МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ

ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет Информационных технологий

Кафедра Программной инженерии

Специальность 1-40 01 01 «Программное обеспечение информационных технологий»

Направление специальности 1-40 01 01 «Программное обеспечение информационных технологий»(программирование интернет приложений)

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**КУРСОВОГО ПРОЕКТА:**

по дисциплине «Объектно-ориентированные технологии программирования и стандарты проектирования»

Тема Программное средство «Магазин спортивных товаров»

Исполнитель

Студент(ка) 2 курса группы 5 Хатченок Дмитрий Николаевич

(Ф.И.О.)

Руководитель работы асс. Панченко О.Л.

(учен. степень, звание, должность, подпись, Ф.И.О.)

Курсовой проект защищен с оценкой

Председатель Панченко О.Л.

(подпись)

Минск 2024

**Содержание**

[**Введение 4**](#_Toc166835258)

[**1 Аналитический обзор прототипов и литературных источников 5**](#_Toc166835259)

[**1.1 Обзор аналогичных решений 5**](#_Toc166835260)

[**1.2 Постановка задачи 7**](#_Toc166835261)

[**1.3 Вывод по разделу 8**](#_Toc166835262)

[**2 Анализ требований к программному средству и разработка требований 9**](#_Toc166835263)

[**2.1 Технические средства разработки 9**](#_Toc166835264)

[**2.1.1 Язык программирования C# и платформа .Net 9**](#_Toc166835265)

[**2.1.2 Entity Framework 10**](#_Toc166835266)

[**2.1.3 Windows Presentation Foundation (WPF) 11**](#_Toc166835267)

[**2.1.4 Microsoft SQL Server 11**](#_Toc166835268)

[**2.2 Спецификация функциональных требований к программному средству 12**](#_Toc166835269)

[**3 Проектирование программного средства 13**](#_Toc166835270)

[**3.1 Проектирования логической структуры базы данных 13**](#_Toc166835271)

[**3.2 Проектирование архитектуры приложения 16**](#_Toc166835272)

[**3.3 UML-диаграмма классов 18**](#_Toc166835273)

[**3.4 Проектирование последовательности взаимодействия 18**](#_Toc166835274)

[**4 Реализация программного средства 20**](#_Toc166835275)

[**4.1 Основные классы и методы программного средства 20**](#_Toc166835276)

[**4.2 Реализация паттерна MVVM 20**](#_Toc166835277)

[**4.3 Классы для реализации паттерна Command 21**](#_Toc166835278)

[**4.4 Классы для реализации взаимодействия с базой данных 23**](#_Toc166835279)

[**4.5 Реализация связи между моделями представления 24**](#_Toc166835280)

[**5 Тестирование, проверка работоспособности и анализ полученных результатов 26**](#_Toc166835281)

[**5.1 Тестирование регистрации и авторизации 26**](#_Toc166835282)

[**5.2 Тестирование заказа товара 27**](#_Toc166835283)

[**5.3 Вывод по разделу 29**](#_Toc166835284)

[**6. Руководство по использованию 30**](#_Toc166835285)

[**Заключение 36**](#_Toc166835286)

[**Список использованных источников 37**](#_Toc166835287)

[**Приложение А 38**](#_Toc166835288)

[**Приложение Б 39**](#_Toc166835289)

[**Приложение В 40**](#_Toc166835290)

[**Приложение Г 41**](#_Toc166835291)

[**Приложение Д 44**](#_Toc166835292)

[**Приложение Е 50**](#_Toc166835293)

# **Введение**

Спорт становится неотъемлемой частью современного образа жизни, и вместе с растущим интересом к здоровому образу жизни увеличивается и спрос на спортивные товары различного назначения. Онлайн магазины стремительно вытесняют с рынка обычные за счет своей доступности в любом месте и в любое время, что очень удобно для современного человека, у которого нет времени съездить в магазин и выбрать нужный товар.

Целью данного курсового проектирования является разработка программного средства, используя язык программирования C#, а также систему для построения клиентских приложений Windows WPF (Windows Presentation Foundation). Данное курсовое проектирование будет направлено на создание простого и удобного интерфейса для взаимодействия с пользователями и администраторами.

Реализованным программным средством будет «Магазин спортивных товаров». Пользователи, которые будут использовать данное программное средство, смогут:

1. Просматривать каталог товаров, разделенный на категории, где можно найти любой подходящий товар.
2. Использовать поиск товара по названию и, при необходимости, с указанием фильтров для поиска.
3. Получать подробную и актуальную информацию о товаре.
4. Добавить понравившиеся товары в корзину и заказать доставку на указанный адрес в нужное время.
5. Оставлять отзывы о товаре с указанием оценки. Это поможет другим пользователям получить дополнительную информацию о товаре.

В процессе разработки будет проведен анализ существующих аналогов на рынке, определены ключевые требования и задачи, спроектированы структуры баз данных и архитектура приложения. Завершающим этапом станет тестирование функционала и составление подробной документации, обеспечивающей легкость в освоении и использовании программного продукта.

По итогу, разработанное программное средство позволит пользователю ознакомиться с доступным каталогом товаров и получить в минимальные сроки товар, который позволит полноценно вести здоровый и активный образ жизни, развиваться в спортивном направлении.

# **1 Аналитический обзор прототипов и литературных источников**

## **Обзор аналогичных решений**

Одним из первых этапов в создании программного продукта является анализ прототипов и литературных источников. На сегодняшний день можно встретить достаточно большое количество программных решений со схожей тематикой. По этой причине, мы рассмотрим ряд таких программных средств.

Первый аналог – «Sportmaster.by»[1]. На рисунке 1.1 представлен интерфейс данного интернет-ресурса:

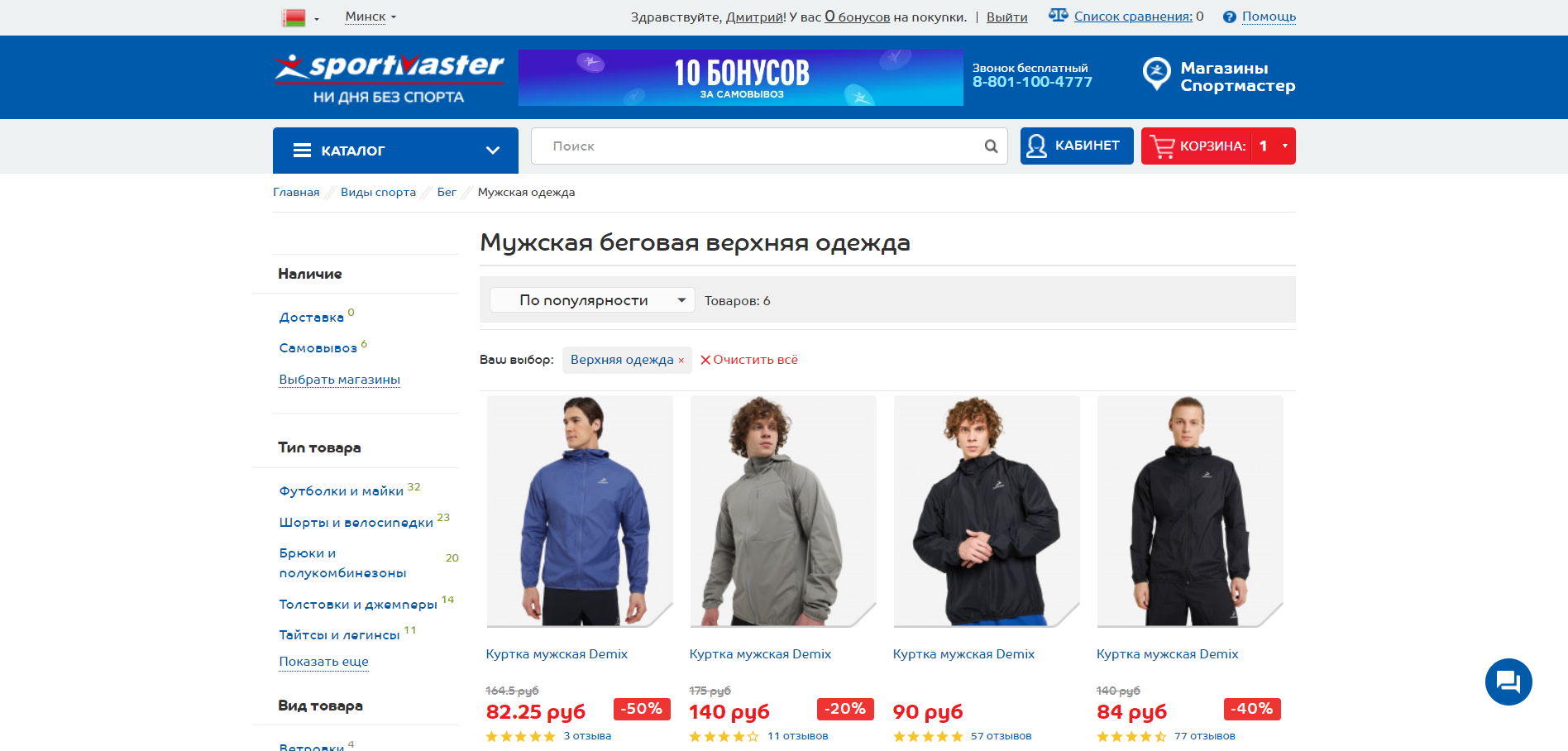


Рисунок 1.1 – Интерфейс «Sportmaster.by»

Sportmaster – это магазин спортивной одежды и инвентаря, работающие во всех странах СНГ. На сайте представлена широкая линия одежды и инвентаря для спорта. Ознакомившись с данным сайтом можно выделить преимущества и недостатки данного интернет ресурса:

Преимущества интернет-ресурса:

* огромное разнообразие категорий товаров;
* наличие поиска и фильтрации, что делает данный интернет-ресурс удобным для использования;
* наличие акций и программы лояльности для постоянных клиентов.

Недостатки интернет-ресурса:

* отсутствует доставка товаров клиенту, имеется только самовывоз;
* дизайн выглядит достаточно устаревшим.

В целом, «Sportmaster.by» предоставляет подробную информацию о имеющихся товарах. На данном интернет-ресурсе также присутствуют отзывы клиентов, с которыми пользователь может ознакомиться при выборе конкретного товара.

В качестве второго аналога рассмотрим «Nike»[2]. На рисунке 1.2 представлен интерфейс данного интернет-ресурса:

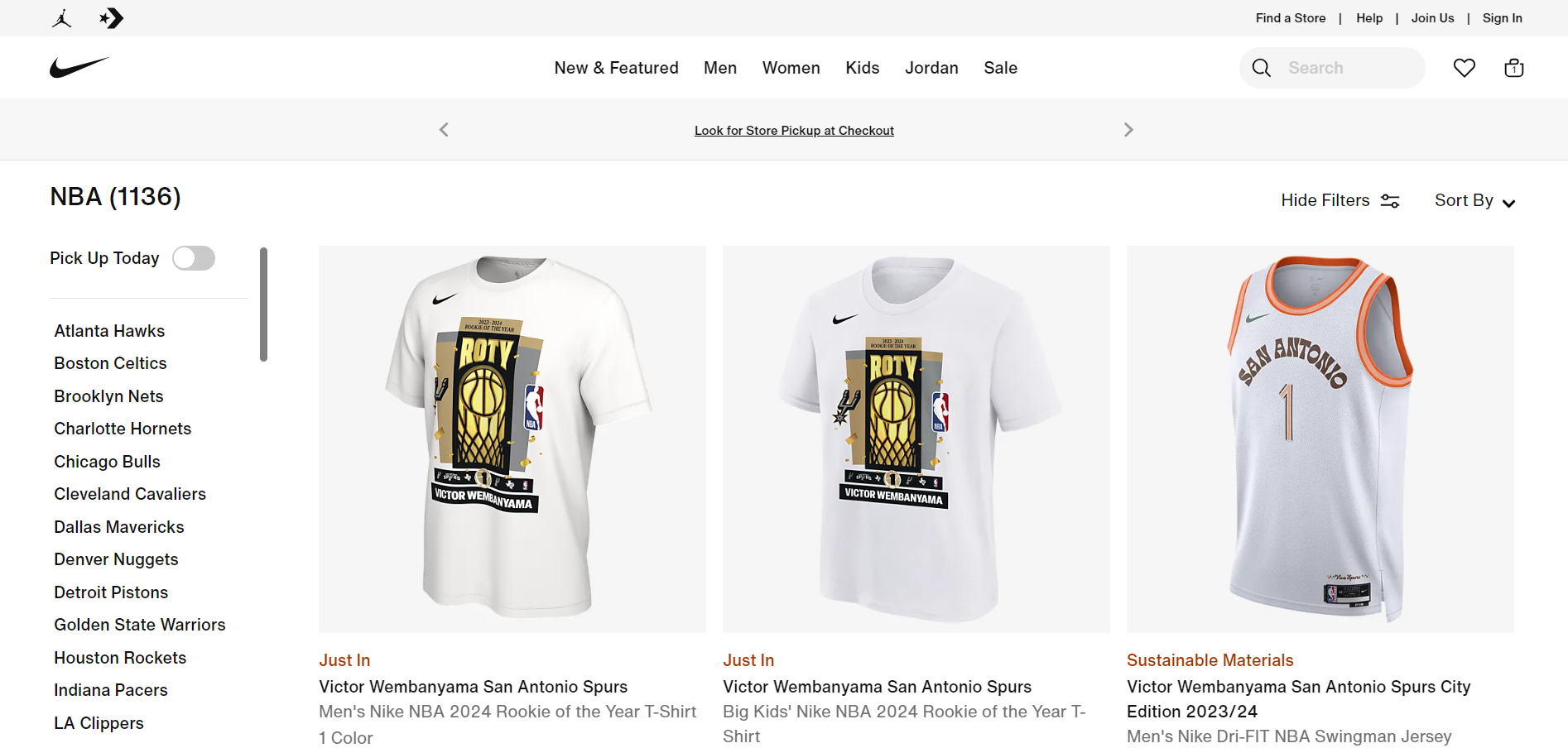


Рисунок 1.2 – Интерфейс «Nike»

Nike – один из самых популярных брендов спортивной одежды по всему миру. На данном интернет-ресурсе представлен огромных выбор одежды для различных видов спорта, а также инвентарь. Бренд сотрудничает с различными командами и спортивными лигами, что делает его более распространенным и единым со спортом в целом.

Преимущества интернет-ресурса:

* большой выбор товаров по различным критериям;
* простой и стильный дизайн;
* наличие доставки по всему миру;
* удобная система фильтрации и поиска.

Интернет-ресурс «Nike» не имеет каких-либо существенных недостатков, что делает его одним из самых востребованных в своей сфере. «Nike» имеет достаточно большую и разнообразную клиентскую базу. Поэтому интернет-ресурс должен подходить под различные требования пользователей, с чем у него нет каких-либо проблем.

Третий аналог – «Sport-center.by»[3]. На рисунке 1.3 можно увидеть интерфейс данного аналога:

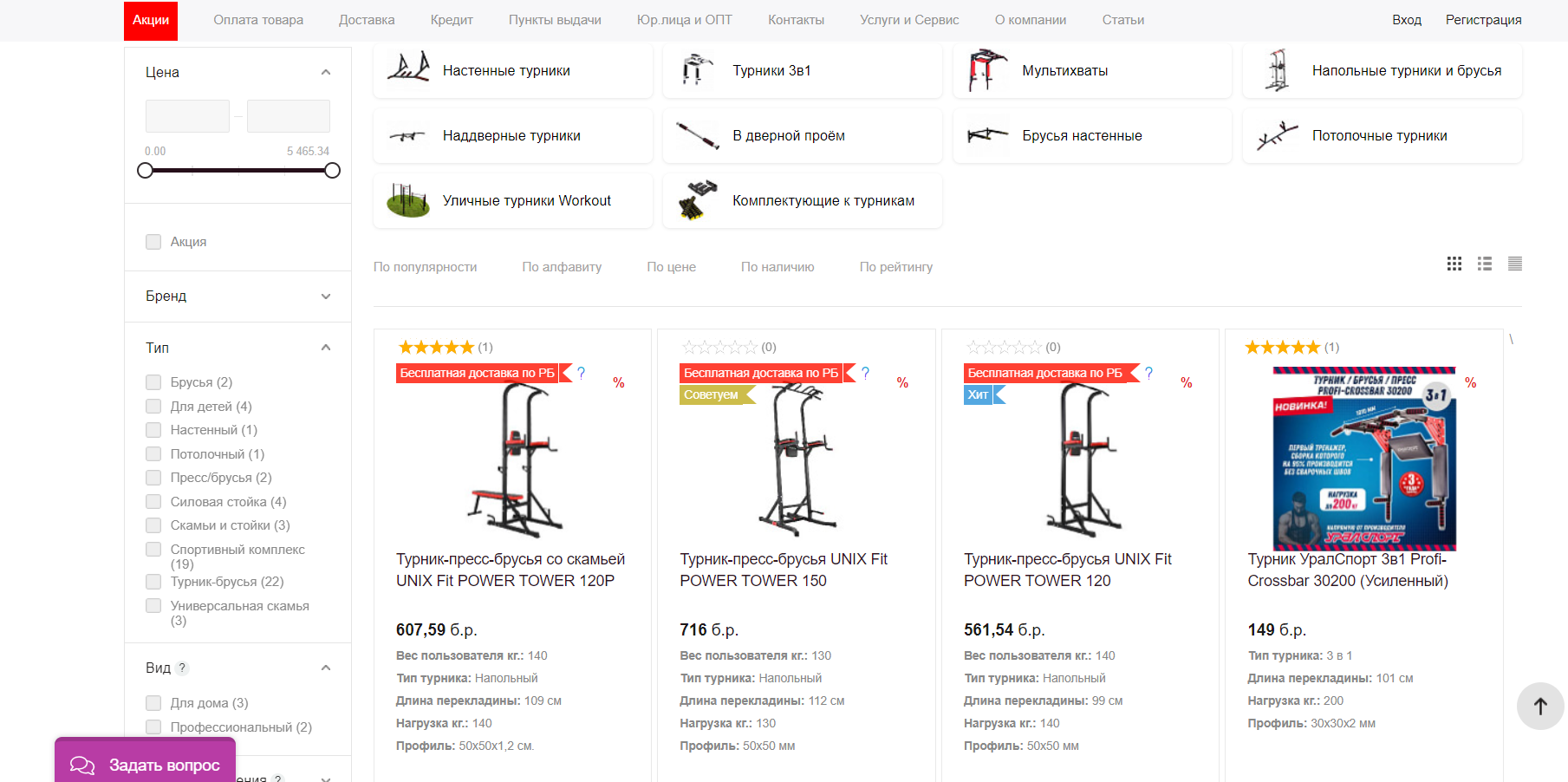


Рисунок 1.3 – Интерфейс «Sport-center.by»

Sport-center – это популярный на территории Беларуси интернет-магазин, которые представляет широкий выбор тренажеров и инвентаря для занятие спортом дома.

Преимущества интернет-ресурса:

* обширный каталог с хорошей фильтрацией;
* наличие видеообзоров на товары;
* простой и понятный интерфейс;

Недостатки:

* отсутствует какой-либо выбор спортивной одежды;

Интерфейс интернет-ресурса «Sport-center.by» интуитивно понятен, что привлекает огромное количество клиентов. Интернет-магазин предоставляет возможность доставки по всей территории Республики Беларусь. Основной функционал присутствует.

1.2 Постановка задачи

После анализа прототипов и аналогичных решений, их преимуществ и недостатков, можно выделить ряд основных задач, которые должно решать программное средство.

Основные задачи:

* создание интуитивно понятного и привлекательного пользовательского интерфейса, который обеспечит удобную навигацию и понимание функционала приложения;
* разработка эффективного механизма поиска и фильтрации товаров, который позволит пользователям быстро находить необходимые продукты с учетом их предпочтений и требований.
* внедрение системы управления корзиной;
* создание функционала для системы отзывов, позволяющей пользователям делиться своими впечатлениями о продукции и оценивать ее качество;
* реализация персонализированного кабинета пользователя с возможностью просмотра истории заказов и управления персональными данными;
* проведение всестороннего тестирования программного средства в различных сценариях использования для выявления и устранения возможных ошибок и недочетов.
* подготовка подробного руководства пользователя, содержащего информацию о установке приложения, его основных функциях и способах использования.

При разработке программного средства необходимо учитывать требования и пожелания пользователей, а также использовать современные технологии и методы программирования, чтобы обеспечить удобство и безопасность использования.

**1.3 Вывод по разделу**

После оценки аналогов можно заметить, что многие программы имеют схожий функционал. Исходя из этого были выделены основные задачи проекта: разработка программного средства, позволяющего пользователям просматривать каталог товаров, иметь возможность использовать поиск и фильтрацию в каталоге, иметь возможность просмотра подробной информации о товаре, добавление и просмотр отзывов, добавление товара в корзину, заказ товара из корзины и просмотр последних заказов в профиле. Интерфейс программного средства должен быть простым, интуитивно понятным и удобным.

**2 Анализ требований к программному средству и разработка требований**

Анализ требований – это процесс сбора требований к программному обеспечению, их систематизации, документирования, анализа, выявления противоречий, неполноты, разрешения конфликтов в процессе разработки программного обеспечения.

Цель анализа требований в проектах – получить максимум информации о заказчике и специфике его задач, уточнить рамки проекта, оценить возможные риски. На этом этапе происходит идентификация принципиальных требований методологического и технологического характера, формулируются цели и задачи проекта, а также определяются критические факторы успеха, которые впоследствии будут использоваться для оценки результатов внедрения. Определение и описание требований – шаги, которые во многом определяют успех всего проекта, поскольку именно они влияют на все остальные этапы. Этот раздел начнём с технических средств, которые использовались для разработки программного средства, после чего перейдём к архитектуре системы и диаграмме вариантов использования.

**2.1 Технические средства разработки**

Для создания программного средства «Магазин спортивных товаров» были выбраны передовые технологии и инструменты, чтобы обеспечить высокое качество и удобство использования. В процессе разработки был использован язык программирования C#, платформа .Net Framework и инновационная технология Windows Presentation Foundation (WPF).

Организация работы с базой данных в приложении осуществляется с помощью мощного фреймворка Entity Framework, который обеспечивает эффективное взаимодействие с базой данных. База данных разработана с использованием Microsoft SQL Server, обеспечивая надежность и гибкость хранения данных.

**2.1.1 Язык программирования C# и платформа .Net**

C# – это не только объектно-ориентированный язык программирования, но и сокровищница возможностей для разработки мощных и гибких приложений в различных областях. Рожденный в конце 90-х годов прошлого века благодаря талантливым разработчикам из Microsoft, C# привносит свежесть и инновации в мир программирования.

Один из главных вдохновителей и архитекторов C# – Андерс Хейлсберг, чье имя тесно связано с идеями Turbo Pascal и Delphi. Под его руководством C# стал языком, способным воплотить амбициозные проекты и преобразить их в реальность.

А что же платформа .Net? Она представляет собой еще одно грандиозное достижение Microsoft. Сочетая в себе набор уникальных особенностей, .Net становится мощной платформой для создания приложений. Во-первых, она поддерживает не только C#, но и другие языки программирования и их диалекты, что расширяет возможности разработчиков. Во-вторых, .Net является переносимой платформой, способной работать на различных операционных системах благодаря своей актуальной версии, доступной для большинства платформ. В-третьих, платформа предоставляет разработчикам мощную библиотеку классов, которая является их надежным партнером в создании сложных приложений. И наконец, автоматическая сборка мусора в .Net облегчает жизнь разработчикам, позволяя им фокусироваться на создании функциональности, не отвлекаясь на управление памятью.

Таким образом, C# и платформа .Net представляют собой мощный союз, обеспечивающий разработчикам средства и среду для творчества и инноваций. Они открывают новые горизонты в программировании и являются надежными инструментами для создания современных и высокоэффективных приложений.

**2.1.2 Entity Framework**

Entity Framework – это инновационная объектно-ориентированная технология, основанная на фреймворке .Net, которая преображает работу с данными. Вместо традиционных таблиц, индексов и ключей, Entity Framework возвышает данные до уровня объектов, создавая удивительно элегантный уровень абстракции.

Основной концепцией, лежащей в основе Entity Framework, является сущность. Сущность представляет собой связанный с определенным объектом набор данных, открывая возможности для работы с данными на более высоком уровне абстракции. Забудьте о громоздких таблицах - здесь важны объекты и их связи.

Одной из ключевых особенностей Entity Framework является использование запросов LINQ, позволяющих легко и элегантно извлекать данные из базы данных. Это мощный инструмент, который дает разработчикам полный контроль над выборкой и манипуляцией данными.

Entity Framework предлагает три различных подхода для взаимодействия с базами данных: Database First, Model First и Code First. При разработке программного средства был выбран подход Database First, который позволяет начать с существующей базы данных и автоматически создать соответствующую модель объектов.

В итоге, благодаря Entity Framework, разработка программного средства становится не только эффективной, но и изящной. Эта технология поднимает работу с данными на новый уровень, обеспечивая удобство и гибкость взаимодействия с базами данных. Она становится надежным союзником разработчиков, помогая создавать высококачественные и масштабируемые приложения.

**2.1.3 Windows Presentation Foundation (WPF)**

Windows Presentation Foundation (WPF) представляет собой изысканную структуру пользовательского интерфейса, которая вдохнула жизнь в настольные клиентские приложения. Эта платформа является истинным творческим пространством, предоставляющим разработчикам мощный инструментарий для создания впечатляющих и интерактивных пользовательских интерфейсов.

Одной из фундаментальных особенностей WPF является использование языка XAML – декларативного языка описания интерфейса, основанного на XML. XAML открывает двери в мир бесконечных возможностей, предоставляя панели, элементы управления, документы и графические фигуры. Это позволяет разработчикам создавать сложные макеты с несколькими страницами, воплощая свои идеи в прекрасные визуальные композиции.

WPF воплощает суть красоты и гибкости, предоставляя возможность создавать стили, темы и словари ресурсов для приложений. Благодаря этому, дизайнеры и разработчики могут сотворить неповторимые пользовательские элементы управления и преобразить шаблоны уже существующих. Каждый элемент интерфейса становится частью художественного произведения, где визуальные эффекты и анимации оживляют пользовательский опыт.

В итоге, WPF представляет собой гармоничное слияние технологии и искусства, где разработчики могут воплотить свои творческие видения в великолепные настольные приложения. Она подарит вам не только функциональность, но и эстетическое удовольствие, воплощая мир мультимедиа и анимации в пользовательском интерфейсе. WPF - это вдохновение для создания приложений, которые не только впечатляют функциональностью, но и приносят радость взаимодействия с пользователем.

**2.1.4 Microsoft SQL Server**

Microsoft SQL Server является великолепной системой управления реляционными базами данных, созданной с нежным вниманием к деталям корпорацией Microsoft. В ее сердце заложена мощь и гибкость, которая обеспечивает высочайшую скорость работы, надежность и безопасность данных, а также простоту использования и администрирования.

С верным спутником - языком запросов Transact-SQL, Microsoft SQL Server становится готовым рыцарем, способным исполнить все желания. Transact-SQL предлагает богатый набор операторов, охватывающих создание, изменение и удаление объектов баз данных, манипулирование данными, контроль доступа и управление транзакциями. Этот язык открывает двери в мир гибкости и функциональности, позволяя воплотить самые смелые идеи в работу с данными.

Microsoft SQL Server не только предлагает выдающуюся производительность и надежность, но и обладает впечатляющей системой шифрования данных, обеспечивая безопасность и конфиденциальность важной информации. Она становится надежным союзником в сохранении ценных данных и защите их от нежелательных глаз.

В наше время Microsoft SQL Server является великим лидером среди систем управления базами данных. Ее популярность и превосходство подтверждают ее путь к славе, ставя ее на передовом месте в мире баз данных. Благодаря своим превосходным возможностям и широкому применению, Microsoft SQL Server становится надежным союзником в решении самых сложных задач управления данными.

**2.2 Спецификация функциональных требований к программному средству**

Для выявления функциональных требований были определены следующие роли и группы пользователей:

1. Пользователь:

Пользователь имеет следующий функционал:

* выполнять регистрацию и авторизацию в аккаунт;
* просматривать каталог товаров;
* просматривать страницу товара;
* выполнять поиск товара с использованием фильтров;
* оставлять отзыв о товаре;
* добавлять товар в корзину;
* заказывать товары из корзины.

1. Администратор:

Администратор имеет следующий функционал:

* выполнять авторизацию в аккаунт;
* просматривать каталог товаров;
* просматривать страницу товара;
* выполнять поиск товара с использованием фильтров;
* добавлять, изменять и удалять товар;
* просматривать заказы;
* просматривать статистику товара.

В приложении А представлена диаграмма использования программного средства разными группами пользователей. В целом, спецификация функциональных требований к программному средству позволяет определить функциональность и ожидания пользователей, а также задает рамки для разработки и тестирования программного продукта.

# **3 Проектирование программного средства**

## **3.1 Проектирования логической структуры базы данных**

Для создания базы данных описываемого приложения использовалась система управления реляционными базами данных Microsoft SQL Server 2022.

База данных – это совокупность данных, организованных по определенным правилам, предусматривающим общие принципы описания, хранения и манипулирования данными, независимо от прикладных программ.

Система управления базами данных – совокупность программ и языковых средств, предназначенных для управления данными в базе данных, ведения базы данных и обеспечения взаимодействия ее с прикладными программами. База данных данного курсового проекта состоит из 5 таблиц. Схема базы данных изображена на рисунке 3.1

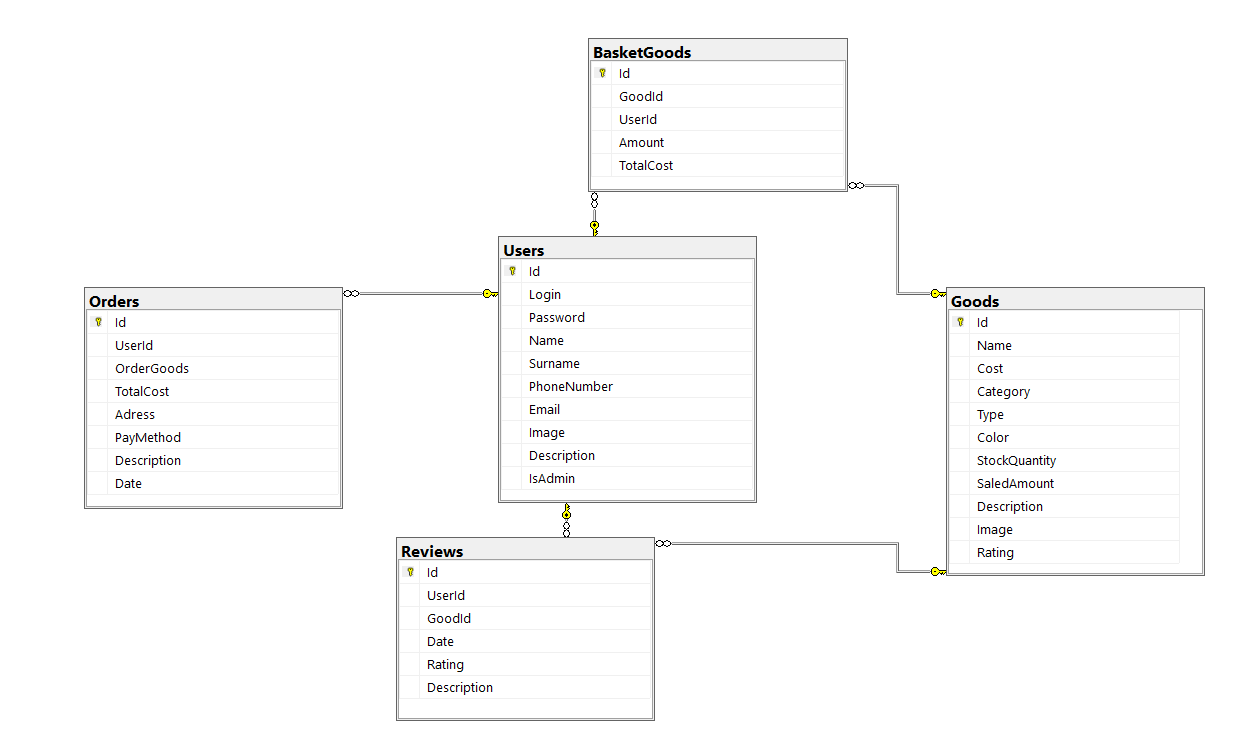


Рисунок 3.1 – Структура базы данных

Остановимся более подробно на таблицах и рассмотрим какие поля они содержат.

Таблица Users содержит информацию про всех зарегистрированных пользователей. isAdmin определяет какой у аккаунта – пользователь или администратор, в зависимости от этого, при авторизации интерфейс и возможности будут различаться. На рисунке 3.2 представлена подробная информация про данную таблицу:

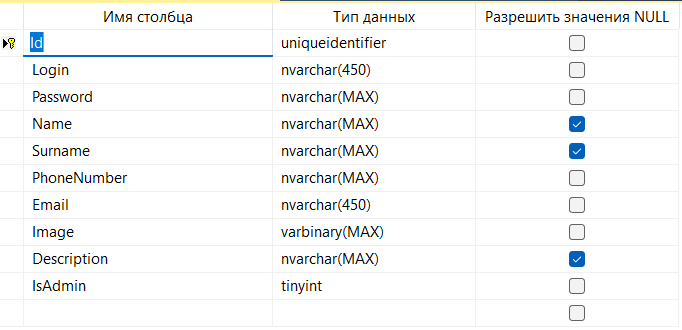


Рисунок 3.2 – Таблица Users

Таблица Goods содержит всю информацию про товары. На риснуке 3.3 представлена информация про эту таблицу:

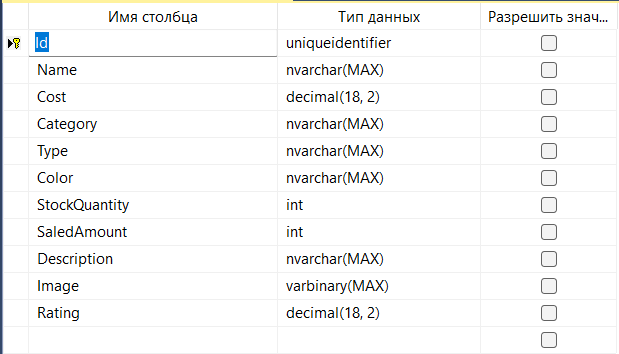


Рисунок 3.3 – Таблица Goods

Таблица Reviews содержит отзывы пользователей. В таблице используются вторичные ключи UserId для связи с таблицей User и GoodId для связи с таблицей Goods. Структура таблицы представлена на рисунке 3.4:

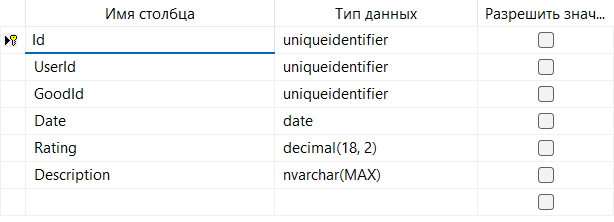


Рисунок 3.4 – Таблица Reviews

Таблица BasketGoods содержит информацию о товарах в корзине. В таблице используются вторичные ключи UserId для связи с таблицей User и GoodId для связи с таблицей Goods. Структура таблицы представлена на рисунке 3.5:

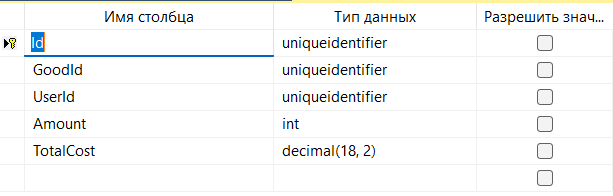


Рисунок 3.5 – Таблица BasketGoods

Таблица Orders содержит информацию о заказах пользователей. В таблице используется вторичный ключ UserId для связи с таблицей User. Структура таблицы представлена на рисунке 3.6:

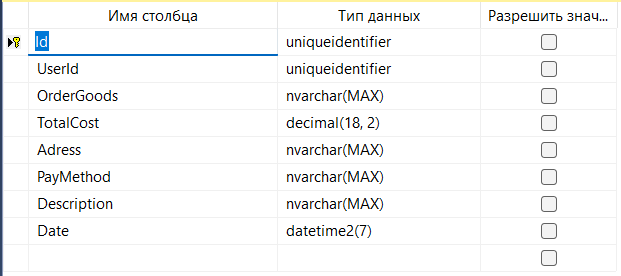


Рисунок 3.6 – Таблица Orders

Между таблицами реализована связи один ко многим. Связь один ко многим значит, что несколько строк дочерней таблицы зависят от одной строки родительской таблицы.

При работе с базой данных, нашим выбранным инструментом является Entity Framework Core 6. Этот фреймворк предоставляет нам множество значимых преимуществ. Мы можем работать с таблицами базы данных, как с обычными классами на языке C#, где столбцы таблиц становятся свойствами этих классов. Кроме того, синтаксис SQL-запросов заменяется на более удобный и интуитивно понятный LINQ. Entity Framework Core берет на себя ответственность по преобразованию нашего кода на C# в соответствующие SQL-инструкции.

В проектировании базы данных существуют три подхода, из которых мы выбрали подход Code-First. При использовании данного подхода мы отказываемся от использования модели EDMX и вместо этого вручную настраиваем классы C# в объектной модели данных. Code-First поддерживает как генерацию классов сущностей из существующей базы данных, так и создание базы данных из ранее созданной модели объектов C#. Это дает нам полный контроль над структурой базы данных и позволяет более гибко адаптироваться к требованиям проекта.

Использование Entity Framework Core и подхода Code-First позволяет нам более эффективно и удобно работать с базой данных, а также обеспечивает гибкость и контроль в процессе разработки и поддержки приложения.

## **3.2 Проектирование архитектуры приложения**

Архитектура программного обеспечения представляет собой совокупность ключевых решений, касающихся организации программной системы. Она включает выбор структурных элементов и их интерфейсов, определение их взаимодействия, а также объединение этих элементов в более крупные системы. Однако, центральную роль играет архитектурный стиль, который определяет всю организацию системы, включая элементы, их интерфейсы, их сотрудничество и способы их соединения.

Для удовлетворения требований проектируемой системы с точки зрения различных атрибутов качества, применяются различные архитектурные шаблоны, так называемые паттерны. В разрабатываемом нами приложении мы используем архитектурный шаблон Model-View-ViewModel (MVVM). Этот шаблон состоит из трех компонентов: модели (Model), модели представления (ViewModel) и представления (View).

На рисунке 3.8 представлена диаграмма, которая показывает общую структуру приложения в рамках шаблона MVVM.



Рисунок 3.7 – Структура шаблона MVVM

Модель описывает данные, используемые в приложении. В моделях может содержаться логика, связанная с этими данными, например, валидация свойств модели. В то же время модель не должна содержать логику, связанную с отображением данных и взаимодействием с визуальными элементами управления. Часто модель реализует интерфейс INotifyPropertyChanged, который позволяет системе автоматически обнаруживать изменения свойств модели и облегчает привязку к представлению, хотя сама модель не взаимодействует напрямую с представлением.

Представление, или View, определяет визуальный интерфейс, через который пользователь взаимодействует с приложением. Важным аспектом является эффективное проектирование пользовательского интерфейса, чтобы создать приятный и удобный опыт для конечного пользователя. Представление не обрабатывает события, а в основном выполняет действия посредством команд.

ViewModel, или модель представления, связывает модель и представление через механизм привязки данных. Если значения свойств модели изменяются, с использованием интерфейса INotifyPropertyChanged, данные в представлении автоматически обновляются, хотя сама модель и представление не прямо связаны. ViewModel также содержит логику получения данных из модели и обновления данных в модели.

Исходя из этого, проект был разделен на соответствующие папки. Описание структуры основных папок проекта представлено в таблице 3.1.

Таблица 3.1 – Описание структуры папок проекта

|  |  |
| --- | --- |
| Название папки/файла | Описание |
| Command | Содержит класс, реализующий интерфейс ICommand |
| Database | Содержит класс SportsContext, необходимый для работы с базой данных |
| Migrations | Содержит миграции для базы данных |
| ResourceDictionaries | Cодержит стили для элементов управления |
| Resources | Содержит изображения, используемые в приложении |
| Models | Хранит модели данных |
| ViewModels | Хранит реализацию ViewModel |
| View | Хранит представления окон |

## **3.3 UML-диаграмма классов**

В приложении Б представлена UML-диаграмма классов, используемых в проекте. В рамках разработки программного средства диаграмма классов является одним из ключевых инструментов моделирования и проектирования системы. Диаграмма классов представляет структуру системы, ее классы, атрибуты и отношения между классами.

В процессе проектирования программы диаграмма классов помогает определить основные классы, их свойства и методы. Она позволяет увидеть структуру данных, которые будут использоваться в программе, а также взаимосвязи между классами. Диаграмма классов также помогает лучше понять концептуальную модель системы и обеспечивает основу для дальнейшей разработки.

3.4 Проектирование последовательности взаимодействия

При проектировании последовательности взаимодействия объектов приложения была использована UML-диаграмма последовательности.

Диаграмма последовательности (Sequence Diagram) – это один из видов UML-диаграмм, который используется для описания последовательности взаимодействия объектов в системе. Диаграмма последовательности часто используется для описания процессов бизнес-логики, взаимодействия между компонентами системы, процессов ввода-вывода, взаимодействия между пользователем и системой и т.д.

Диаграмма последовательности взаимодействия пользователя, приложения и базы данных при заказе товара из корзины представлена в Приложении В.

**4 Реализация программного средства**

**4.1 Основные классы и методы программного средства**

В данном разделе будут рассмотрены основные классы с точки зрения архитектуры программы. Естественно, что при разработке программного средства создаётся большое множество и других вспомогательных классов, но они не несут такой же большой ценности, как классы, описанные ниже.

Мы остановимся на классах:

* Для реализации паттерна MVVM;
* Для реализации паттерна Command;
* Для взаимодействия с базой данных.

**4.2 Реализация паттерна MVVM**

Как отмечалось в предыдущих разделах, при разработке приложения использовался архитектурный паттерн MVVM. Для его реализации необходимо использовать следующие классы:

* ViewModel (модель представления);
* View (визуальный интерфейс);
* Model (модель данных).

На примере страницы Basket рассмотрим пример использования вышеупомянтой модели. В листинге 4.1 представлен фрагмент кода из xaml, демонстрирующий использование связывания для команды. В приложении Г можно ознакомиться с примером реализации xaml-кода для всей страницы.

<Button Style="{DynamicResource BlackWhiteButton}" Height="60" Content="Удалить" Command="{Binding DataContext.DeleteFromBasket, RelativeSource={RelativeSource AncestorType={x:Type UserControl}}}" CommandParameter="{Binding}"></Button>

Листинг 4.1 – XAML-код при связывании команд

Связывание команды DeleteFromBasket происходит при помощи Binding, которое передаётся в свойство Command, а в ComandParameter передаётся параметр, в данном случае это экземпляр объекта класса BasketGoods, к которому привязана кнопка.

В приложении Д представлена реализация модели представления для страницы Basket. Каждая модель представления наследуется от класса ViewModelBase, в котором реализован метод RaisePropertyChanged, который позволяет сообщать представлению об изменении свойства объекта модели представления. Пример использования данного метода представлен в листинге 4.2:

public Order CurrentOrder {

get => currentOrder;

set {

currentOrder = value;

RaisePropertyChanged(nameof(CurrentOrder));

}

}

Листинг 4.2 – Пример использования RaisePropertyChanged

Модель представления создается при инициализации страницы через конструктор и содержит в себе свойства, вспомогательные методы и команды. Команда представляет собой экземпляр класса MyCommand, который мы рассмотрим в последующем подразделе.

И, наконец, модель данных представляет из себя обычный класс, объекты которого и будут использоваться на странице для отображения данных. В приложении Е представлен пример реализации модели на основе класса BasketGood.

**4.3 Классы для реализации паттерна Command**

Как уже можно было заметить, паттерн MVVM вплотную взаимодействует с командами, которые реализованы в данном случае с помощью паттерна Command.

Все команды в проекте являются экземплярами класса MyCommand, который в свою очередь реализует интерфейс ICommand. Класс MyCommand представлен в листинге 4.3:

public class MyCommand : ICommand

{

private Action<object> execute;

private Func<object, bool> canExecute;

public event EventHandler CanExecuteChanged

{

add { CommandManager.RequerySuggested += value; }

remove { CommandManager.RequerySuggested -= value; }

}

public MyCommand(Action<object> execute, Func<object, bool> canExecute = null)

{

this.execute = execute;

this.canExecute = canExecute;

}

public bool CanExecute(object parameter)

{

return this.canExecute == null || this.canExecute(parameter);

}

public void Execute(object parameter)

{

this.execute(parameter);

}

}

Листинг 4.3 – Класс MyCommand

В классе присутсвуют два поля: execute – действие, которое должно быть выполнено, когда команда вызвана, и canExecute – определяет, может ли команда в данный момент быть выполнена. В классе присутствует событие CanExecuteChanged, которое оповещает систему привязки об изменении состояния команды. Метод CanExecute возвращает значение true, когда команда может быть выполнена, метод Execute выполняет действие.

После создания команды, неободимо с ней взаимодействовать. Для этого, в модели представления у нас происходит вызов команды, которая связана непосредственно с ЭУ в нашем визуальном интерфейсе. В листинге 4.4 представлен пример вызова команды:

private MyCommand deleteFromBasket;

public MyCommand DeleteFromBasket

{

get

{

return deleteFromBasket ??= new MyCommand(obj =>

{

var good = obj as BasketGood;

CurrentOrder.TotalCost -= good.TotalCost;

GlobalResources.Context.BasketGoods.RemoveRange(GlobalResources.Context.BasketGoods.Where(bg => bg.Id == good.Id));

BasketGoods.Remove(good);

GlobalResources.Context.SaveChanges();

});

}

}

Листинг 4.4 – Пример вызова команды

Команды можно связывать как для стандартных событий, например, нажатие на кнопку, так и для других событий при помощи триггеров и бибилиотеки System.Windows.Interactivity.

**4.4 Классы для реализации взаимодействия с базой данных**

Осталось рассмотреть примеры классов, которые необходимы для работы с базой данных, взаимодействие с которой, происходит при помощи Entity Framework. Для работы с ним, необходимо было реализовать следующие классы:

* Сущность;
* Контекст данных;

Сущность представляет из себя обычный класс, в котором создаются свойства – это и будут столбцы в нашей базе данных. Пример такого класса приводится в Приложении Е. В качестве контекста данных выступпает класс, который наследуется от класса DbContext. В листинге 4.5 приводится реализация данного класса.

public class SportsContext:DbContext

{

public DbSet<User> Users { get; set; }

public DbSet<Good> Goods { get; set; }

public DbSet<Review> Reviews { get; set; }

public DbSet<BasketGood> BasketGoods { get; set; }

public DbSet<Order> Orders { get; set; }

public SportsContext() : base()

{

Database.EnsureCreated();

}

protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)

{

optionsBuilder.UseSqlServer("Data Source=(local);Initial Catalog=Sports\_Store;Integrated Security=True;TrustServerCertificate=true;");

}

protected override void OnModelCreating(ModelBuilder builder)

{

base.OnModelCreating(builder);

builder.Entity<User>().HasKey(e => e.Id);

builder.Entity<User>(entity => {

entity.HasIndex(e => e.Login).IsUnique();

});

builder.Entity<User>(entity => {

entity.HasIndex(e => e.Email).IsUnique();

});

builder.Entity<BasketGood>()

.HasOne(bg => bg.Good)

.WithMany(g => g.BasketGoods)

.HasForeignKey(bg => bg.GoodId);

builder.Entity<BasketGood>()

.HasOne(bg => bg.User)

.WithMany(u => u.BasketGoods)

.HasForeignKey(bg => bg.UserId);

builder.Entity<Review>()

.HasOne(r => r.User)

.WithMany(u => u.Reviews)

.HasForeignKey(r => r.UserId);

builder.Entity<Order>()

.HasOne(r => r.User)

.WithMany(u => u.Orders)

.HasForeignKey(r => r.UserId);

builder.Entity<Review>()

.HasOne(r => r.Good)

.WithMany(u => u.Reviews)

.HasForeignKey(r => r.GoodId);

}

}

Листинг 4.5 – Реализация класса SportsContext

В нем при помощи DbSet с указанием tempate-типа и происходит создание таблицы в нашей базе данных. В перегруженном методе OnConfiguring происходит подключение к БД. А в методе OnModelCreating – определение первичных и вторичных ключей.

**4.5 Реализация связи между моделями представления**

При соблюдении модульности моделей представления для каждой страницы важно иметь возможность пересылать информацию от одной модели представления к другой. Для этого в проекте используется библиотека MvvmLightLibs, которая предоставляет реализацию полезных методов и классов для реализации паттерна MVVM.

Один из этих классов – Messenger. Данный класс реализует паттерн «Наблюдатель», который используется для обмена сообщениями между моделями представления. Для начала нам надо зарегистрировать получение сообщений, которое выполняется в конструкторе класса. Пример представлен в листинге 4.6:

Messenger.Default.Register<Good>(this, "add to basket", good =>

{

if (!GlobalResources.Context.BasketGoods.Where(bg => (bg.UserId == GlobalResources.CurrentUser.Id && bg.GoodId == good.Id)).Any())

{

BasketGood NewBasketGood = new BasketGood(good);

BasketGoods.Add(NewBasketGood);

GlobalResources.Context.BasketGoods.Add(NewBasketGood);

GlobalResources.Context.SaveChanges();

CurrentOrder.TotalCost += NewBasketGood.TotalCost;

}

else

{

MessageBox.Show("Товар уже находится в вашей корзине!");

}

});

Листинг 4.6 – Пример регистрации получателя сообщений

При регистрации мы указываем тип сообщений, которые мы принимаем. Теперь мы можем отправлять сообщения при помощи метода Send, тем самым передавая объекты между моделями представления. Обмен сообщениями между компонентами приложения происходит без явной зависимости между ними.

# **5 Тестирование, проверка работоспособности и анализ полученных результатов**

## **5.1 Тестирование регистрации и авторизации**

Начнём тестирование с авторизации. Попробуем первым делом ввести неверные данные. В результате этих действий будет выведено соответствующее сообщение, которое демонстрируется на рисунке 5.1.

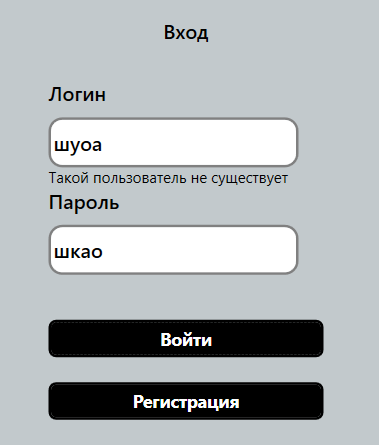


Рисунок 5.1 – Ошибка авторизации

Следующим этапом попробуем оставить все поля для заполнения пустыми. В результате данных действий будут выведены соответствующие сообщения. Демонстрируется это на рисунке 5.2.

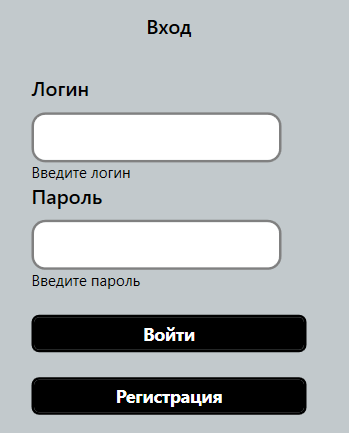


Рисунок 5.2 – Ошибка при заполнении полей

Как мы видим, окно авторизации работает корректно.

Попробуем при регистрации указать логин пользователя, который уже существует. На рисунке 5.3 демонстрируется результат, в котором выводится соответствующее сообщение.

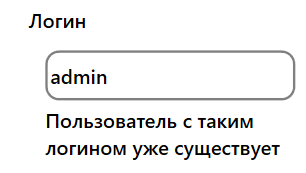


Рисунок 5.3 – Ошибка регистрации

Далее укажем неверный адрес электронной почты. На рисунке 5.4 показано окно, которое будет показано.

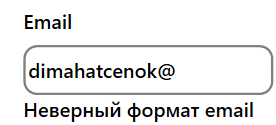


Рисунок 5.4 – Ошибка ввода электронной почты

Таким образом, можно сделать вывод, что окна для авторизации и регистрации обрабатывают все основные случаи возникновения ошибок.

## **5.2 Тестирование заказа товара**

В приложении интернет-магазина одной из главных составляющих является заказ товара из корзины. Корректно введенные данные позволяют администрации магазина четко и в короткие сроки ознакомиться с заказом и доставить его в считанные дни.

Допустим, в корзине нет товаров и мы попробуем оформить заказ. Мы не сможет сделать это, так кнопка становится активной, когда в корзине есть хотя бы один товар.

Попробуем увеличить количество товара на один. Результат отображен на рисунке 5.5.

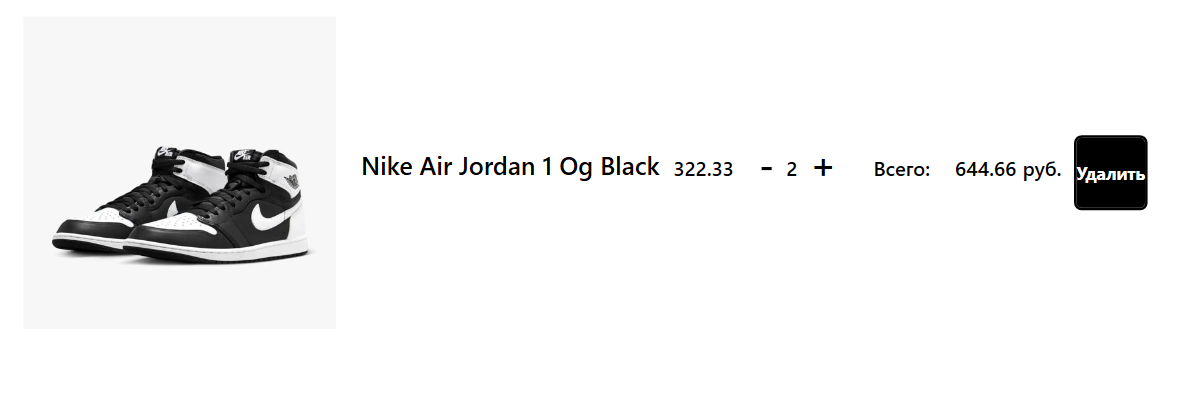


Рисунок 5.5 – Результат увеличения количества товара на один

Далее попробуем заказать товар, не введя корректно адрес доставки. Результат представлен на рисунке 5.6.

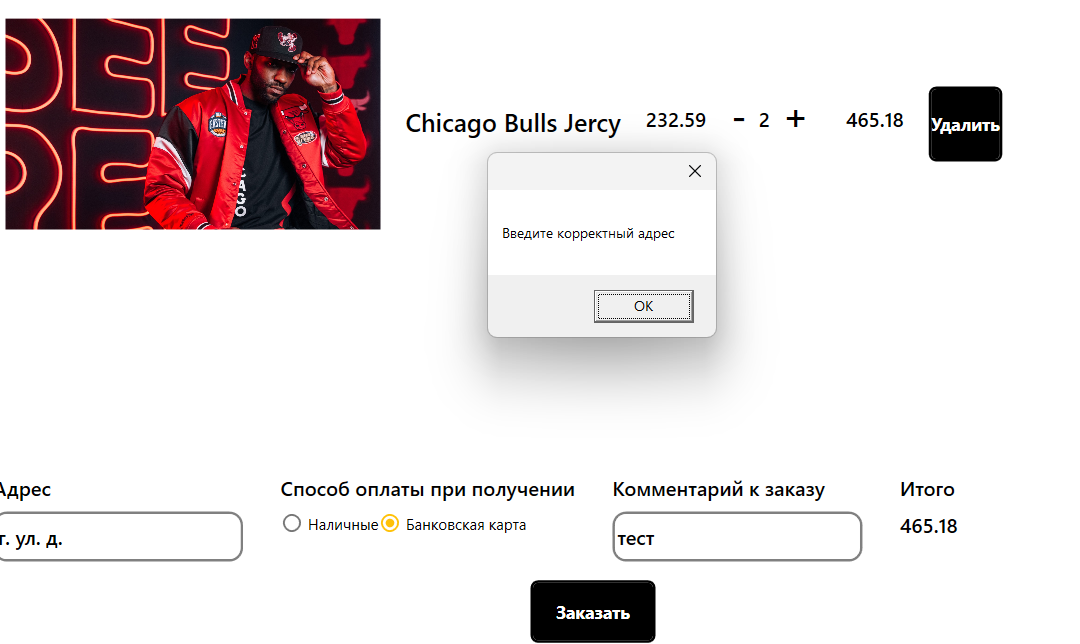


Рисунок 5.6 – Ошибка некорректного адреса

Попробуем заказать товара больше, чем количество, имеющееся на складе. Результат представлен на рисунке 5.7.

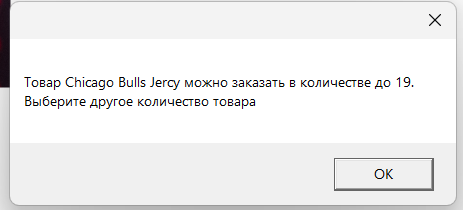


Рисунок 5.7 – Сообщения при неправильном вводе даты

Также попробуем заказать товар, если его нет в наличии. В результате будет выведено сообщение, которое показано на рисунке 5.8.

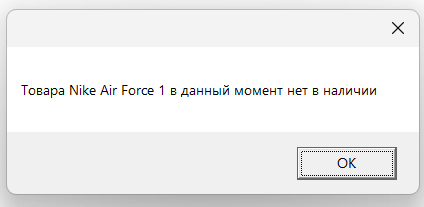


Рисунок 5.8 – Сообщения с оповещением про ошибку ввода логина пользователя

5.3 Вывод по разделу

В ходе тестирования была проверена обработка исключений и было подтверждено, что пользователи не смогут случайно выбить не обрабатывающееся исключение, которое приведет к сбою работы программы, проверена валидация полей. Также были протестированы некоторое функции пользователя, которые могли привести к некорректной работе программы. Проведенное тестирование подтвердило работоспособность программного средства.

6. Руководство по использованию

При запуске приложения появляется окно авторизации, представленное на рисунке 6.1.

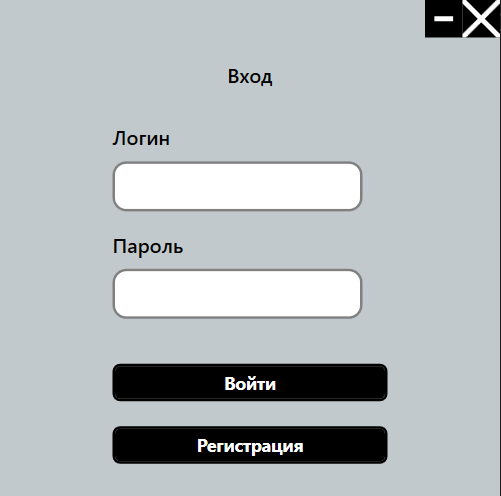


Рисунок 6.1 – Окно авторизации

Далее, если у пользователя нет аккаунта, то он нажимает на кнопку «Регистрация». Появляется форма регистрации, представленная на рисунке 6.2.

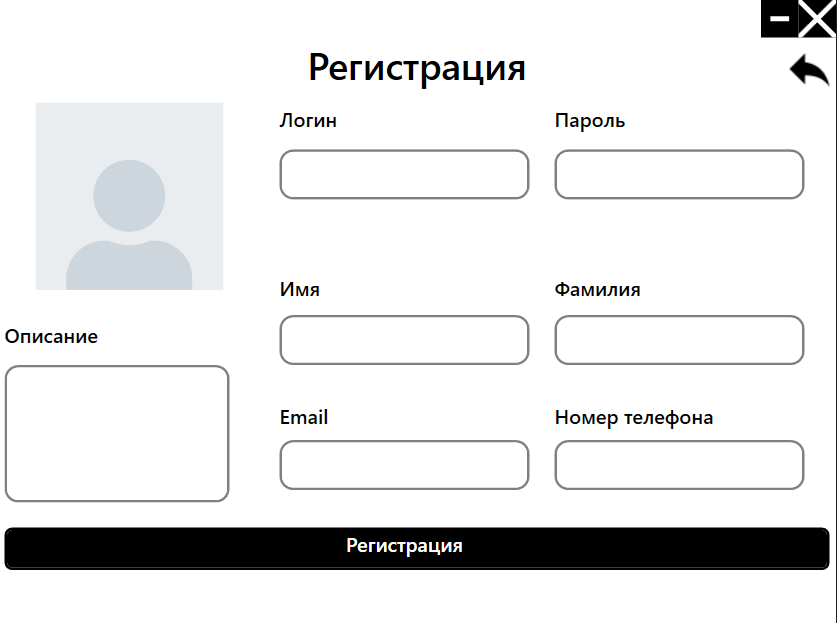


Рисунок 6.2 – Окно регистрации

После прохождения процедуры регистрации в базе данных появятся данные о пользователе и он сможет войти в систему.

После прохождения авторизации появляется главное окно приложения, в котором, в зависимости от прав, будет отличаться верхняя панель окна: у администратора доступна кнопка добавления нового товара, у пользователя – переход в корзину (Рисунок 6.3 и 6.4).



Рисунок 6.3 – Навигационная панель пользователя



Рисунок 6.4 – Навигационная панель администратора

В каталоге товаров пользователь может увидеть список всех товаров, выставить нужные фильтры поиска и нажать кнопку «Применить». Сбросить фильтры можно при нажатии кнопки «Сбросить». При вводе символов в поисковую строку, каталог будет изменяться в соответствии с введенными символами. Интерфейс представлен на рисунке 6.5.

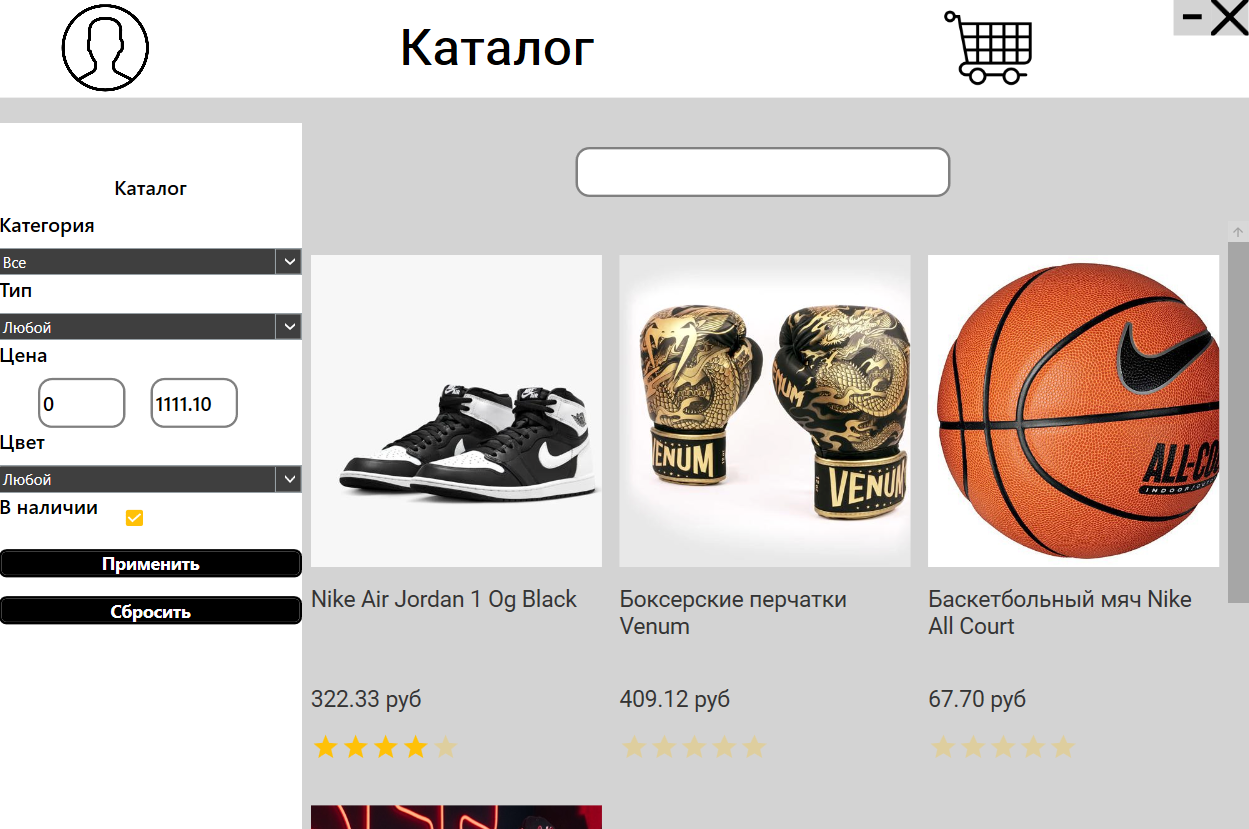


Рисунок 6.5 – Интерфейс каталога

При нажатии на нужный товар происходит переход на страницу товара, которая отличается в зависимости от прав пользователя: обычный пользователь может добавлять комментарии, а администратор вместо этого видит статистику о товаре и кнопку для удаления товара и кнопку для редактирования товара. Интерфейсы представлены на рисунках 6.6 и 6.7.

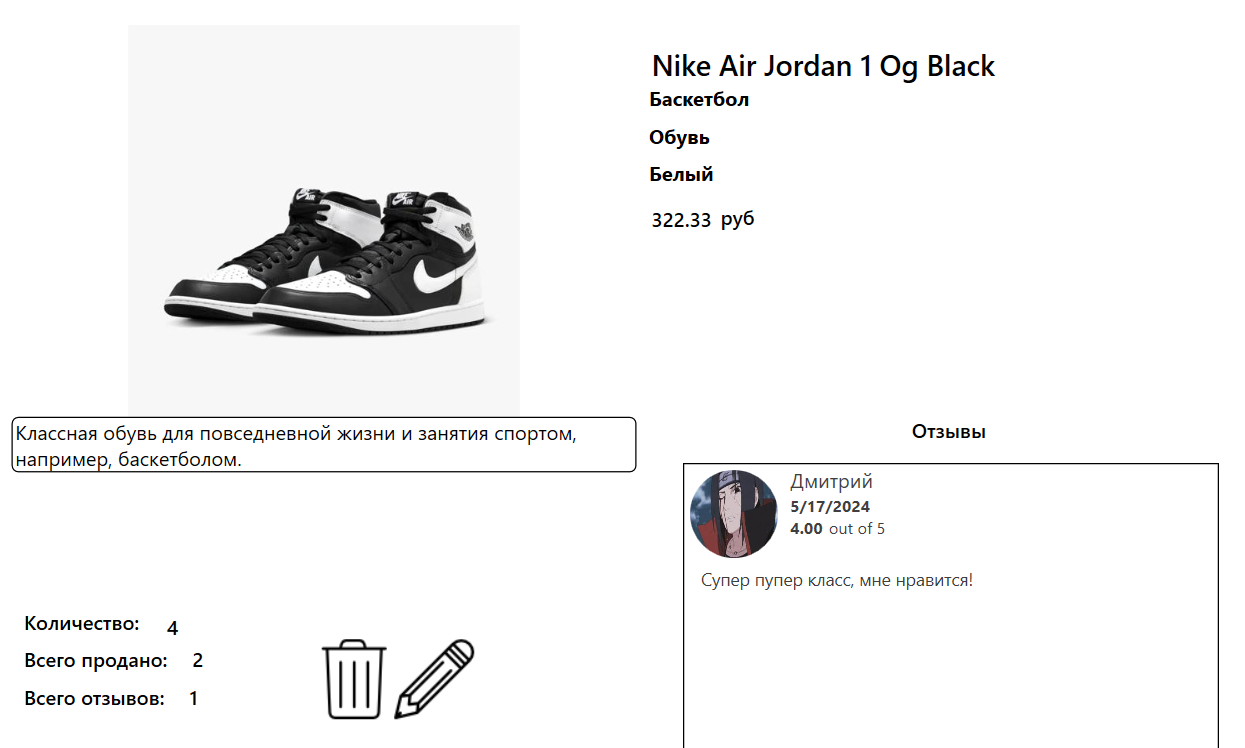


Рисунок 6.6 – Интерфейс страницы товара администратора

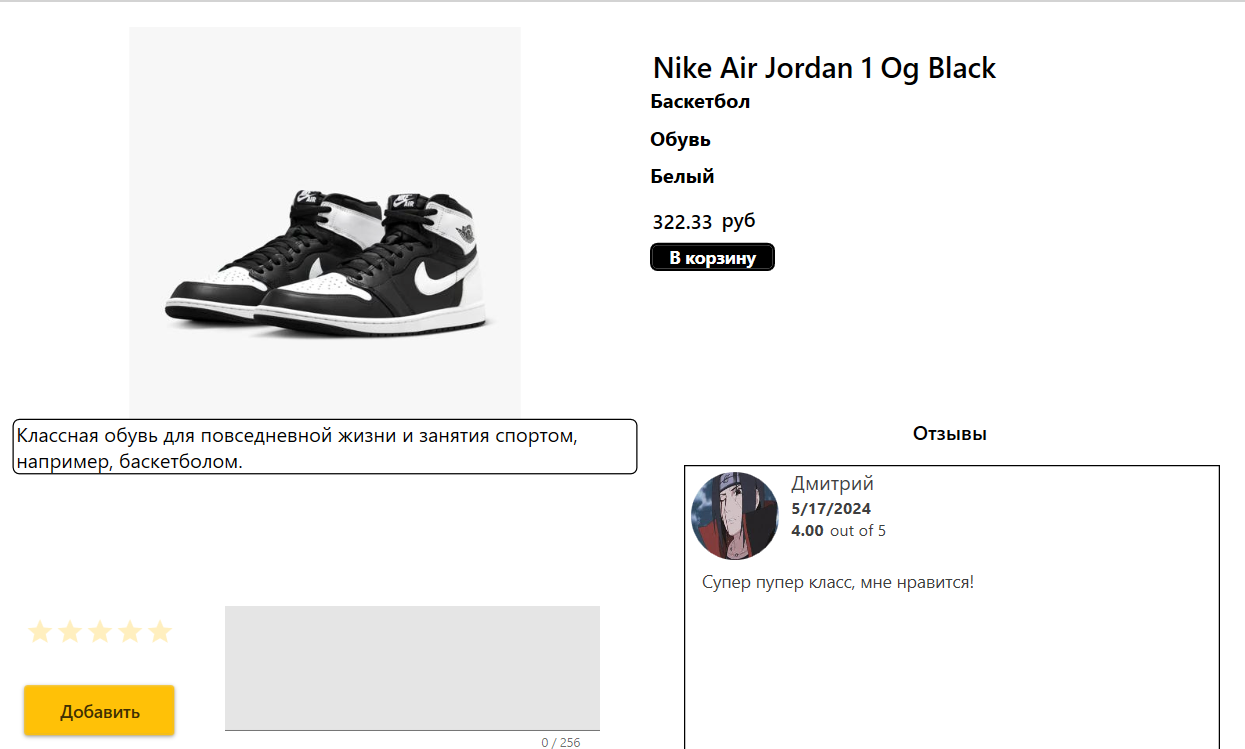


Рисунок 6.7 – Интерфейс страницы товара пользователя

При нажатии на кнопку «Добавить в корзину», товар добавляется в корзину, в которой можно поменять количество товара для заказа, заполнить форму заказа и нажать кнопку «Заказать». Интерфейс представлен на рисунке 6.8.

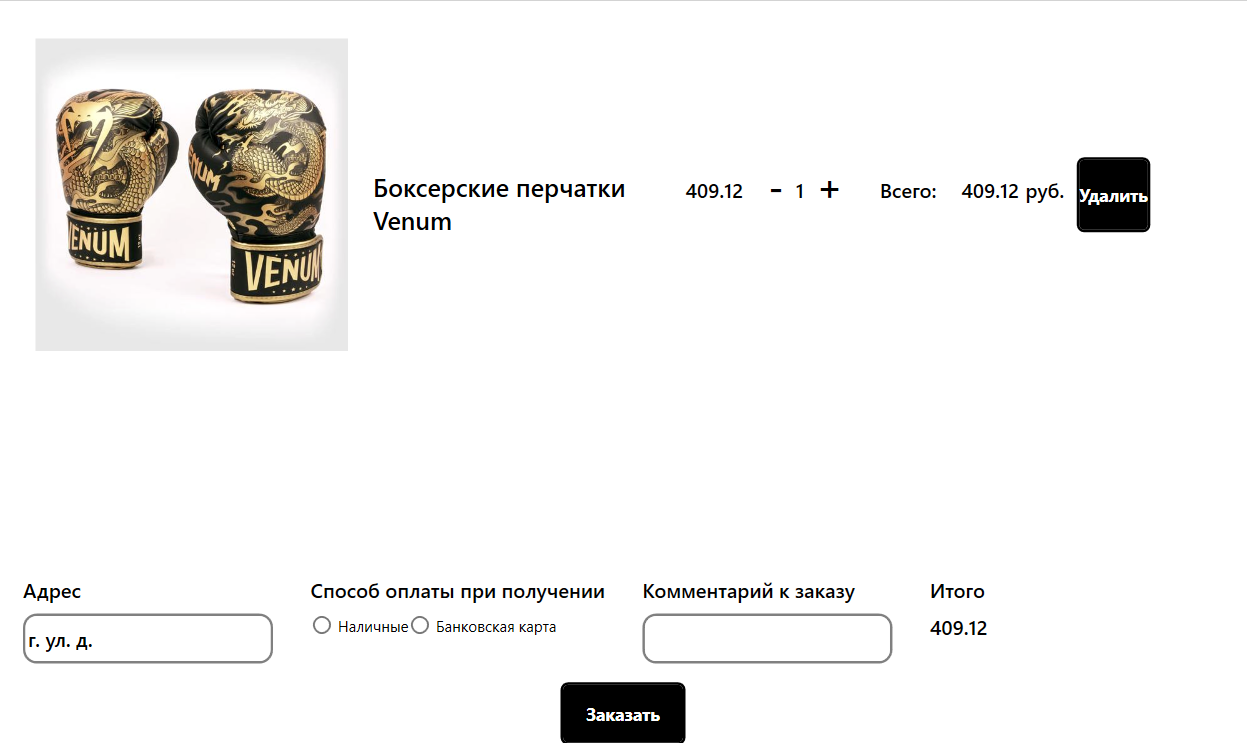


Рисунок 6.8 – Интерфейс корзины товаров

При переходе на страницу профиля, пользователь видит свою информацию. Он может поменять данные о себе, изменить пароль и нажать кнопку «Сохранить изменения». Также пользователь видит свои заказы, а администратор видит заказы всех пользователей. Интерфейс представлен на рисунке 6.9.

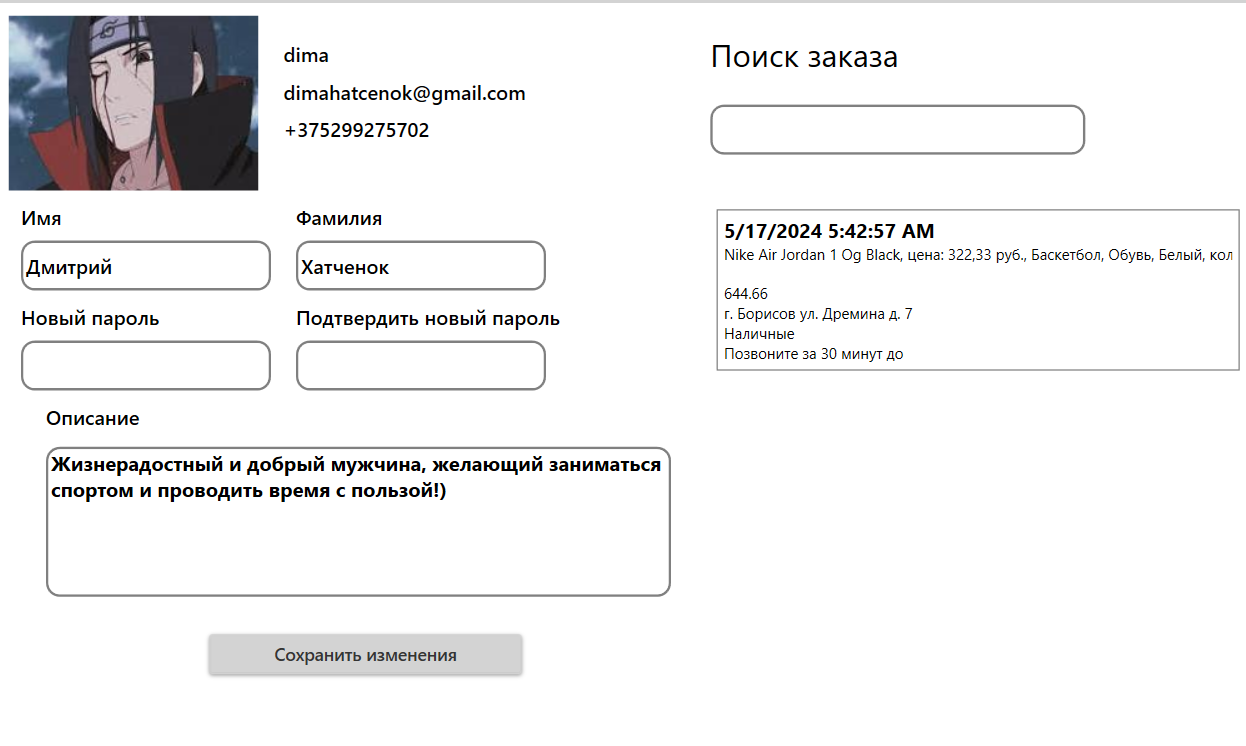


Рисунок 6.9 – Интерфейс страницы профиля

Заключение

В ходе выполнения курсового проекта было разработано программное средство «Магазин спортивных товаров» на языке C# с использованием технологий Entity Framework Core, WPF.

При разработке программного средства были выполнены все пункты из указанного списка основного функционала приложения, а именно:

* + Добавлять, удалять и изменять товар;
  + Выполнять поиск товара с использованием фильтров;
  + Просматривать товар;
  + Оставлять отзыв о товаре;
  + Заказывать товары из корзины;
  + Просматривать статистику товара от лица администратора;
  + Просматривать заказы от лица администратора;
  + Авторизация/регистрация от лица пользователя;
  + Авторизация от лица администратора.

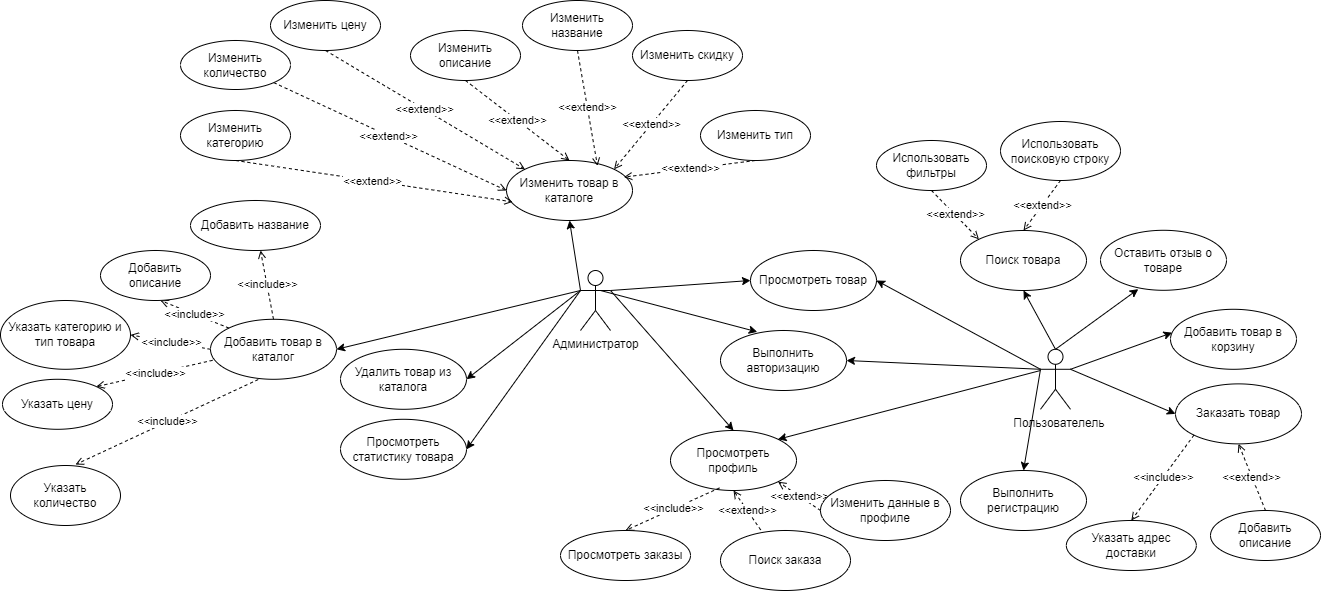
Тестирование программного средства показало, что оно работает корректно и выполняет все свои функции.

Также в процессе выполнения данного курсового проекта были закреплены навыки в программировании на языке C#, создании приложений на WPF, использование Entity Framework Core, проектирование базы данных в СУБД MS SQL Server 2019 и работа с современным паттерном MVVM.

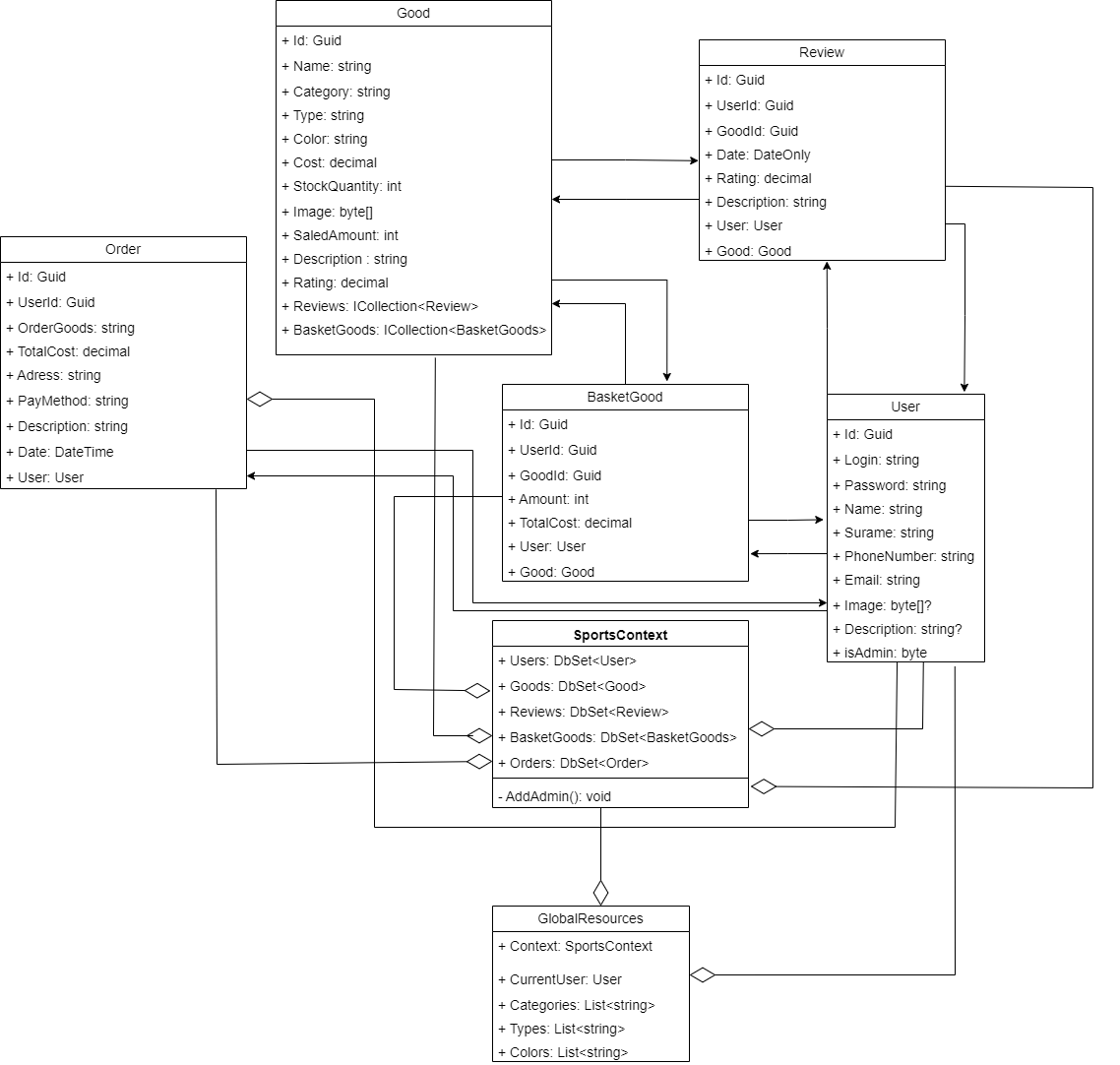
# **Список использованных источников**

1. Sportmaster [Электронный ресурс]. – Режим доступа: https://www.sportmaster.by/.
2. Nike [Электронный ресурс]. – Режим доступа: https://www.nike.com/.
3. Sport-center [Электронный ресурс]. – Режим доступа: https://sport-center.by/.
4. Microsoft Visual Studio [Электронный ресурс] – https://ru.wikipedia.org/wiki/Microsoft\_Visual\_Studio – Дата доступа 23.04.2023
5. Полное руководство по языку программирования C# 7.0 и платформе .NET 4.7. Режим доступа: https://metanit.com/sharp/tutorial/ – Дата доступа: 23.04.2023
6. Пацей, Н.В. Курс лекций по языку программирования C# / Н. В. Пацей. – Минск: БГТУ, 2018. – 175 с.
7. Руководство по WPF // [Электронный ресурс]. – Режим доступа: https://metanit.com/sharp/wpf/ – Дата доступа: 28.04.2023
8. Руководство по XAML // [Электронный ресурс]. – Режим доступа: https://www.tutorialspoint.com/xaml/index.htm– Дата доступа: 20.04.2023
9. Работа с Entity Framework Core [Электронный ресурс] – https://professorweb.ru/my/entity-framework/6/level1/ – Дата доступа 23.04.2023
10. Блинова, Е.А. Курс лекций по Базам данным / Е.А. Блинова. – Минск: БГТУ, 2019. – 175 с.

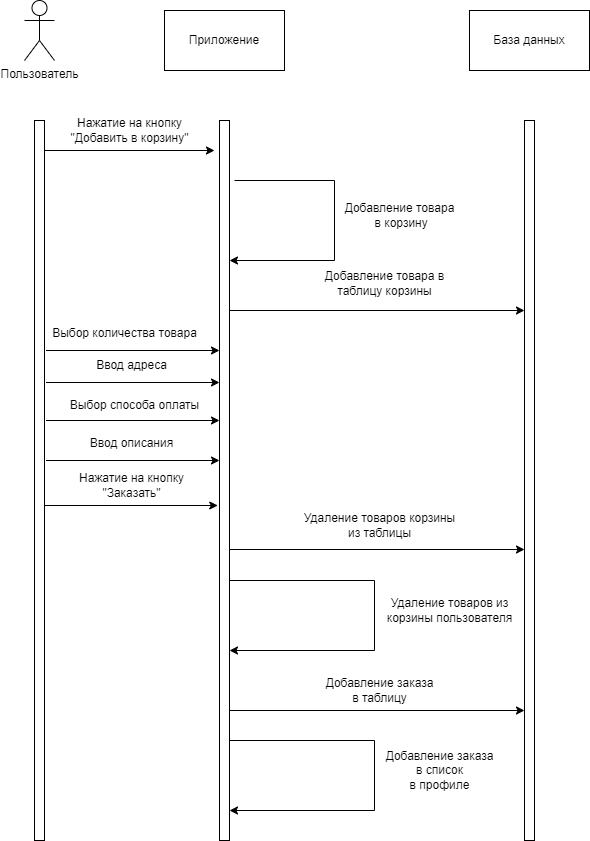
Приложение А



Приложение Б



Приложение В



Приложение Г

<UserControl x:Class="Sports\_store.Views.MainView.Basket"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:local="clr-namespace:Sports\_store.Views.MainView"

xmlns:viewmodels="clr-namespace:Sports\_store.ViewModels"

xmlns:i="http://schemas.microsoft.com/xaml/behaviors"

mc:Ignorable="d"

d:DesignHeight="700" d:DesignWidth="1000" Background="White">

<UserControl.DataContext>

<viewmodels:BasketVM/>

</UserControl.DataContext>

<Grid>

<Grid.RowDefinitions>

<RowDefinition Height="5\*"/>

<RowDefinition/>

<RowDefinition Height="60"/>

</Grid.RowDefinitions>

<ScrollViewer VerticalScrollBarVisibility="Auto">

<ItemsControl Grid.Row="0" ItemsSource="{Binding BasketGoods}" VerticalAlignment="Top" Margin="30">

<ItemsControl.ItemsPanel>

<ItemsPanelTemplate>

<UniformGrid Columns="1"/>

</ItemsPanelTemplate>

</ItemsControl.ItemsPanel>

<ItemsControl.ItemTemplate>

<DataTemplate>

<StackPanel Orientation="Horizontal" Margin="0 0 0 30">

<Image Height="250" Width="250" Source="{Binding Good.Image}" Stretch="Fill"/>

<TextBlock Margin="20 40 0 0" FontSize="20" TextWrapping="Wrap" Width="250" Height="80" Style="{StaticResource TextBlockAuth}" Text="{Binding Good.Name}"/>

<TextBlock Style="{StaticResource TextBlockAuth}" Text="{Binding Good.Cost}"/>

<TextBlock Style="{StaticResource TextBlockAuth}" FontSize="30" Margin="0 0 0 25" Text="-">

<i:Interaction.Triggers>

<i:EventTrigger EventName="MouseDown">

<i:InvokeCommandAction Command="{Binding DataContext.Minus, RelativeSource={RelativeSource AncestorType={x:Type UserControl}}}" CommandParameter="{Binding}"/>

</i:EventTrigger>

</i:Interaction.Triggers>

</TextBlock>

<TextBlock Style="{StaticResource TextBlockAuth}" Margin="10 0 10 0" Text="{Binding Amount, UpdateSourceTrigger=PropertyChanged}"/>

<TextBlock Style="{StaticResource TextBlockAuth}" FontSize="30" Margin="0 0 30 25" Text="+">

<i:Interaction.Triggers>

<i:EventTrigger EventName="MouseDown">

<i:InvokeCommandAction Command="{Binding DataContext.Plus, RelativeSource={RelativeSource AncestorType={x:Type UserControl}}}" CommandParameter="{Binding}"/>

</i:EventTrigger>

</i:Interaction.Triggers>

</TextBlock>

<TextBlock Style="{StaticResource TextBlockAuth}" Text="Всего:"/>

<TextBlock Style="{StaticResource TextBlockAuth}" Text="{Binding TotalCost, UpdateSourceTrigger=PropertyChanged}"/>

<TextBlock Style="{StaticResource TextBlockAuth}" Margin="-15 0 0 0" Text="руб."/>

<Button Style="{DynamicResource BlackWhiteButton}" Height="60" Content="Удалить" Margin="10 0 0 0"

Command="{Binding DataContext.DeleteFromBasket, RelativeSource={RelativeSource AncestorType={x:Type UserControl}}}" CommandParameter="{Binding}">

</Button>

</StackPanel>

</DataTemplate>

</ItemsControl.ItemTemplate>

</ItemsControl>

</ScrollViewer>

<StackPanel Grid.Row="1" Orientation="Horizontal">

<StackPanel Margin="20 10 10 10">

<TextBlock Style="{StaticResource TextBlockAuth}" Text="Адрес"/>

<TextBox Style="{StaticResource TextBoxAuth}" Text="{Binding CurrentOrder.Adress}"/>

</StackPanel>

<StackPanel Margin="0 10 10 10">

<TextBlock Style="{StaticResource TextBlockAuth}" Text="Способ оплаты при получении"/>

<StackPanel Orientation="Horizontal">

<RadioButton Content="Наличные" IsChecked="{Binding Cash, Mode=TwoWay}" />

<RadioButton Content="Банковская карта" IsChecked="{Binding Card, Mode=TwoWay}" />

</StackPanel>

</StackPanel>

<StackPanel Margin="0 10 10 10">

<TextBlock Style="{StaticResource TextBlockAuth}" Text="Комментарий к заказу"/>

<TextBox Style="{StaticResource TextBoxAuth}" Text="{Binding CurrentOrder.Description}"/>

</StackPanel>

<StackPanel Margin="0 10 10 10">

<TextBlock Style="{StaticResource TextBlockAuth}" Text="Итого"/>

<TextBlock Style="{StaticResource TextBlockAuth}" Text="{Binding CurrentOrder.TotalCost}"/>

</StackPanel>

</StackPanel>

<Button Grid.Row="2" Style="{StaticResource BlackWhiteButton}" Content="Заказать" Command="{Binding OrderingGoods}" Height="50" Width="100"/>

</Grid>

</UserControl>

Листинг – Xaml-код страницы Basket

Приложение Д

using GalaSoft.MvvmLight;

using GalaSoft.MvvmLight.Messaging;

using Microsoft.EntityFrameworkCore;

using Sports\_store.Command;

using Sports\_store.Models;

using System.Collections.ObjectModel;

using System.Text.RegularExpressions;

using System.Windows;

namespace Sports\_store.ViewModels

{

public class BasketVM : ViewModelBase

{

public List<string> Payment = new List<string>()

{

"Наличные",

"Банковская карта"

};

private ObservableCollection<BasketGood> basketGoods;

private Order currentOrder;

public Order CurrentOrder

{

get => currentOrder;

set

{

currentOrder = value;

RaisePropertyChanged(nameof(CurrentOrder));

}

}

public ObservableCollection<BasketGood> BasketGoods { get => basketGoods; set { basketGoods = value; RaisePropertyChanged(nameof(BasketGoods)); } }

private MyCommand deleteFromBasket;

public MyCommand DeleteFromBasket

{

get

{

return deleteFromBasket ??= new MyCommand(obj =>

{

var good = obj as BasketGood;

CurrentOrder.TotalCost -= good.TotalCost;

GlobalResources.Context.BasketGoods.RemoveRange(GlobalResources.Context.BasketGoods.Where(bg => bg.Id == good.Id));

BasketGoods.Remove(good);

GlobalResources.Context.SaveChanges();

});

}

}

private MyCommand minus;

public MyCommand Minus

{

get

{

return minus ??= new MyCommand(obj =>

{

var good = obj as BasketGood;

if (good.Amount != 1)

{

good.Amount--;

good.TotalCost = good.Good.Cost \* good.Amount;

var bdgood = GlobalResources.Context.BasketGoods.Where(bg => bg.Id == good.Id).Include(bg => bg.Good).FirstOrDefault();

bdgood.Amount--;

bdgood.TotalCost = bdgood.Amount \* bdgood.Good.Cost;

GlobalResources.Context.BasketGoods.Update(bdgood);

GlobalResources.Context.SaveChanges();

CurrentOrder.TotalCost -= good.Good.Cost;

}

});

}

}

private MyCommand plus;

public MyCommand Plus

{

get

{

return plus ??= new MyCommand(obj =>

{

var good = obj as BasketGood;

good.Amount++;

good.TotalCost = good.Good.Cost \* good.Amount;

var bdgood = GlobalResources.Context.BasketGoods.Where(bg => bg.Id == good.Id).Include(bg => bg.Good).FirstOrDefault();

bdgood.Amount++;

bdgood.TotalCost = bdgood.Amount \* bdgood.Good.Cost;

GlobalResources.Context.BasketGoods.Update(bdgood);

GlobalResources.Context.SaveChanges();

CurrentOrder.TotalCost += good.Good.Cost;

});

}

}

private bool cash;

private bool card;

public bool Cash

{

get { return cash; }

set { cash = value; if (cash) CurrentOrder.PayMethod = "Наличные"; RaisePropertyChanged(nameof(Cash)); }

}

public bool Card

{

get { return card; }

set { card = value; if (card) CurrentOrder.PayMethod = "Банковская карта"; RaisePropertyChanged(nameof(Card)); }

}

private bool IsGoodsAvailable()

{

bool AllGood = true;

foreach (var bgood in BasketGoods)

{

if (bgood.Amount > bgood.Good.StockQuantity && bgood.Good.StockQuantity != 0)

{

MessageBox.Show("Товар " + bgood.Good.Name + " можно заказать в количестве до " + bgood.Good.StockQuantity + ".\n" +

"Выберите другое количество товара");

AllGood = false;

}

else if (bgood.Good.StockQuantity == 0)

{

MessageBox.Show("Товара " + bgood.Good.Name + " в данный момент нет в наличии");

AllGood = false;

}

}

if (!Regex.IsMatch(CurrentOrder.Adress, "г[.]\\s\*([А-ЯЁ][а-яё]\*)\\s\*ул[.]\\s\*([А-ЯЁ][а-яё]\*)\\s\*д[.]\\s\*(\\d+)"))

{

AllGood = false;

MessageBox.Show("Введите корректный адрес");

}

if (CurrentOrder.PayMethod == null)

{

MessageBox.Show("Выберите способ оплаты");

}

return AllGood;

}

private MyCommand orderingGoods;

public MyCommand OrderingGoods

{

get

{

return orderingGoods ??= new MyCommand(obj =>

{

if (IsGoodsAvailable())

{

foreach (var bgood in BasketGoods)

{

var orderedGood = GlobalResources.Context.Goods.Where(g => g.Id == bgood.GoodId).FirstOrDefault();

orderedGood.StockQuantity -= bgood.Amount;

orderedGood.SaledAmount += bgood.Amount;

GlobalResources.Context.Goods.Update(orderedGood);

GlobalResources.Context.SaveChanges();

}

CurrentOrder.Date = DateTime.Now;

CurrentOrder.BasketToString(BasketGoods);

GlobalResources.Context.Orders.Add(CurrentOrder);

var deleted\_basket = GlobalResources.Context.BasketGoods.Where(bg => bg.UserId == GlobalResources.CurrentUser.Id);

GlobalResources.Context.BasketGoods.RemoveRange(deleted\_basket);

BasketGoods = new ObservableCollection<BasketGood>();

Messenger.Default.Send<Order>(CurrentOrder);

GlobalResources.Context.SaveChanges();

CurrentOrder = new Order();

Card = false;

Cash = false;

}

}, (\_) =>

{

return BasketGoods.Count() > 0;

});

}

}

public BasketVM()

{

CurrentOrder = new Order();

BasketGoods = new ObservableCollection<BasketGood>(GlobalResources.Context.BasketGoods.AsNoTracking().Where(g => g.UserId == GlobalResources.CurrentUser.Id).Include(g => g.User).Include(g => g.Good));

if (BasketGoods.Count > 0)

{

foreach (var good in BasketGoods)

{

CurrentOrder.TotalCost += good.TotalCost;

}

}

Messenger.Default.Register<Good>(this, "add to basket", good =>

{

if (!GlobalResources.Context.BasketGoods.Where(bg => (bg.UserId == GlobalResources.CurrentUser.Id && bg.GoodId == good.Id)).Any())

{

BasketGood NewBasketGood = new BasketGood(good);

BasketGoods.Add(NewBasketGood);

GlobalResources.Context.BasketGoods.Add(NewBasketGood);

GlobalResources.Context.SaveChanges();

CurrentOrder.TotalCost += NewBasketGood.TotalCost;

BasketGoods = new ObservableCollection<BasketGood>(GlobalResources.Context.BasketGoods.AsNoTracking().Where(g => g.UserId == GlobalResources.CurrentUser.Id).Include(g => g.User).Include(g => g.Good));

}

else

{

MessageBox.Show("Товар уже находится в вашей корзине!");

}

});

}

}

}

Листинг – Реализация модели представления для страницы Basket

Приложение Е

using GalaSoft.MvvmLight;

namespace Sports\_store.Models

{

public class BasketGood : ViewModelBase

{

public Guid Id { get; set; }

public Guid GoodId { get; set; }

public Guid UserId { get; set; }

public User User { get; set; }

public Good Good { get; set; }

private int amount;

public int Amount { get => amount; set { amount = value; RaisePropertyChanged(nameof(Amount)); } }

private decimal totalCost;

public decimal TotalCost { get => totalCost; set { totalCost = value; RaisePropertyChanged(nameof(TotalCost)); } }

public override string ToString()

{

return $"{this.Good.ToString()}, количество: {this.Amount}, общая стоимость: {this.TotalCost}";

}

public BasketGood(Good good)

{

Good = GlobalResources.Context.Goods.Find(good.Id);

GoodId = good.Id;

UserId = GlobalResources.CurrentUser.Id;

User = GlobalResources.Context.Users.Find(GlobalResources.CurrentUser.Id);

Amount = 1;

TotalCost = Good.Cost;

}

public BasketGood()

{

Amount = 0;

TotalCost = 0;

}

}

}

Листинг – реализация модели на примере класса BasketGood