[Area("Admin")]
[Authorize(Roles = "Admin")]
public class SellerController : Controller
{
    private ISellerService _sellerService;
    private IShopService _shopService;
    public SellerController(ISellerService sellerService, IShopService
shopService);

    public async Task<IActionResult> Index(int shopId);

    [HttpGet]
    public async Task<IActionResult> Save(int id);

    [HttpPost]
    [ValidateAntiForgeryToken]
    public async Task<IActionResult> Save([Bind("Id,Login,Password,ShopId")]
DAL.Entities.Seller item);

    [HttpPost]
    [ValidateAntiForgeryToken]
    public async Task<IActionResult> Delete(int id);
}

[Area("Admin")]
[Authorize(Roles = "Admin")]
public class ShopController : Controller
{
    private IShopService _shopService;
    public ShopController(IShopService shopService);

    public async Task<IActionResult> Index(string? name);

    [HttpGet]
    public async Task<IActionResult> Details(int id);

    [HttpGet]
    public async Task<IActionResult> Save(int id);

    [HttpPost]
    [ValidateAntiForgeryToken]
    public async Task<IActionResult> Save([Bind("Id,Name,Description")] Shop
item);


    [HttpGet]
    public async Task<IActionResult> Delete(int id);
}
```

Листинг 2 – Контроллеры области покупателя

```
[Area("Customer")]
[Authorize(Roles = "Customer")]
public class CartController : Controller
{
    private readonly ICartService _cartService;
    public CartController(ICartService cartService);
    public async Task<IActionResult> Index();
    public async Task<IActionResult> Add(int productId);
    public async Task<IActionResult> Remove(int productId);

}
```

```
[Area("Customer")]
[Authorize(Roles = "Customer")]
public class CustomerController : Controller
{
    private readonly ICustomerService _customerService;
    private readonly IAccountService _accountService;

    public CustomerController(ICustomerService customerService,
IAccountService accountService);


    [HttpGet]
    public async Task<IActionResult> Edit();

    [HttpPost]
    public async Task<IActionResult> Edit(DAL.Entities.Customer item);
}

[Area("Customer")]
[Authorize(Roles = "Customer")]
public class OrderController : Controller
{
    private readonly IOrderService _orderService;

    public OrderController(IOrderService orderService);

    public async Task<IActionResult> Index();

    [HttpPost]
    public async Task<IActionResult> Delete(int id);
}

[Area("Customer")]
[Authorize(Roles = "Customer")]
public class OrderPreviewController : Controller
{
    private readonly ICartService _cartService;

    public OrderPreviewController(ICartService cartService);

    public async Task<IActionResult> Index(IEnumerable<int> cartItemsIds);
}

[Area("Customer")]
[Authorize(Roles = "Customer")]
public class PaymentController : Controller
{
    private readonly ICartService _cartService;
    private readonly IOrderService _orderService;

    public PaymentController(ICartService cartService, IOrderService
orderService);

    public async Task<IActionResult> Buy(IEnumerable<int> cartItemsIds);

    public IActionResult Thanks();
}
```

Листинг 3 – Контроллеры области покупателя

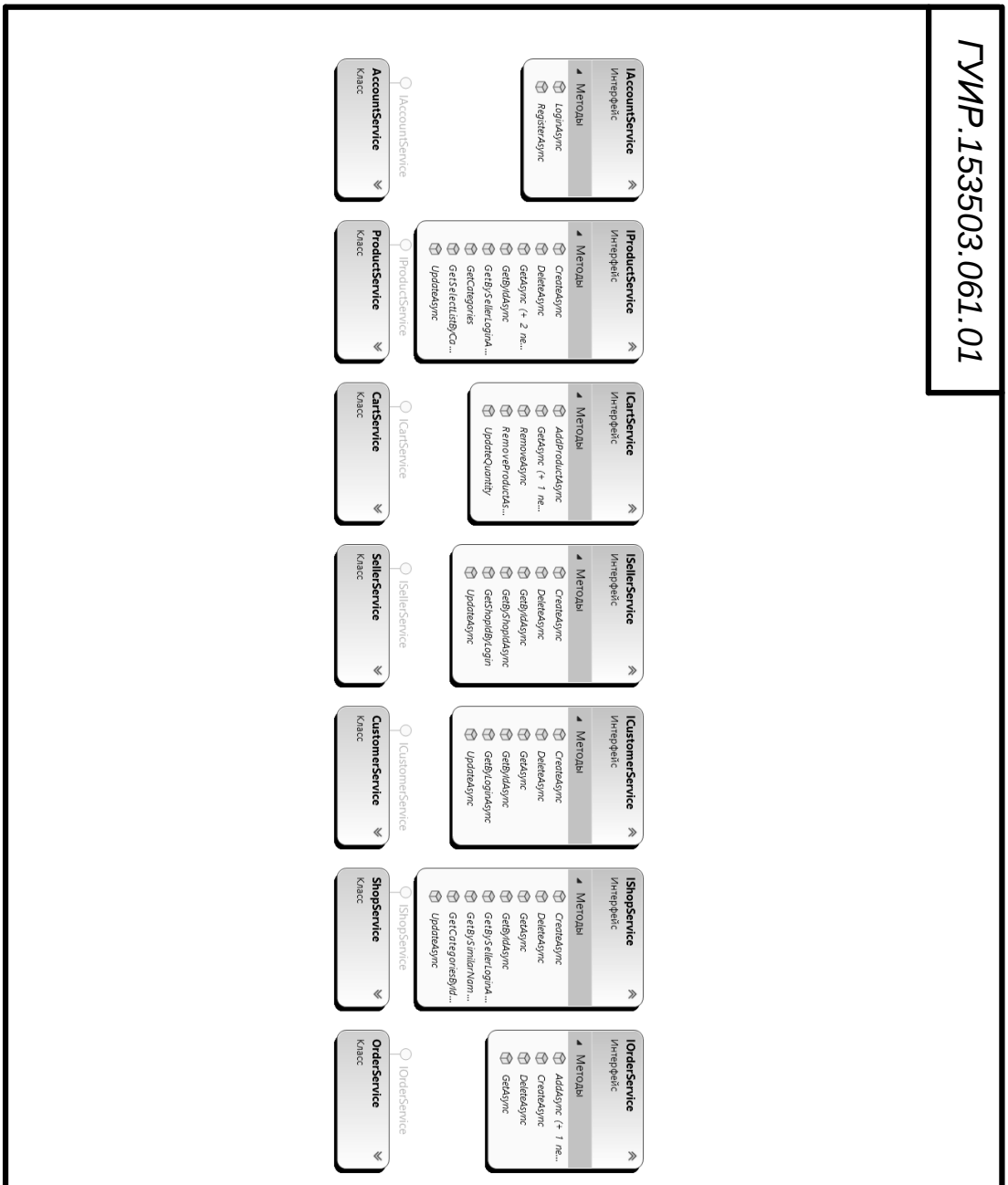
```
[Area("Seller")]
[Authorize(Roles = "Seller")]
public class ProductController : Controller
{
    private readonly IProductService _productService;
    private readonly ISellerService _sellerService;
    private readonly IShopService _shopService;

    public ProductController(IProductService productService,
        ISellerService sellerService, IShopService shopService);

    public async Task<IActionResult> Index(ProductCategory category,
SortOrder order);

    [HttpGet]
    public async Task<IActionResult> Save(int id);

    [HttpPost]
    public async Task<IActionResult>
Save([Bind("Id,Name,Description,Price,Quantity,Category")] Product item,
IFormFile? photo);

    [HttpPost]
    public async Task<IActionResult> Delete(int id);
}

[Area("Seller")]
[Authorize(Roles = "Seller")]
public class ShopController : Controller
{
    IShopService _shopService;

    public ShopController(IShopService shopService);

    [HttpGet]
    public async Task<IActionResult> Index();

    [HttpGet]
    public async Task<IActionResult> Edit();

    [HttpPost]
    public async Task<IActionResult> Edit([Bind("Id,Name,Description")] Shop
item);
}
```

Листинг 4 – Контроллеры, не относящиеся к областям

```
public class AccountController : Controller
{
    private readonly IAccountService _accountService;

    public AccountController(IAccountService accountService);

    [HttpGet]
    public IActionResult AccessDenied();

    [HttpGet]
    public IActionResult UnauthorizedModal();

    public async Task<IActionResult> Logout();
```

```
    [HttpGet]
    public IActionResult Login();

    [HttpPost]
    public async Task<IActionResult> Login(LoginDTO vm);

    [HttpGet]
    public IActionResult Register();

    [HttpPost]
    public async Task<IActionResult> Register(RegisterDTO dto);
}

public class HomeController : Controller
{
    private readonly IProductService _productService;

    public HomeController(IProductService productService);

    public IActionResult Index();
}

public class ProductController : Controller
{
    private readonly IProductService _productService;
    private readonly ICartService _cartService;

    public ProductController(IProductService productService, ICartService
cartService)
    {
        _productService = productService;
        _cartService = cartService;
    }

    [ResponseCache(Location = ResponseCacheLocation.None, NoStore = true)]
    public async Task<IActionResult> Index(IEnumerable<ProductCategory>
categories, string searchString)
    {
        var productResponse = await _productService.GetAsync(categories,
searchString);
        if (productResponse.StatusCode == HttpStatusCode.OK)
        {
            return View(new ProductListViewModel()
            {
                Products = productResponse.Data!.OrderBy(p =>
p.Category.GetDisplayName()).ThenBy(p => p.Name),
                SelectedCategories =
_productService.GetSelectListByCategories(categories).Data!,
                SearchString = searchString
            });
        }
        return View("Error", new
ErrorViewModel(productResponse.Deconstruct()));
    }

    public async Task<IActionResult> Details(int id);
}
```

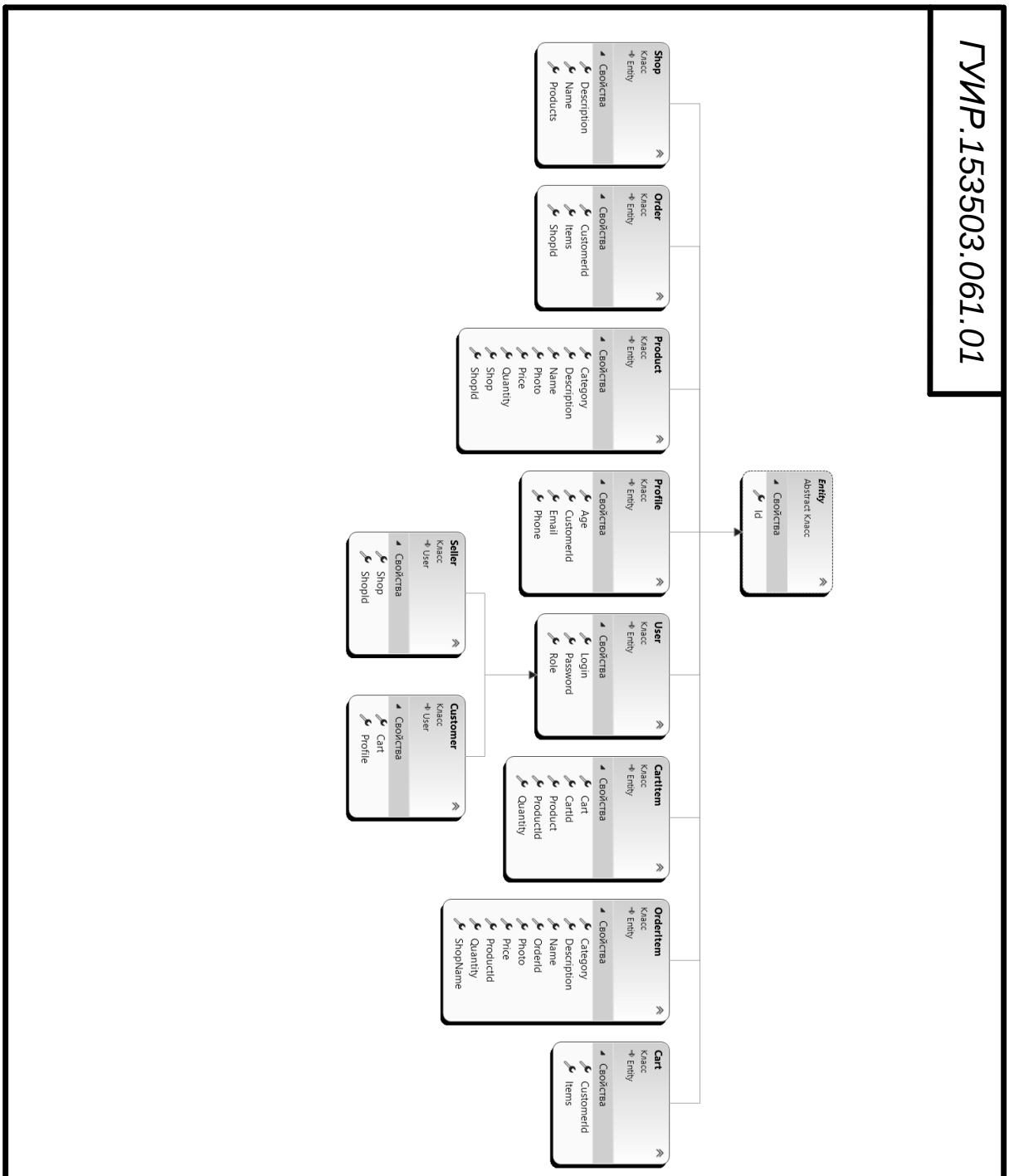
ПРИЛОЖЕНИЕ Б
(обязательное)
Диаграмма классов



Изм.	Лист	№ докум.	Подп.	Дата	ГУИР.153503.061.01		
Разраб.		Кончик			Диаграмма классов	Лит. У	Масса
Пров.		Владымцев					Масштаб 1:1
						Лист 1	Листов 5
						Кафедра информатики группа 153503	

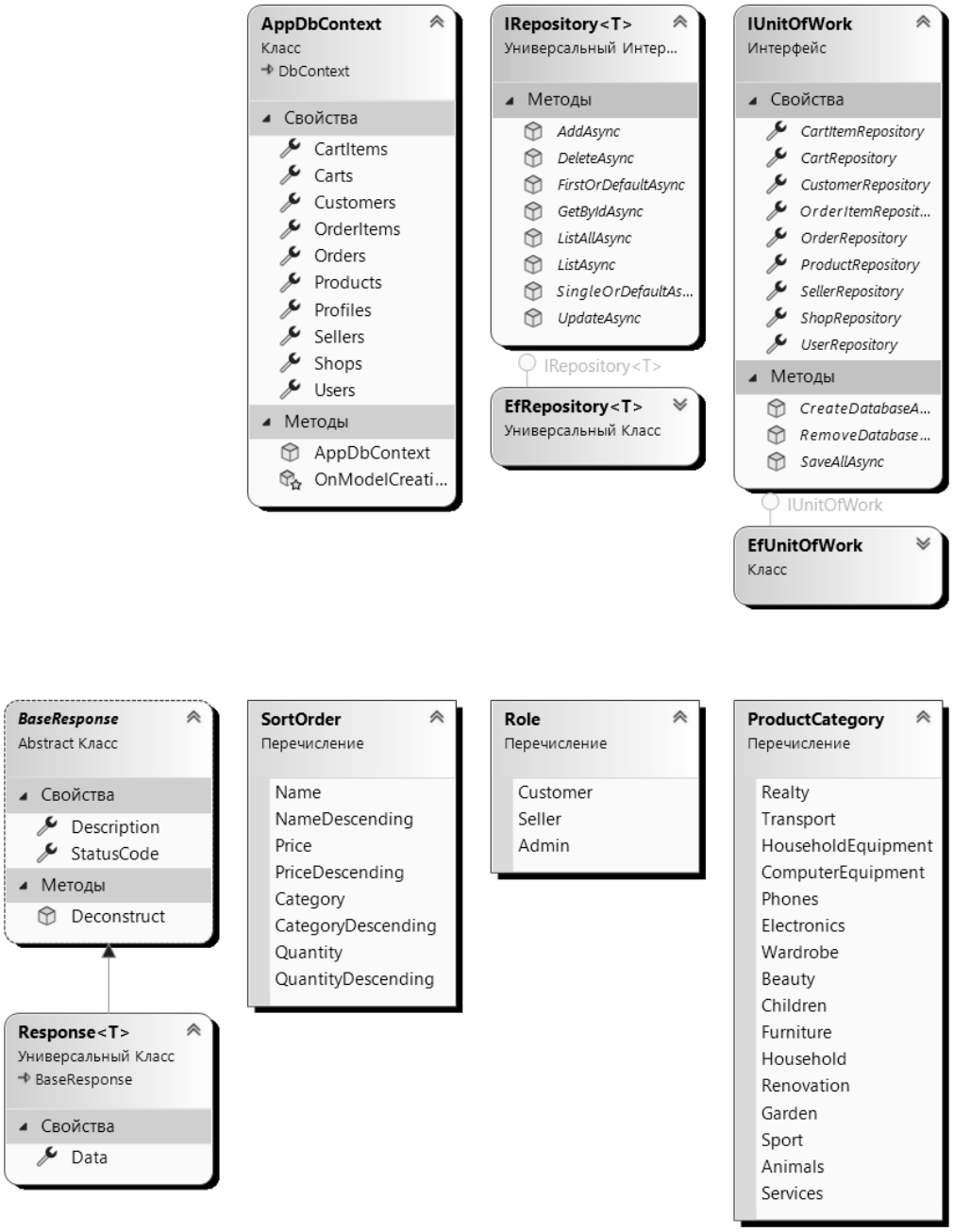
Формат А4

ГУИР.153503.061.01



					ГУИР.153503.061.01		
Изм.	Лист	№ докум.	Подп.	Дата			
Разраб.		Кончик			Диаграмма классов	Лит.	Масса / Масштаб
Пров.		Владымцев				У	1:1
						Лист 2	Листов 5
						Кафедра информатики группа 153503	

Формат А4

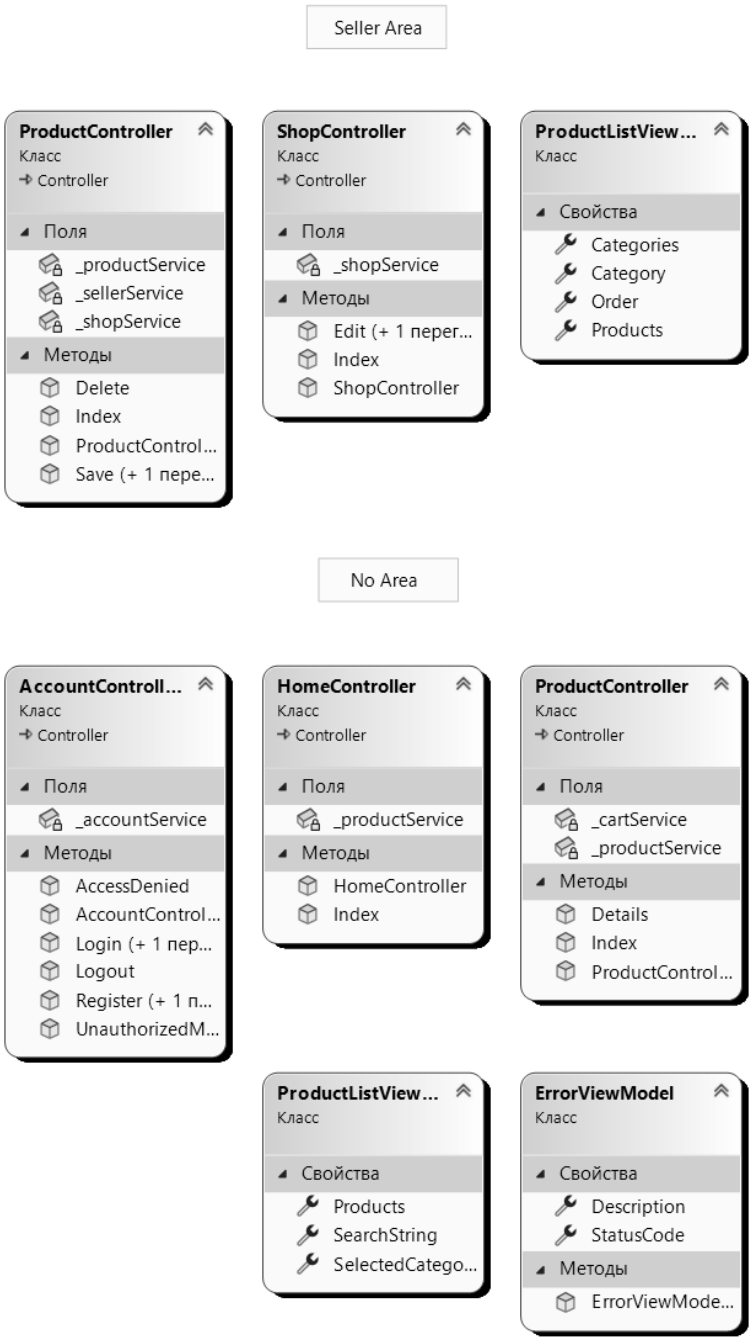


					ГУИР.153503.061.01	
Изм.	Лист	№ докум.	Подп.	Дата		
Разраб.		Кончик			Диаграмма классов	Лит. У / Масса / Масштаб 1:1
Пров.		Владымцев				Лист 3 / Листов 5
						Кафедра информатики группа 153503

Формат А4

					ГУИР.153503.061.01	
Изм.	Лист	№ докум.	Подп.	Дата		
Разраб.		Кончик			Диаграмма классов	Лит. У / Масса / Масштаб 1:1
Пров.		Владымцев				Лист 4 / Листов 5
						Кафедра информатики группа 153503

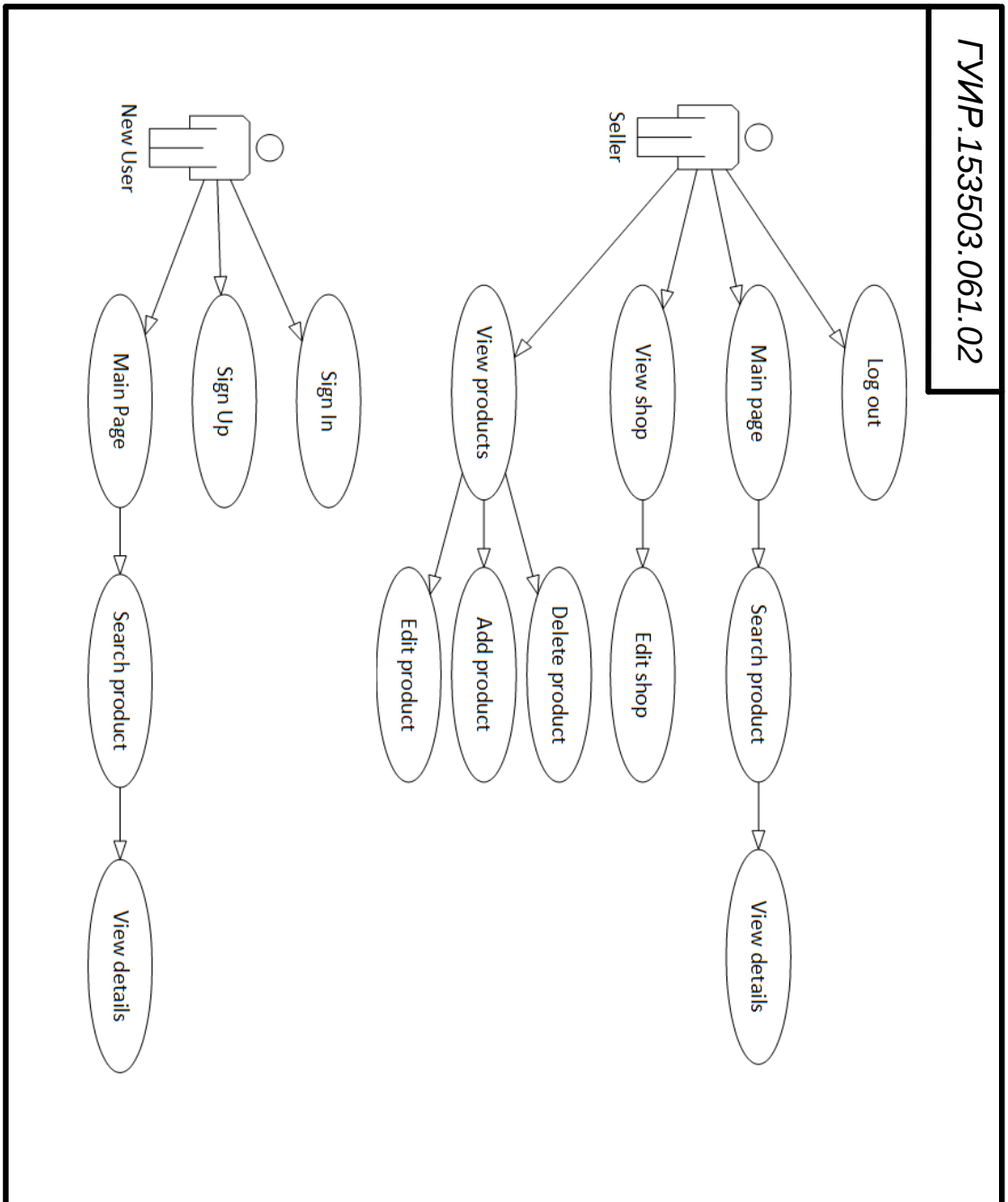
Формат А4



					ГУИР.153503.061.01	
Изм.	Лист	№ докум.	Подп.	Дата		
Разраб.		Кончик			Диаграмма классов	Лит. У / Масса / Масштаб 1:1
Пров.		Владымцев				Лист 5 / Листов 5
						Кафедра информатики группа 153503

Формат А4

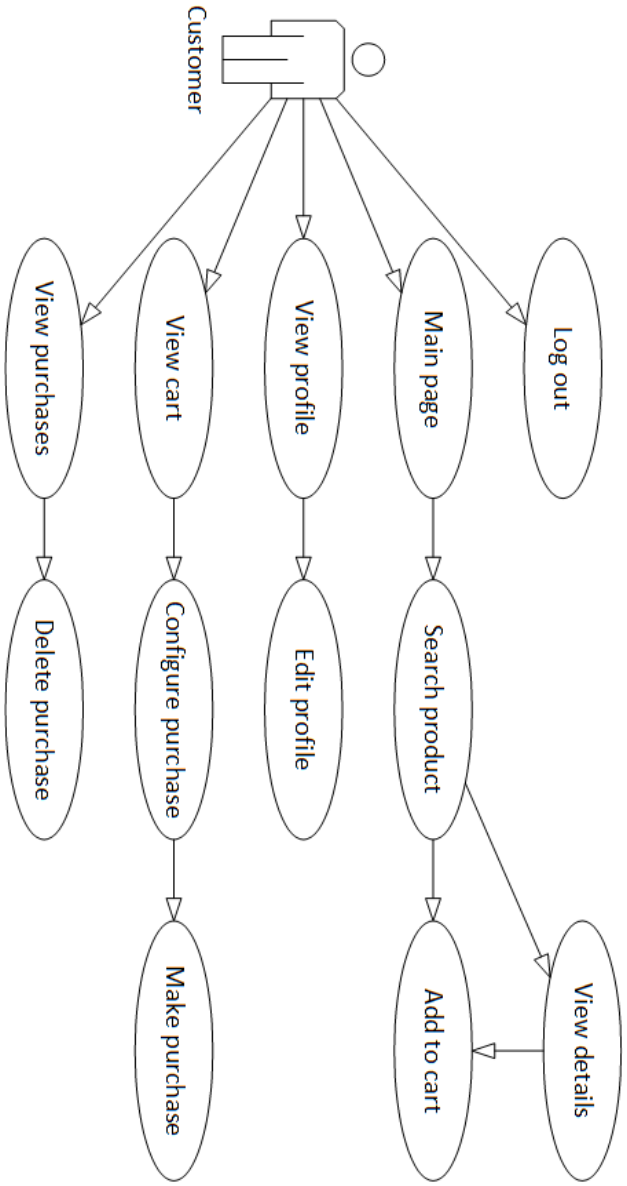
ПРИЛОЖЕНИЕ В
(обязательное)
USE-CASE диаграмма



					ГУИР.153503.061.02	
Изм.	Лист	№ докум.	Подп.	Дата		
Разраб.		Кончик			USE-CASE диаграмма	Лит. У / Масса / Масштаб 1:1
Пров.		Владымцев				Лист 1 / Листов 3
						Кафедра информатики группа 153503

Формат А4

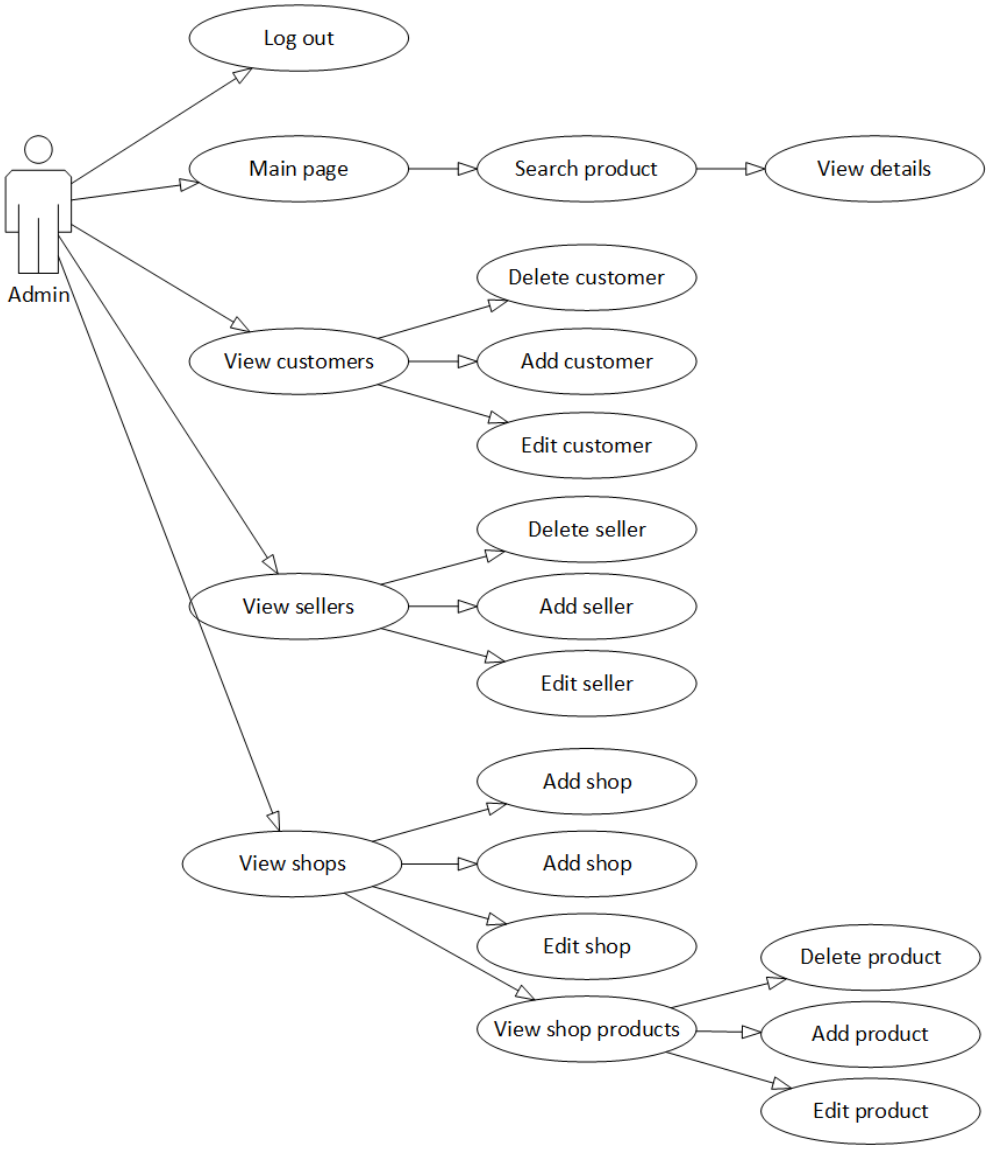
ГУИР.153503.061.02



					ГУИР.153503.061.02		
Изм.	Лист	№ докум.	Подп.	Дата			
Разраб.		Кончик			USE-CASE диаграмма	Лит.	Масса / Масштаб
Пров.		Владымцев				У	1:1
						Лист 2	Листов 3
						Кафедра информатики группа 153503	

Формат А4

ГУИР.153503.061.02



					ГУИР.153503.061.02	
Изм.	Лист	№ докум.	Подп.	Дата		
Разраб.		Кончик			USE-CASE диаграмма	Лит. У / Масса / Масштаб 1:1
Пров.		Владымцев				Лист 3 / Листов 3
						Кафедра информатики группа 153503

Формат А4

ПРИЛОЖЕНИЕ Г
(обязательное)
Графический интерфейс

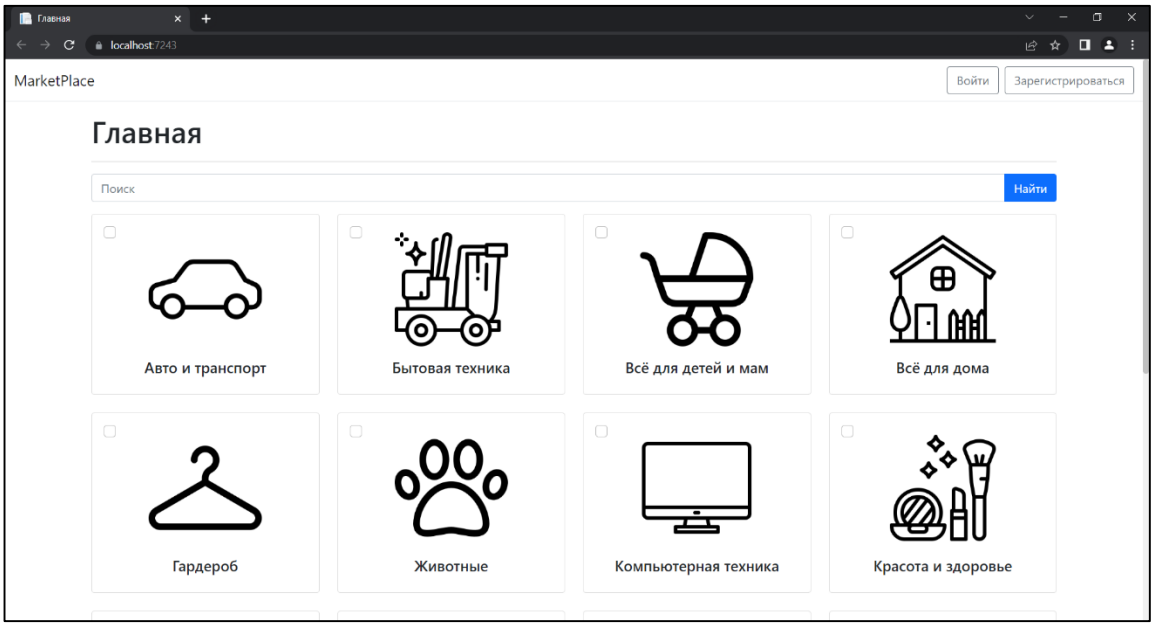


Рисунок 1 – Главная страница

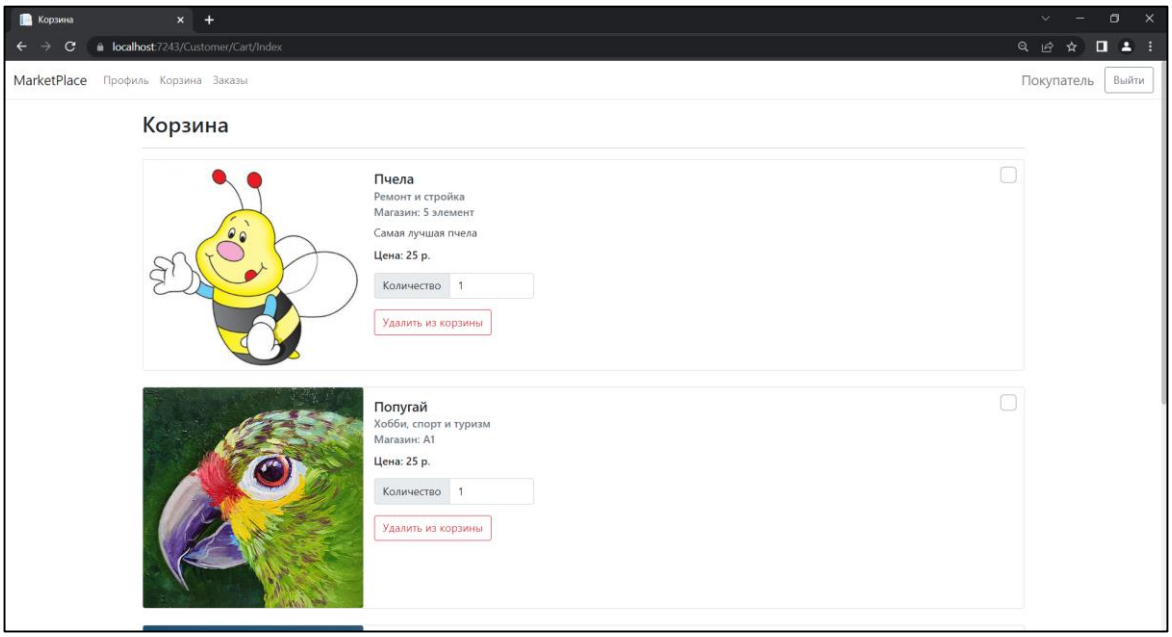


Рисунок 2 – Корзина покупателя

					ГУИР 153503.061.03 ПЛ			
Изм	Лист	№ докум	Подпись	Дата				
Разраб.		Кончик			Графический интерфейс	Литера	Лист	Листов
Пров.		Владымцев				У		
Рец.						Кафедра информатики группа 153503		
Н.контр.		Владымцев						
Утв.		Владымцев						