МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ

ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет Информационных Технологий

Кафедра Программной инженерии

Специальность 6-05-0612-01 «Программная инженерия»

Направление специальности 6-05-0612-01 «Программная инженерия»

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**КУРСОВОГО ПРОЕКТА:**

по дисциплине «Объектно-ориентированное программирование»

Тема Программное средство «Магазин по продаже дизайнерской одежды»

Исполнитель

студент 2 курса группы 10 Гулецкий Прохор Олегович

(Ф.И.О.)

Руководитель работы асс. Мущук Артур Николаевич

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

Курсовой проект защищен с оценкой

Председатель Мущук А.Н.

(подпись)

Минск 2025

Оглавление

[ВВЕДЕНИЕ 6](#_Toc198487752)

[1. Аналитический обзор литературы и формирование требований 7](#_Toc198487753)

[**1.1 Анализ прототипов** 7](#_Toc198487754)

[**1.1.1.** **Balenciaga** 7](#_Toc198487755)

[**1.1.2.** **Maison Margiela** 8](#_Toc198487756)

[**1.1.3.** **Rick Owens** 8](#_Toc198487757)

[**1.2** **Требования к проекту** 9](#_Toc198487758)

[**1.3** **Выводы к разделу** 9](#_Toc198487759)

[2. Анализ требований к программному средству и разработка функциональных требований 10](#_Toc198487760)

[2.1. Описание средств разработки 10](#_Toc198487761)

[2.1.1. Microsoft Visual Studio 2022 10](#_Toc198487762)

[2.1.2. Программная платформа .NET Core 9.0 10](#_Toc198487763)

[2.1.3. Язык программирования C# 10](#_Toc198487764)

[2.1.4. Технология WPF 11](#_Toc198487765)

[2.1.5. Расширяем язык разметки XAML 11](#_Toc198487766)

[2.1.6. Технология Entity Framework Core 11](#_Toc198487767)

[2.1.7. Технология Fluent API 11](#_Toc198487768)

[2.1.8. MS SQL Server 12](#_Toc198487769)

[2.2. Спецификация функциональных требований к программному средству 12](#_Toc198487770)

[2.3. Спецификация функциональных требований 13](#_Toc198487771)

[2.4. Вывод по разделу 13](#_Toc198487772)

[3. Проектирование программного средства 14](#_Toc198487773)

[3.1. Общая структура 14](#_Toc198487774)

[3.2. Взаимотношения между классами 15](#_Toc198487775)

[3.3. Модель базы данных 15](#_Toc198487776)

[3.4. Проектирование функицонального проекта 16](#_Toc198487777)

[3.5. Проектирование последовательностей проекта 16](#_Toc198487778)

[3.6. Вывод по разделу 16](#_Toc198487779)

[4. Реализация программного средства 17](#_Toc198487780)

[4.1. Основные классы программного средства 17](#_Toc198487781)

[4.2. Основные классы программного средства 17](#_Toc198487782)

[4.2.1. Выполнение входа 17](#_Toc198487783)

[4.2.2. Управление товарами 18](#_Toc198487784)

[4.2.3. Управление корзиной 18](#_Toc198487785)

[4.2.4. Управление заказами 19](#_Toc198487786)

[4.2.5. Управление комментариями и отзывами 19](#_Toc198487787)

[4.3. Выводы по разделу 19](#_Toc198487788)

[5. Тестирование, проверка работоспособности и анализ полученных результатов 20](#_Toc198487789)

[5.1. Тестирование входа 20](#_Toc198487790)

[5.2. Тестирование управления товарами 21](#_Toc198487791)

[5.3. Тестирование управления корзиной 22](#_Toc198487792)

[5.4. Тестирование управления заказами 23](#_Toc198487793)

[5.5. Тестирование управления комментарями и отзывами 25](#_Toc198487794)[5.6. Выводы по разделу 26](#_Toc198487796)

[6. Руководство по установке и использованию 27](#_Toc198487797)

[6.1. Выводы по разделу 32](#_Toc198487800)

[**Заключение** 33](#_Toc198487801)

[Список литературы 34](#_Toc198487802)

[**ПРИЛОЖЕНИЕ А** 35](#_Toc198487803)

[**ПРИЛОЖЕНИЕ Б** 36](#_Toc198487804)

[**ПРИЛОЖЕНИЕ В**  37](#_Toc198487805)

[**ПРИЛОЖЕНИЕ Г** 38](#_Toc198487806)

# **ВВЕДЕНИЕ**

Современный рынок одежды предлагает огромное разнообразие стилей и направлений, но особое место занимает дизайнерская одежда, которая сочетает в себе уникальный дизайн, высокое качество материалов и эксклюзивность. Покупка дизайнерских вещей раньше требовала посещения бутиков или специализированных магазинов, однако с развитием цифровых технологий приобрести такие товары можно онлайн, не выходя из дома, в любое удобное время.

Программное средство «Магазин дизайнерской одежды» — это веб-приложение, которое позволяет пользователям удобно и быстро выбирать, заказывать и оплачивать эксклюзивные предметы гардероба.

Целью курсового проекта стоит разработка программного средства позволяющее пользователям:

* просмотр и поиск товара;
* применение фильтров поиска;
* оформление заказа;
* оставление отзывов и рейтингов о товаре;

так же будет реализована сторона администратора:

* добавление, удаление, редактирование пользователей;
* изменение статуса заказа;
* управление правами пользователей.
  1. **Аналитический обзор литературы и формирование требований**

1. **Анализ прототипов**

Были проанализированы цели и задачи, поставленные в данном курсовом проекте, а также рассмотрены аналогичные примеры их решений. На основании анализа всех достоинств и недостатков данных альтернативных решений были сформулированы требования к данному программному средству.

* + 1. **Balenciaga**

Сайт Balenciaga[1] гармонично сочетает минималистичный дизайн с передовыми технологическими решениями, создавая уникальный пользовательский опыт. Главная страница регулярно обновляется в соответствии с актуальными кампаниями бренда, демонстрируя крупные медиа-блоки, включая динамичные видео и анимации, привлекающие внимание посетителей.

Основное внимание уделяется новинкам и самым популярным позициям, которые сразу выводятся на первый экран, облегчая поиск трендовых товаров. Кроме того, сайт предлагает удобный и функциональный поиск по каталогу, расширенные фильтры по категориям и быстрый доступ к личному кабинету, что делает процесс покупок максимально комфортным. Интерфейс представлен на рисунке 1.1.

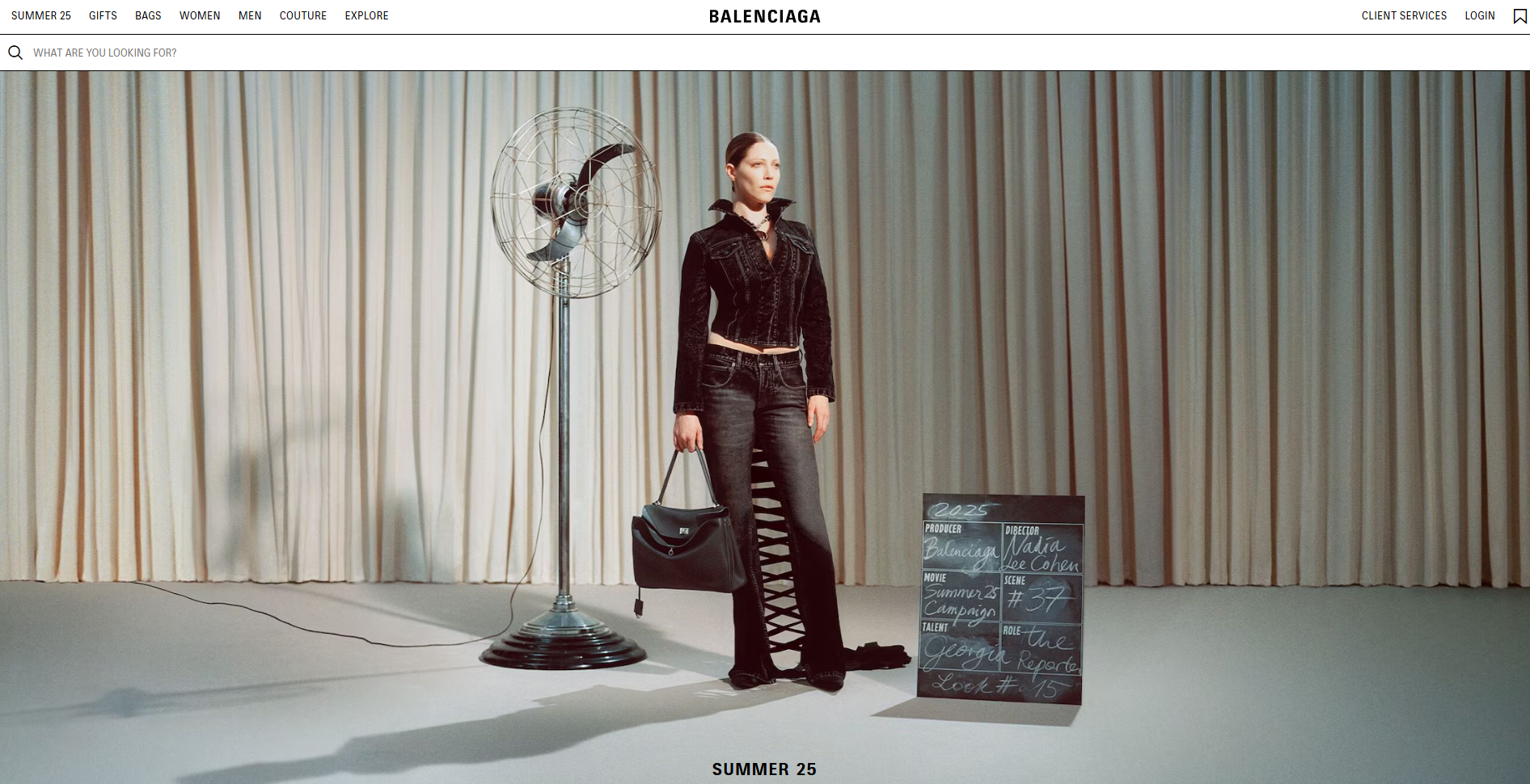


Рисунок 1.1 – Главная страница Balenciaga

Отличительной особенностью платформы является интеграция с цифровыми проектами бренда, такими как фирменные видеоигры и возможность виртуальной примерки в дополненной реальности.

* + 1. **Maison Margiela**

На сайте Maison Margiela[2] главная страница отражает авангардный подход бренда. Здесь используется симметричная компоновка элементов, что создает ощущение арт-проекта, а не просто интернет-магазина. Основной упор сделан на интерактивность: параллакс-эффекты, нестандартные hover-анимации и неочевидные переходы между разделами. Интерфейс представлен на рисунке 1.2.

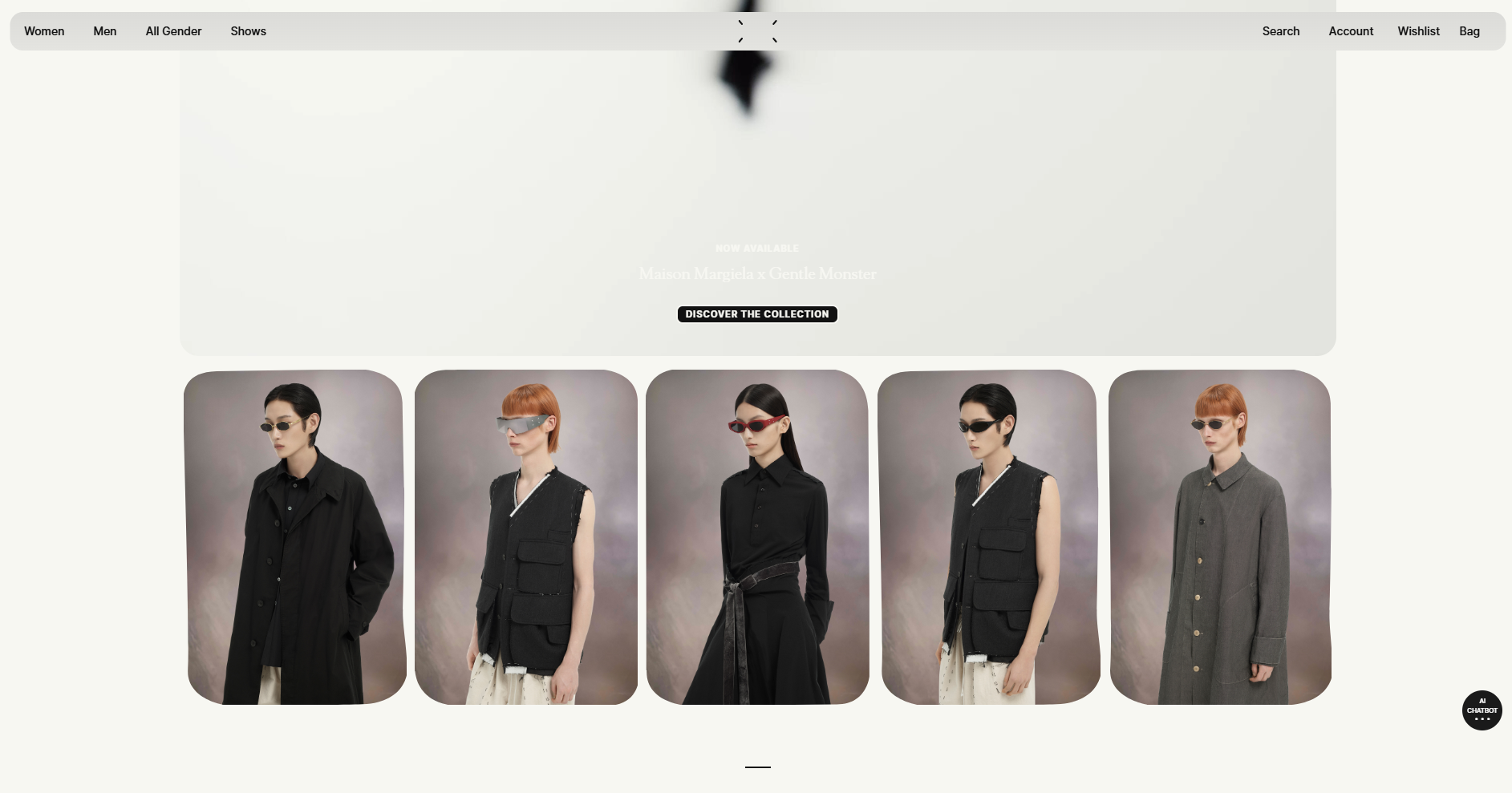


Рисунок 1.2 – Главная страница Maison Margiela

Навигация менее традиционна — вместо привычных категорий вроде "Женское" или "Мужское" используются концептуальные названия, такие как "Replica" или "Artisanal". Текстовые описания сведены к минимуму, акцент делается на визуальную составляющую.

* + 1. **Rick Owens**

На сайте Rick Owens[2] главная страница выполнена в строгом минималистичном стиле, характерном для бренда. Основной акцент сделан на крупных визуальных элементах — изображениях новых коллекций и видеоконтенте. Навигация интуитивно понятна, но при этом ненавязчива. Пользователь сразу видит ключевые разделы, такие как текущие коллекции, магазины и архив. Интерактивность реализована через плавные переходы и анимации при наведении, что добавляет сайту динамики без перегруженности. Интерфейс представлен на рисунке 1.3.

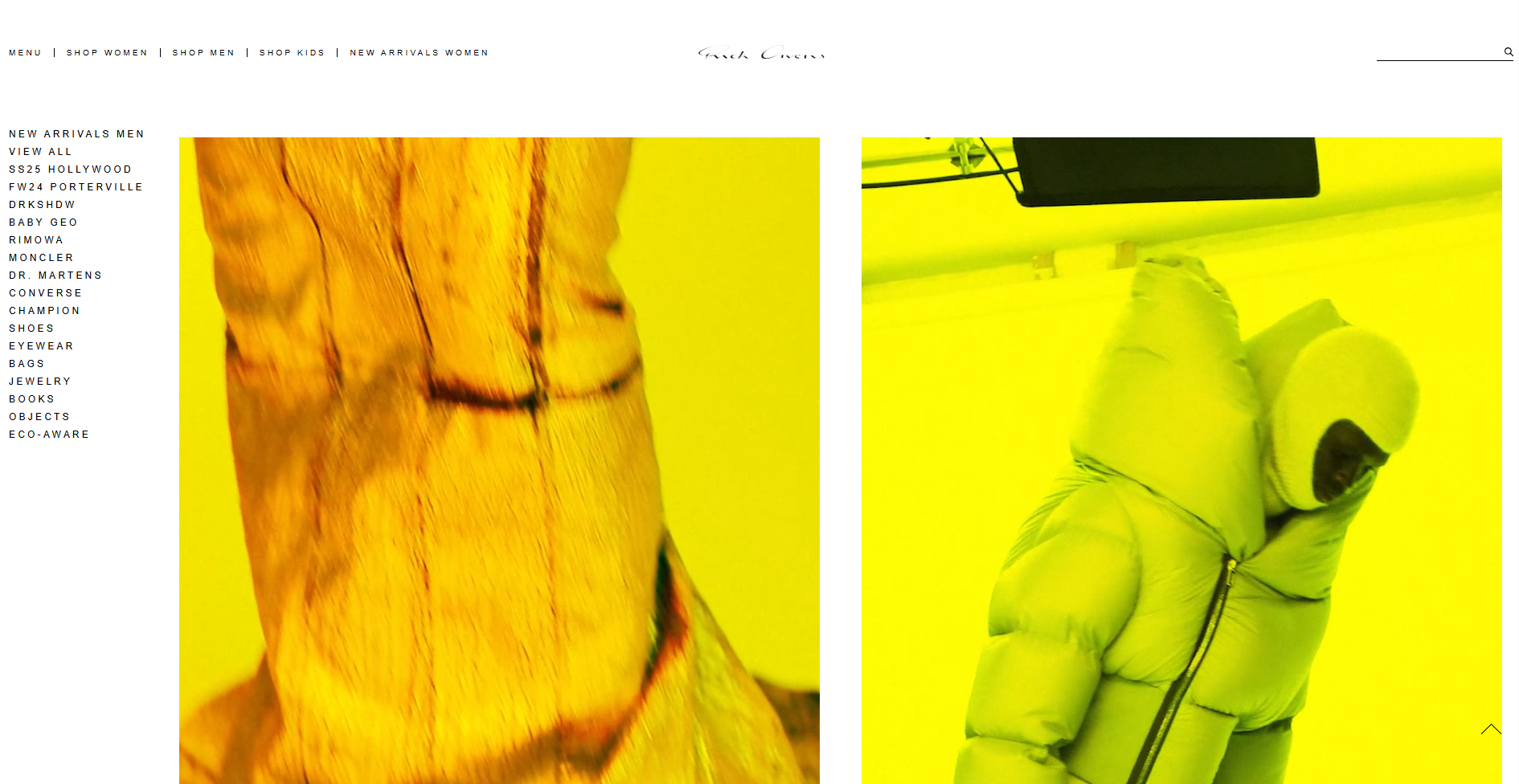


Рисунок 1.3 – Главная страница Rick Owens

Контент подается как curated selection — вместо навязчивых баннеров со скидками или категорий вроде «Новинки» и «Бестселлеры» здесь доминируют полноэкранные видео lookbook’ов, статичные кадры перформансов и крупные планы деталей одежды, снятые как фрагменты скульптур.

* 1. **Требования к проекту**

Исходя из предоставленного анализа, можно выделить следующие требования к приложению для приобритения дизайнерской одежды:

* минималистичный и концептуальный дизайн: главная страница должна отражать ДНК бренда через лаконичность, монохромную или авангардную цветовую гамму, акцент на визуальный контент.
* Быстрая навигация: пользователям должна быть предоставлена интуитивно понятная система навигации с выпадающими меню.
* Гибкая система фильтрации товаров: пользователям должна быть предоставлена возможность фильтровать товары по категориям, размерам, цветам, материалам и ценовому диапазону.
* История покупок: приложение должен предоставлять историю покупок для удобства пользователей.
* Обратная связь: возможность оставлять оценки и отзывы на товар.
  1. **Выводы к разделу**

В данном разделе были рассмотрены три сайта аналога, из них были взяты достоинства каждого аналога, так же были добавлены функции, которые аналоги не предлогали.

1. **Анализ требований к программному средству и разработка функциональных требований**

# **Описание средств разработки**

При разработке приложения были использованы:

* интегрированная среда разработки Microsoft Visual Studio 2022;
* программная платформа .NET Core 9.0;
* язык программирования C#;
* расширяемый язык разметки XAML;
* технология WPF;
* технология Entity Framework Core;
* технология Fluent API;
* база данных MS SQL Server.

# **Microsoft Visual Studio 2022**

Microsoft Visual Studio 2022 — это интегрированная среда разработки для написания, отладки и сборки кода, а также последующей публикации приложений. Данный продукт позволяет разрабатывать не только консольные, но и десктопные приложения, с использованиями таких технологий, как WinForms или WPF.

# **Программная платформа .NET Core 9.0**

Платформа .NET Core — это созданная Microsoft в 2016 году кроссплатформенная, с открытым исходным кодом технология, предназначенная для создания и выполнения современных приложений. Она поддерживает разработку различных типов приложений, включая веб-приложения, облачные сервисы, десктопные приложения и микросервисы, работая на операционных системах Windows, macOS и Linux.

Основой .NET Core является общеязыковая среда выполнения Common Language Runtime (CLR), которая обеспечивает выполнение кода, написанного на различных поддерживаемых языках программирования, таких как C#, F#, и Visual Basic. .NET Core предлагает модульную архитектуру, где разработчики могут выбирать и использовать только необходимые пакеты, что позволяет создавать более легкие и производительные приложения.

# **Язык программирования C#**

В качестве языка программирования используется C# – основной язык разработки в .NET Framework. Язык объектно-ориентированный, имеет строгую статическую типизацию, поддерживает широкий спектр возможностей, включая перегрузку операторов, указатели на функции-члены классов, использование атрибутов для расширенной настройки поведения, работу с событиями и делегатами, а также полноценную систему свойств и обработку исключений. Используется как основной язык в технологии WPF.

# **Технология WPF**

Для предоставления пользовательского интерфейса и разграничения дизайна и бизнес-логики используется технология Microsoft WPF – аналог WinForms, система для построения клиентских приложений Windows с возможностями взаимодействия с пользователем и графическая подсистема в составе .NET Framework, использующая язык разметки XAML.

# **Расширяем язык разметки XAML**

WPF предоставляет средства для создания визуального интерфейса, включая язык XAML (Extensible Application Markup Language элементы управления, привязку данных, макеты, двухмерную и трёхмерную графику, анимацию, стили, шаблоны, документы, текст, мультимедиа и оформление. XAML представляет собой язык декларативного описания интерфейса, основанный на XML.

# **Технология Entity Framework Core**

Для взаимодействия с базой данных в приложении WPF используется технология Entity Framework Core — кроссплатформенная и расширяемая версия технологии Entity Framework с открытым исходным кодом, которая активно развивается и дополняется. В то время как традиционные средства ADO.NET обеспечивают создание подключений, выполнение команд и работу с другими объектами для взаимодействия с базами данных, Entity Framework предлагает более высокий уровень абстракции. Эта технология позволяет разработчикам абстрагироваться от конкретной базы данных и работать с данными независимо от типа хранилища. Entity Framework Core является последней версией Entity Framework 7, которая совместима с .NET Core и продолжает активно развиваться.  
 Entity Framework Core предоставляет три основных способа взаимодействия с базой данных:

* database First: Entity Framework Core создает набор классов, отражающих модель существующей базы данных;
* model First: разработчик сначала создает модель данных, на основе которой Entity Framework Core затем генерирует реальную базу данных на сервере;
* code First: разработчик создает классы моделей данных, которые будут храниться в базе данных, а затем Entity Framework Core генерирует базу данных и ее таблицы на основе этих моделей.

# **Технология Fluent API**

При настройке отображения модели базы данных в Entity Framework Core могут использоваться два подхода: анотации и Fluent API. В данном проекте для реализации поставленных технических задач был выбран последний подход. Технология Entity Framework Core также способна самостоятельно сопоставить классы моделей с таблицами базы данных при использования согласовании об именовании, однако Fluent API является набором методов, предоставляющим их более тонкую настройку императивным методом.

# **MS SQL Server**

Для организации баз данных MS SQL Server использует реляционную модель, которая предполагает хранение данных в виде таблиц, каждая из которых состоит из строк и столбцов. Каждая строка хранит отдельный объект, а в столбцах размещаются атрибуты этого объекта. Для взаимодействия с базой данных применяется язык SQL (Structured Query Language). Клиент (например, внешняя программа) отправляет запрос 10 на языке SQL должным образом интерпретирует и выполняет запрос, а затем посылает клиенту результат выполнения. Основной используемый язык запросов — Transact-SQL — реализован на структурированном языке запросов (SQL) с расширениями.

# **Спецификация функциональных требований к программному средству**

Программное средство должно предоставлять следующие функциональные возможности:

для гостя:

* просматривать все товары;
* просматривать комментарии к товару;
* войти в существующий аккаунт;
* написать в администрацию;
* использовать помощь с определением размера;
* создать аккаунт;
* использовать фильтры поиска;
* использовать поиск по названию товара.

Для пользователя:

* все функции гостя;
* добавить товар в корзину;
* просматривать товары из корзины;
* удалить товар из корзины;
* просматривать последние просмотренные товары;
* добавить товар в “Избранное”;
* просматривать товары из “Избранное”
* удалить товар из “Избранное”
* просматривать текущие заказы;
* просматривать историю покупок;
* просматривать бонусы;
* оплатить товар из корзины;
* оплатить часть товара с помощью бонусов;
* оставить комментарий к товару.
* оставить рейтинг товару.

для администратора:

* добавить товар в магазин;
* изменить существующий товар;
* удалить товар из магазина;
* ограничение пользователя в правах;
* изменить статус заказа.

# **Спецификация функциональных требований**

Для функциональности ПС необходимо создание базы данных для хранения информации приложения. Подробно база данных описано в следующем разделе.

В программном средстве необходимо реализовать авторизацию пользователей для доступа ко всем возможностям приложения. Для авторизации входными параметрами являются логин и пароль пользователя, которые содержатся в базе данных. Введенные данные, успешно прошедшие валидацию, сверяются с данными в базе данных.   
 Гость может взаимодействовать с товаром с ограниченной функциональностью и использовать функционал, не требующий авторизацию (например, помощь с определением размера).

Пользователь обладает теми же возможностями, что и гость, кроме полноценной возможности взаимодействия с товаром.

Администратор может добавлять, изменять, удалять товар, изменять статус заказа, ограничить пользователя в правах.

# **Вывод по разделу**

В разделе были определены инструменты, которые будут использоваться в программном средстве. К таким относятся:

* программная платформа .NET Core 8.0;
* язык программирования C#;
* расширяемый язык разметки XAML;
* технология WPF;
* технология Entity Framework Core;
* технология Fluent API;
* база данных MS SQL Server.

1. **Проектирование программного средства**

# **Общая структура**

Программное средство использует архитектуру MVVM. Программное средство имеет следующую структуру, представленную на рисунке 3.1.

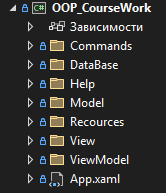


Рисунок 3.1 – Структура проекта

Описание структуры основных папок и файлов проекта представлено в таблице 3.1.

|  |  |
| --- | --- |
| Имя файла | Содержание |
| App.xaml | Файл с общими ресурсами проекта |
| Папка Recources | Папка, содержащая все изображения, шрифты и словари ресурсов. |
| Папка Command | Папка, содержащая класс, реализующий интерфейс ICommand. |
| Папка Help | Папка, содержащая классы, необходимые в работе других классов. |
| Папка DataBase | Папка, содержащая класс Context, реализацию паттернов Repository и UnitOfWork. |
| Папка Model | Папка, содержащая классы, которые являются сущностями базы данных. |
| Папка View | Папка, содержащая файлы разметок и классы соответствующих страниц. |
| Папка ViewModel | Папка, содержащая классы являющимися контекстами для страниц. |

Таблица 3.1 – Описание структуры папок и файлов проекта

# **Взаимотношения между классами**

Для визуализации взаимосвязей между классами используется диаграмма UML – графическое представление набора элементов, изображаемое чаще всего в виде связанного графа с вершинами (сущностями) и ребрами (отношениями).

Для представления внутренней структуры программы в виде классов и связей между ними используется диаграмма классов. Приложение спроектировано таким образом, что каждый класс выполняет свои функции и практически не зависит от других. Диаграмма классов представлена в Приложении А.

# **Модель базы данных**

Для реализации поставленной задачи была создана база данных OOP\_CourseWork. Для ее создания использовалась система управления реляционными базами данных MSQL Server. База данных состоит из таблиц, представленных на рисунке 3.3.

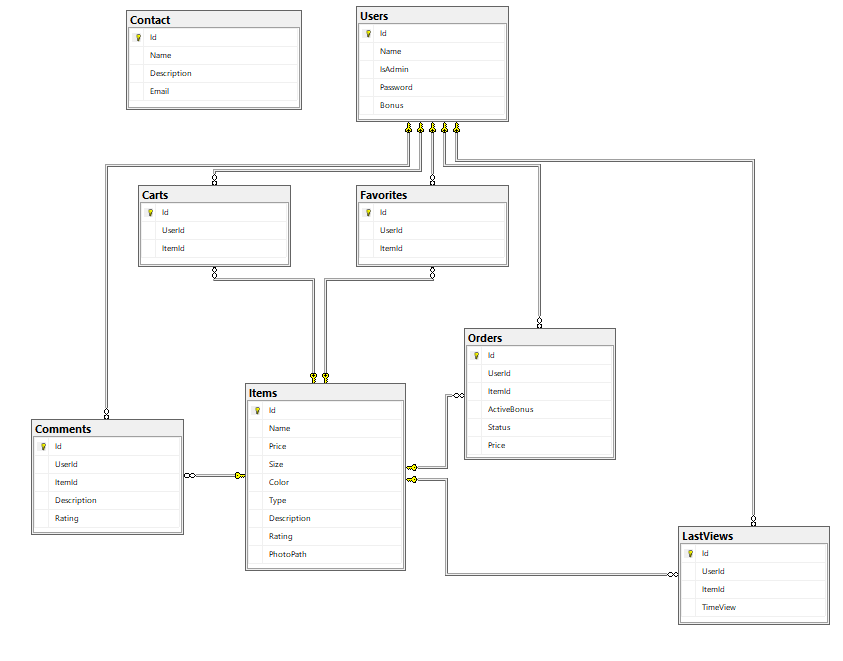


Рисунок 3.3 – Логическая модель базы данных OOP\_CourseWork

В базе данных находится 8 таблиц: Carts, Comments, Contact, Favorites, Items, LastViews, Orders, Users. Краткое пояснение, за что отвечают таблицы:

* таблица Contact содержит данные отправителя с описанием его проблемы;
* таблица Users содержит информация о пользователе;
* таблица Carts содержит предметы в корзине конкретного пользователя;
* таблица Favorites содержит предметы в “Избранное” конкретного пользователя;
* таблица Comments содержит комментарии к каждому товару каждого пользователя;
* таблица Items содержит информацию о товаре;
* таблица Orders содержит заказы каждого пользователя;
* таблица LastViews содержит последние просмотренные товары каждого пользователя.

# **Проектирование функицонального проекта**

Для общего представления функционального назначения системы используется диаграмма использования, описывающая, какой функционал разрабатываемой программной системы доступен каждой группе пользователей. На диаграмме использования применяются два типа основных сущностей: варианты использования и группы пользователей.

Разные группы пользователей в диаграмме называются актёрами, и обозначают любые сущности, использующие систему. Любая функция системы называется вариантом использования. Каждый вариант использования обозначает набор действий, который может быть использован актёром для взаимодействии с системой, и определяет набор действий, выполняемых этой системой.

Диаграмма использования представлена в приложении Б.

# **Проектирование последовательностей проекта**

В целях визуализации взаимодействия объектов системы между собой во времени в едином сценарии использования используется ещё одна UML-диаграмма – диаграмма последовательностей. Данная диаграмма иллюстрирует, как различные части системы взаимодействуют друг с другом для выполнения функции, а также порядок, в котором происходит взаимодействие при выполнении конкретного случая использования.

Для отображения течения времени используется линия жизни объекта, которая изображается с помощью штриховой линии, которая проводится вертикально вниз. С помощью линии жизни показывается период, в течение которого объект существует в системе. Сами объекты изображаются в виде прямоугольников, а сообщения, которыми они обмениваются – в виде линий со стрелками. Диаграмма последовательностей представлена в приложении В.

# **Вывод по разделу**

В разделе была выбрана архитектура проекта, которая задаёт общий план проектирование продукта. Также для программного средства были спроектированы сущности базы данных, на основе которых получены их аналоги в языке программирования C#.

1. **Реализация программного средства**

# **Основные классы программного средства**

Для выполнения технических задач программного средства должны быть реализованы следующие функции и соответствующие им классы и методы:

* выполнение входа;
* управление товарами;
* управление корзиной;
* управление заказами;
* управление комментариями и отзывами.

Далее подробно рассмотрены каждые из необходимых для выполнения технических задач функции, а также созданные для их выполнения классы и методы, и их функционал, и реализация.

# **Основные классы программного средства**

# **Выполнение входа**

Выполнение входа осуществляется в отдельном окне – SignWindow. В своём контексте данное окно использует ViewModel авторизации SignViewModel, которая взаимодействует с Unit of Work для проверки учетных данных. Процесс авторизации характеризуется следующим:

* SignViewModel реализован как зависимый от IUnitOfWork, что позволяет работать с базой данных;
* основные свойства: Username - хранит введенное имя пользователя. Password - хранит введенный пароль;
* основные методы: CanSignIn() - проверяет, что оба поля (логин и пароль) заполнены. SignInAsync() - выполняет асинхронную попытку авторизации;

Процесс авторизации:

* при вводе данных вызывается команда SignInCommand;
* происходит поиск пользователя в базе данных по имени;
* проверяется соответствие хэша введенного пароля с хранимым хэшем;
* для администратора (логин "Admin", пароль "Admin") открывается AdminWindow;
* для обычных пользователей устанавливается текущий пользователь через CurrentUser.

Особенности реализации:

* используется паттерн Unit of Work для работы с БД;
* применяется хэширование паролей через вспомогательный класс HashHelper;
* реализована асинхронная модель выполнения запросов;
* отдельная обработка администраторского входа;
* cостояние текущего пользователя управляется через CurrentUser.

Код класса SignViewModel представлен в приложении Г.

# **Управление товарами**

Управление товарами осуществляется на административной странице, реализованной через класс AdminViewModel. Эта страница позволяет администратору просматривать, добавлять, редактировать и удалять товары. В конструкторе AdminViewModel инициализируются команды управления товарами, пользователями и заказами, а также происходит начальная загрузка данных.  
Метод LoadDataAsync загружает список товаров, пользователей и заказов из базы данных с помощью репозитория IUnitOfWork. Методы AddItemAsync, UpdateItemAsync и DeleteItemAsync реализуют добавление, обновление и удаление товара соответственно. При выполнении каждой операции происходит обновление отображаемой коллекции товаров Items.  
В случае отсутствия выбранного элемента выводится сообщение об ошибке. Все изменения сохраняются через метод CompleteAsync. Листинг класса AdminViewModel представлен в приложении Г.

Также отображение и фильтрация всех товаров реализуется на клиентской части магазина через класс ShopMainViewModel.  
Данный ViewModel управляет представлением всех доступных товаров и поддерживает различные фильтры:

* по категории (SelectedType);
* ценовому диапазону (MinPrice, MaxPrice);
* названию (SearchText);
* размеру (SelectedSize);
* цвету (SelectedColor).

Список всех товаров хранится в коллекции Products, а отображаемый набор после фильтрации — в FilteredProducts.

Методы LoadProductsAsync и ApplyFilters загружают товары из базы данных и применяют фильтрацию на клиенте. Также присутствует команда AddToCartCommand, позволяющая пользователю добавить товар в корзину. Листинг класса ShopMainViewModel представлен в приложении Г.

# **Управление корзиной**

Управление корзиной реализуется в классе ShopItemViewModel с помощью метода AddToCartAsync. Пользователь может добавить выбранный товар в корзину, если он авторизован. В случае, если товар уже находится в корзине, пользователю выводится соответствующее сообщение. Добавление производится через репозиторий Carts, после чего изменения сохраняются методом CompleteAsync.

Перед добавлением проверяется авторизация пользователя (CurrentUser.UserId) и наличие товара в корзине. Также предусмотрены сообщения об ошибке в случае нарушения условий добавления.

Дополнительно управление содержимым корзины осуществляется в классе CartViewModel, который отображает все товары, добавленные пользователем в корзину. Листинги CartViewModel и ShopItemViewModel представлены в приложении Г.

# **Управление заказами**

Администратор может управлять заказами на странице администрирования через класс AdminViewModel. Коллекция заказов Orders отображает список всех заказов, полученных через репозиторий.

Изменение статуса заказа реализовано в методе ChangeOrderStatusAsync. После изменения происходит обновление данных в базе и интерфейсе. Если заказ не выбран, отображается предупреждение.

Дополнительно клиентская часть оформления и отслеживания заказов реализована в классах PaymentViewModel и OrderViewModel. Листинг данных классов представлен в приложении Г.

PaymentViewModel отвечает за этап оформления заказа, включая использование бонусных баллов, расчёт итоговой стоимости и подтверждение покупки.

OrderViewModel предназначен для отображения всех заказов, сделанных текущим пользователем. Он загружает список заказов из базы данных, позволяет просматривать подробности заказа, его статус и состав.

# **Управление комментариями и отзывами**

Функционал добавления и отображения комментариев реализован в классе ShopItemViewModel. Пользователь может оставить отзыв с текстом и рейтингом, используя свойства NewCommentText и NewCommentRating.

Метод AddCommentAsync обрабатывает добавление нового комментария, который сохраняется через репозиторий Comments. Перед добавлением выполняется проверка авторизации пользователя. Метод LoadCommentAsync загружает все отзывы для выбранного товара и обновляет коллекцию Comments.

После добавления комментария можно вызвать метод UpdateItemRatingAsync, который пересчитывает средний рейтинг товара на основе всех отзывов. Все изменения сохраняются через CompleteAsync.

# **Выводы по разделу**

В разделе описан процесс успешной реализации программного средства спроектированного в предыдущий разделах. Были описаны классы и методы этих классов, которые задавали работу программного средства.

1. **Тестирование, проверка работоспособности и анализ полученных результатов**

# **Тестирование входа**

При авторизации программное средство учитывает многие факторы. Одним из таких является проверка корректности заполнения полей, так, например поля логина и пароля не могут быть пустыми. Пример попытки ввода некорректных данных представлен на рисунках 5.1 и 5.2.

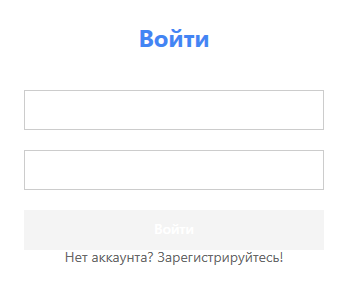


Рисунок 5.1 – Обработка пустых полей

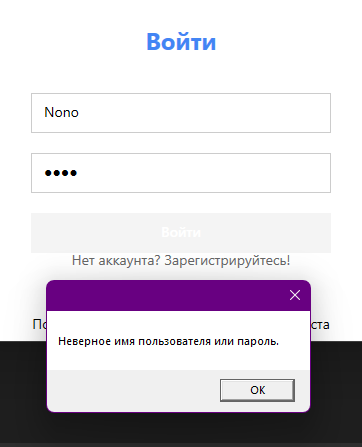


Рисунок 5.2 – Некоректный ввод данных

Таким образом программное средство подсказывает пользователю, что ему необходимо изменить чтобы успешно войти в систему.

# **Тестирование управления товарами**

Одной из основных задач администратора является работа с товаром, добавление, удаление и редактирование. Примеры работы с товарами представлены на рисунках 5.3, 5.4, 5.5 и 5.6.

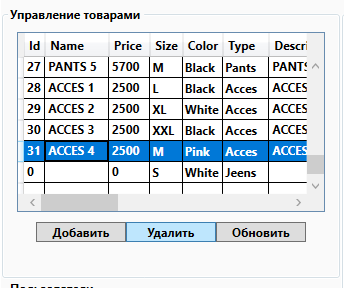


Рисунок 5.3 – Удаление товара

Администратор выделил товар, после нажатия выделенной кнопки, товар будет удален.

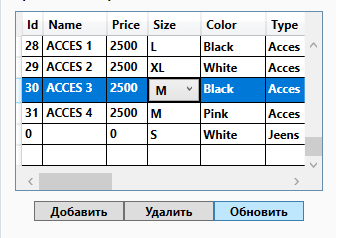


Рисунок 5.4 – Обновление товара

Администратор изменил размер предмета с XXL на M. После нажатия выделенной кнопки размер товара будет обновлен.

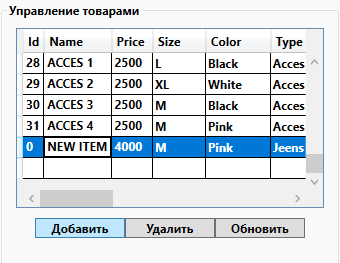


Рисунок 5.5 – Добавление товара

Администратор записал все нужные данные для товара, кроме Id, он дается автоматически, после нажатия кнопки товар будет добавлен.

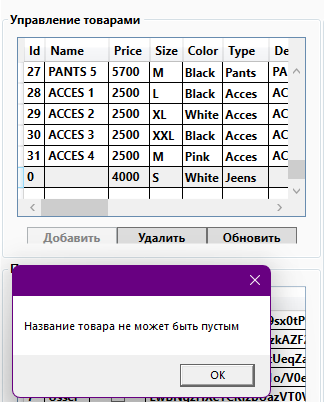


Рисунок 5.6 – Пример некорректной работы

Администратор не ввел данные в поля “Name”, программа отреагировала и выдала соотвестующие исключение.

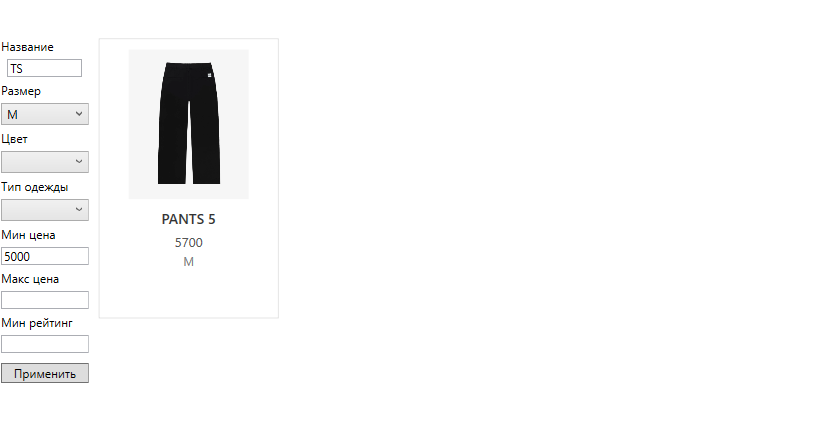


Рисунок 5.6 – Пример работы фильтров поиска

В данном примере пользователей воспользовался фильтрами поиска, где были учтены все введеные параметры.

# **Тестирование управления корзиной**

Пользователь может добавить товар в корзину, только если он авторизован. Негативный пример приведен на рисунке 5.7.

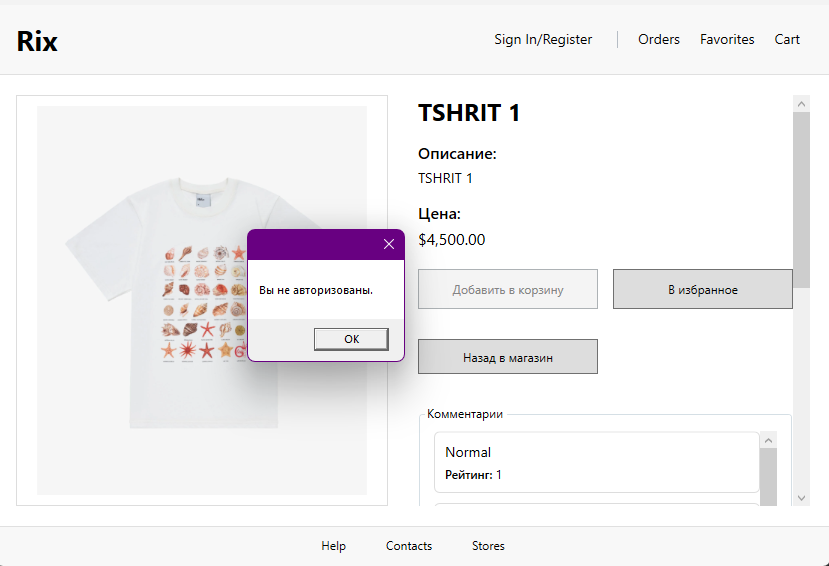


Рисунок 5.7 – Пример негативной работы добавления в корзину

Так же пользователь может просматривать свою текущую корзину. Пример приведен на рисунке 5.8.

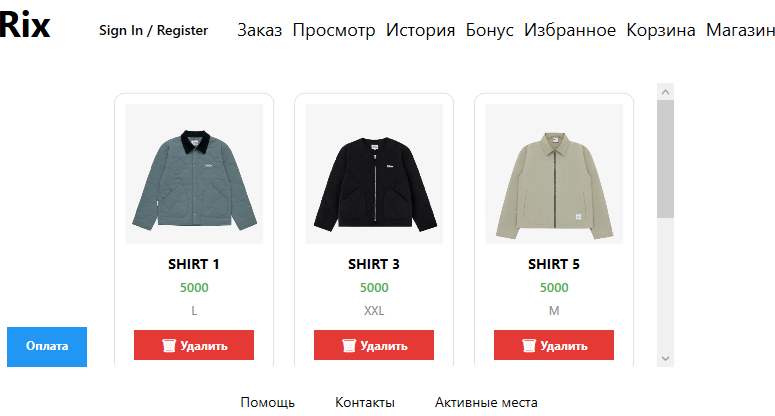


Рисунок 5.8 – Пример отображения корзины

Так же можно удалять предметы с корзины, если нажать на соотвественнуюю кнопку.

# **Тестирование управления заказами**

Так же в обязанность админа входит изменения статуса заказа, пример работы представлен на рисунке 5.9.

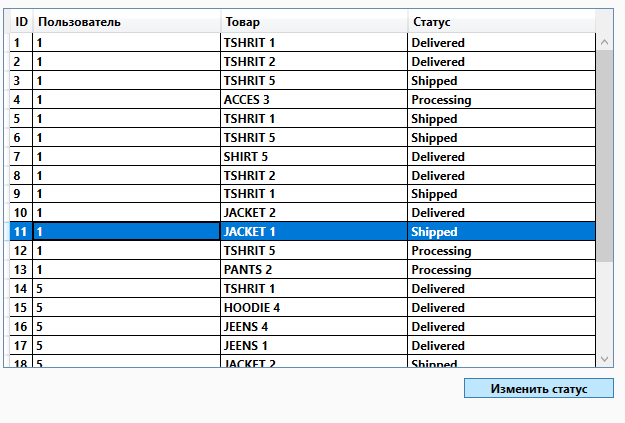


Рисунок 5.9 – Пример изменения статуса заказа

Если администратор введет не корректный статус, программа отреагирует на это, пример приведен на рисунке 5.10.



Рисунок 5.9 – Пример негативного изменения статуса заказа

Так же пользователь может просматривать свои текущие заказы c текущим статусом и историю покупок, пример приведен на рисунках 5.11 и 5.12.

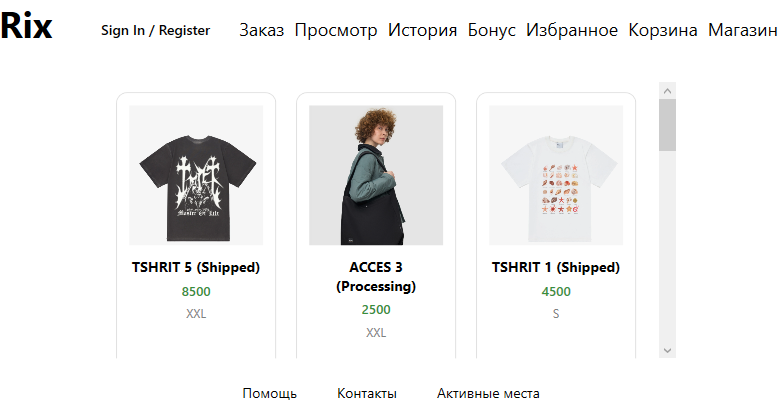


Рисунок 5.11 – Пример заказов пользователя

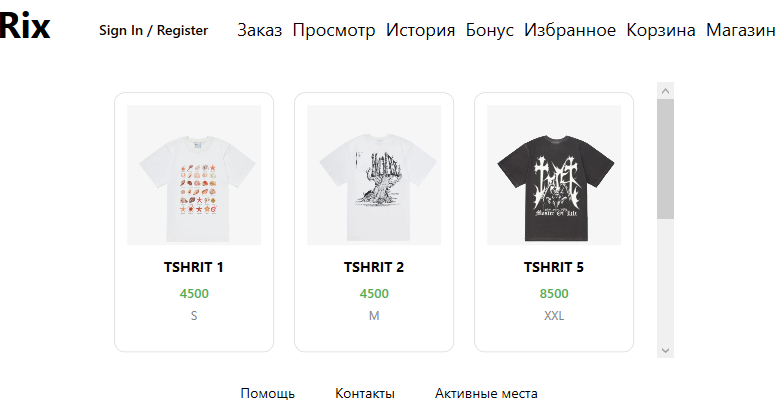


Рисунок 5.11 – Пример истории покупок пользователя

Так же пользователь может оплатить свою текущую корзину, пример приведен на рисунке 5.12 и 5.13.



Рисунок 5.12 – Пример оплаты без бонусов

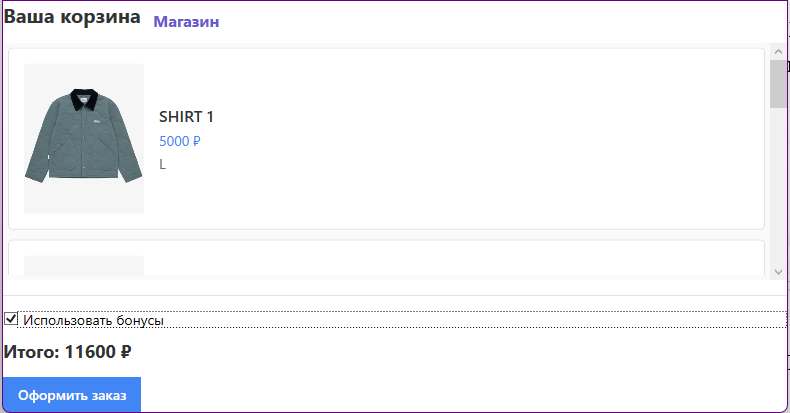


Рисунок 5.13 – Пример оплаты c бонусами

Так же, как видно из рисунков выше, пользователь может воспользоваться бонусной системой оплаты.

# **Тестирование управления комментарями и отзывами**

Важно, чтобы пользователи могли оценивать товар, представленный в магазине, что бы иные пользователи могли получше узнать о товаре, поэтому в приложении реализовано система комментариев и рейтинга. Любой зарегестрированный пользователь может оставить комментарий с рейтингом. Примеры приведены на рисунках 5.14, 5.15 и 5.16.

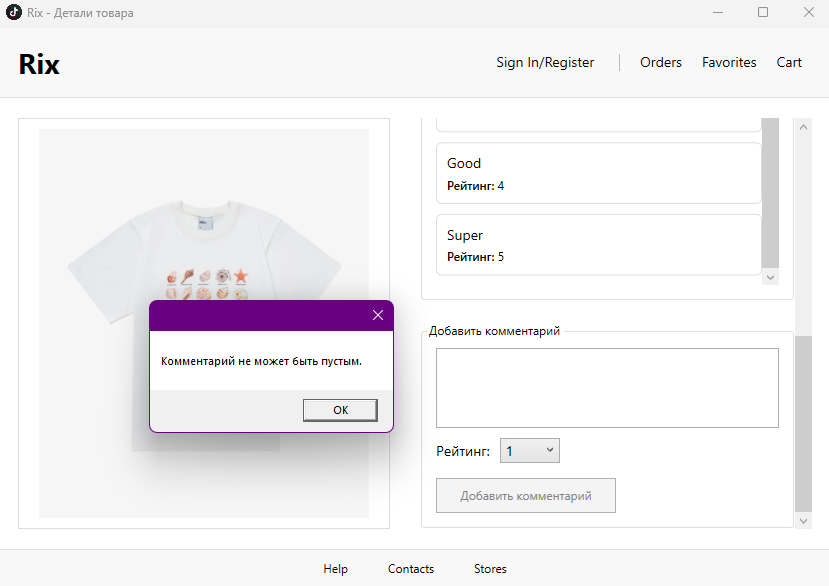


Рисунок 5.14 – Пример негативного результата добавления комментария

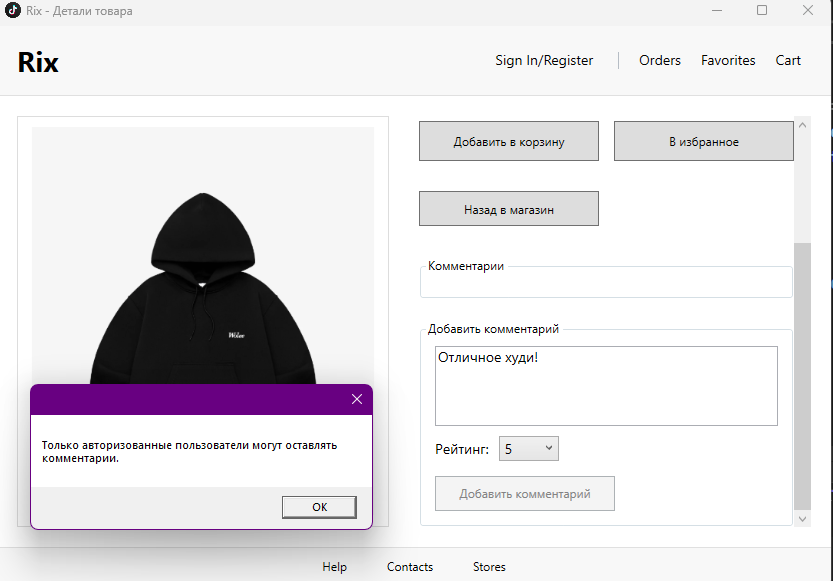


Рисунок 5.15 – Пример негативного результата добавления комментария

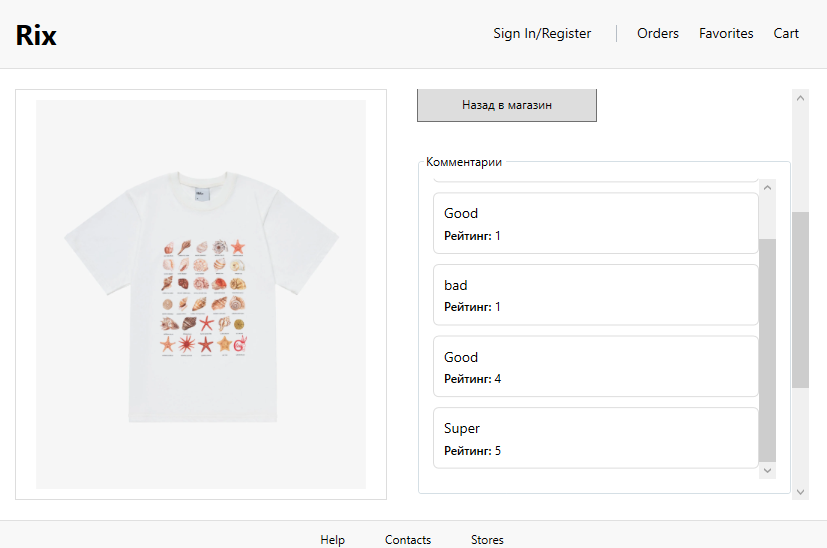


Рисунок 5.16 – Пример комментариев

На рисунке 5.16 был приведен пример отображения комментариев к товару, демонстрируя, каким образом пользовательские отзывы могут быть визуализированы внутри интерфейса.

# **Выводы по разделу**

В разделе описан процесс тестирования продукта. Были проверены случаи негативного и положительного тестирования. Были проверены все случаи валидации вводимых пользователем данных.

1. **Руководство по установке и использованию**

При запуске программного средства пользоавтеля встречает главная страница, из которой он перейти в пункт “sign in/ register” зарегистрироваться или войти в аккаунт. Пример приведен на рисунке 6.1.

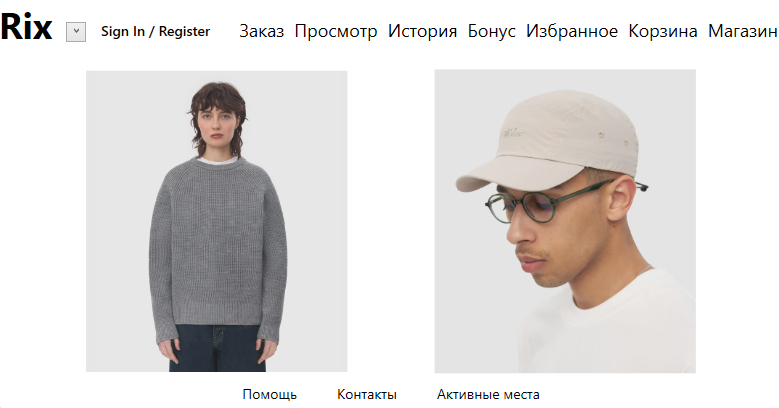


Рисунок 5.16 – Главная страница приложения

При переходе нас встречает форма для входа в аккаунт, где ниже указана строчка для перехода на форму регистрации, окна показаны на рисунке 6.2 и 6.3.

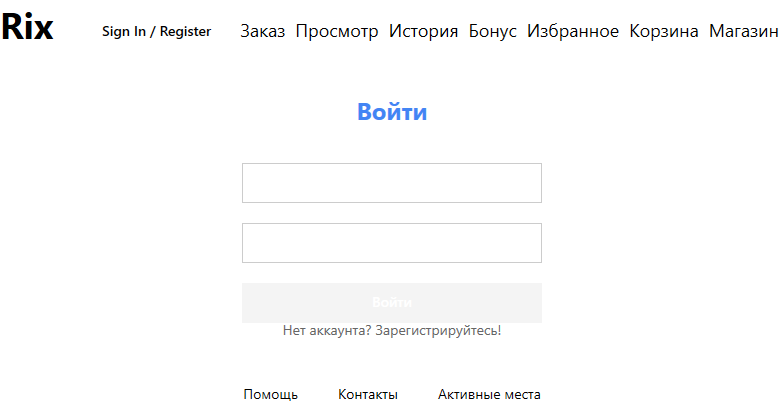


Рисунок 6.2 – Форма авторизации

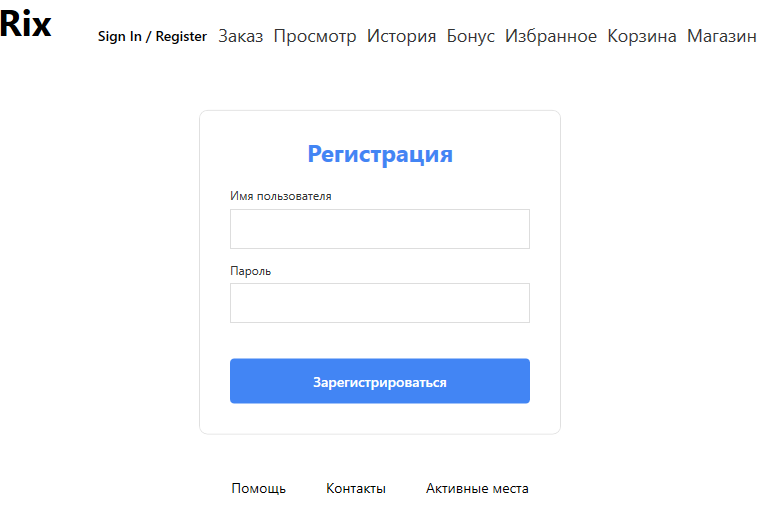
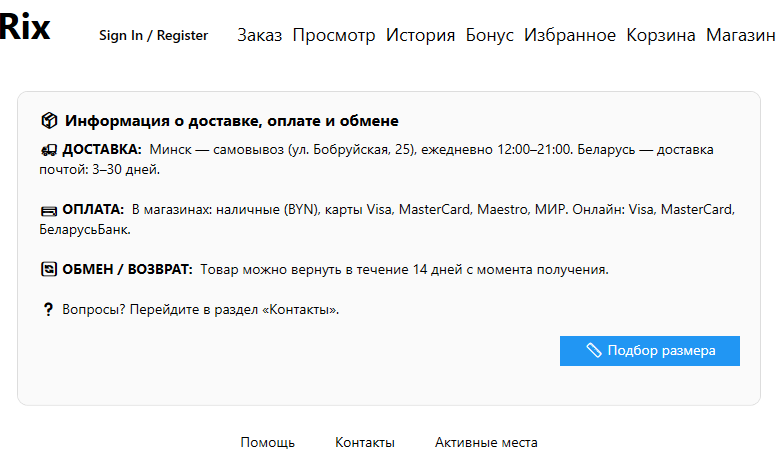
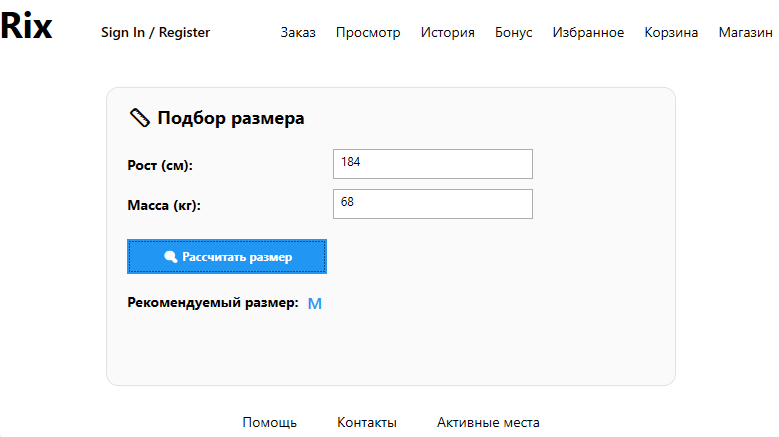


Рисунок 6.3 – Форма регистрации

После регистрации требуется повторить данные в форму авторизации для входа в аккаунт. После входа пользователь будет опевещен сообщением, об успешном входе в аккаунт, после этого пользователю доступны все функции, кроме администраторских.

Далее мы пройдемся по всем функциям, доступные пользователям. Пользователь может перейти в пункт “помощь”, где указаны самые часто задоваемаые вопросы, а также ссылка на форму, где можно определить, какой размер одежды подходит под определенные параметры, пример приведен на 6.4 и 6.5.

  
Рисунок 6.4 – Страница “Помощь”

  
 Рисунок 6.5 – Страница для определения размера

Далее пользователь может перейти в пункт “Магазин”. Это основная страница, где представлен весь товар и фильтры для облегченного поиска. Нажав на товар, можно перейти на страничку товара, где можно будет добавить товар или в корзину, или в избранное, а также просмотреть комментарии к данному товару. Пример страниц приведен на рисунках 6.6 и 6.7.



Рисунок 6.6 – Страница основного каталога товаров

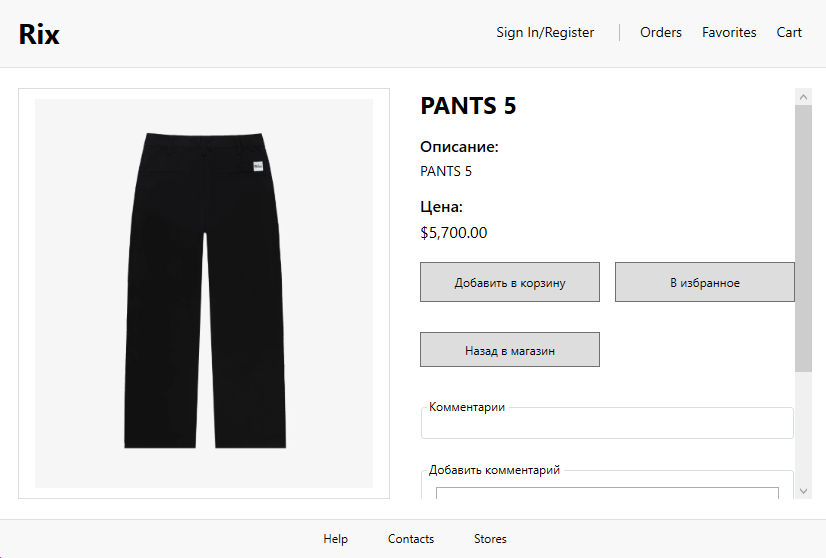


Рисунок 6.7 – Страница товара

После просмотра товара, он будет отображен в окне “Просмотр”, где хранится последние 5 просмотренных вещей пользователем. Пример приведен на рисунке 6.8.

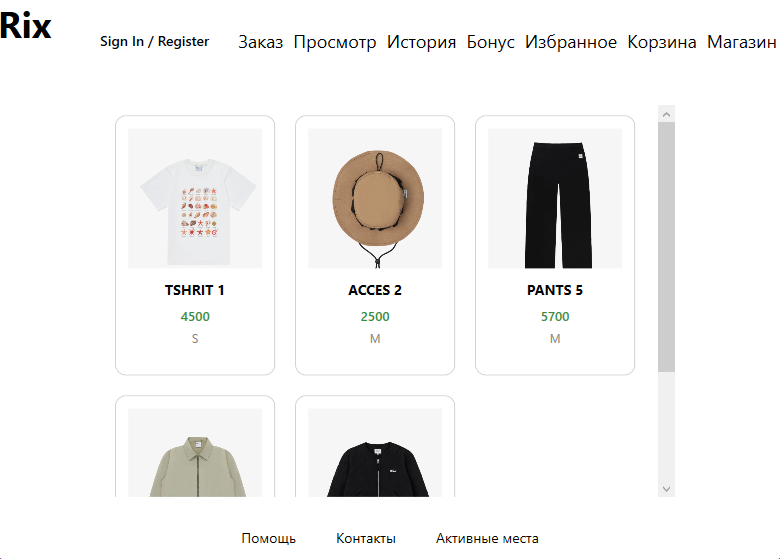


Рисунок 6.8 – Страница последних просмотров

В пункте “Активные места” содержится краткая информация о компании. А в пункте “Контакты” пользователь может оставить сообщение, прилогая свою почту, после чего администратор сможет ответить на вопрос пользователя. Пример приведен на рисунках 6.9 и 6.10.

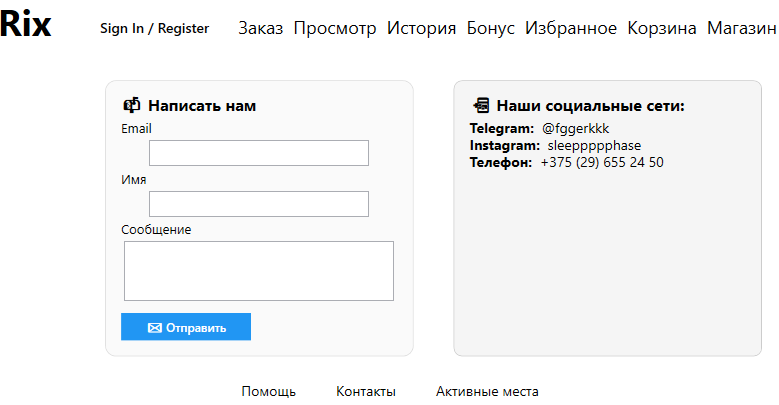


Рисунок 6.9 – Страница “Контакты”

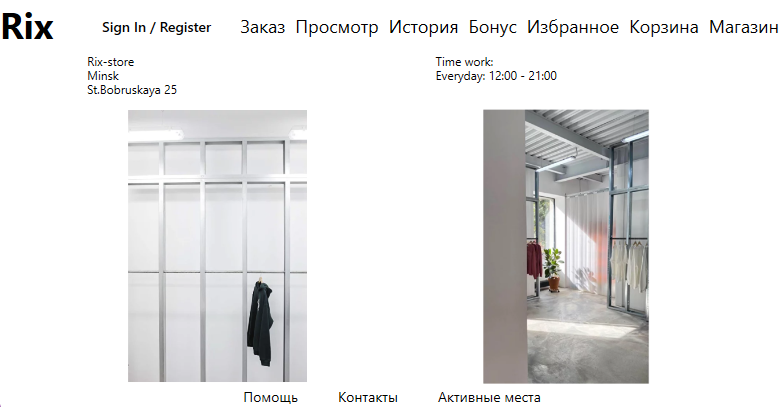


Рисунок 6.10 – Страница “Активные места”

Так же пользователь может при покупке вещей накапливать бонусы, промсотреть текущее количество бонусов и правила их начисления и использовано можно просмотреть, перейдя в пункт “Бонус”. Пример приведен на рисунке 6.11.

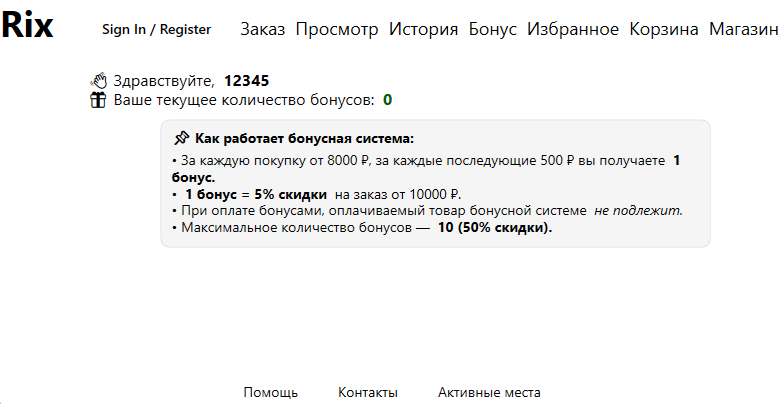


Рисунок 6.11 – Страница “Бонус”

Пользователь может просматривать текущие заказы и история покупок по пунктам “Заказ” и “История” соотвественно. Пример приведен на рисунках 6.12 и 6.13.

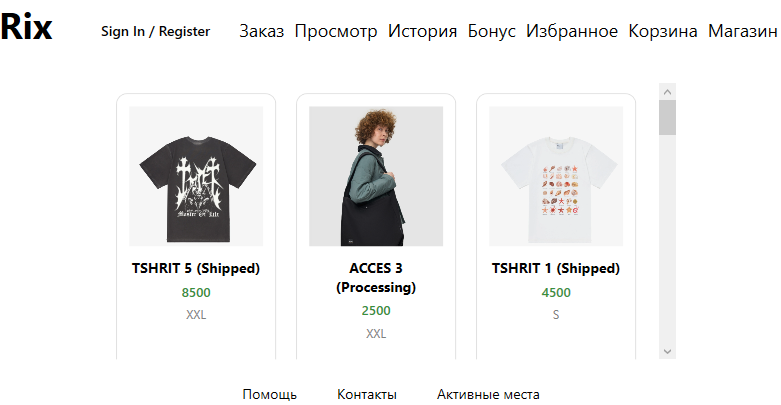


Рисунок 6.12 – Страница “Заказ”

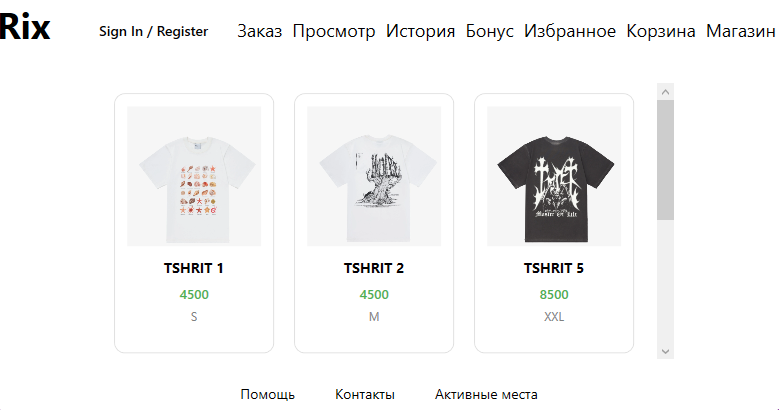


Рисунок 6.13 – Страница “История”

Так же пользователь может увидеть свои товары из корзины в пункте в соотвествующим названием, оттуда же он может оплатить товар по кнопке “Оплата”, где он может воспользоваться бонусной системой. Пример приведен на рисунках 6.14 и 6.15.

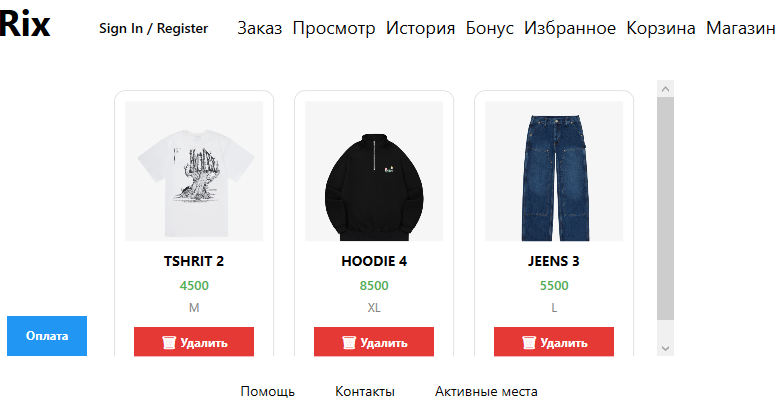


Рисунок 6.13 – Страница “Корзина”

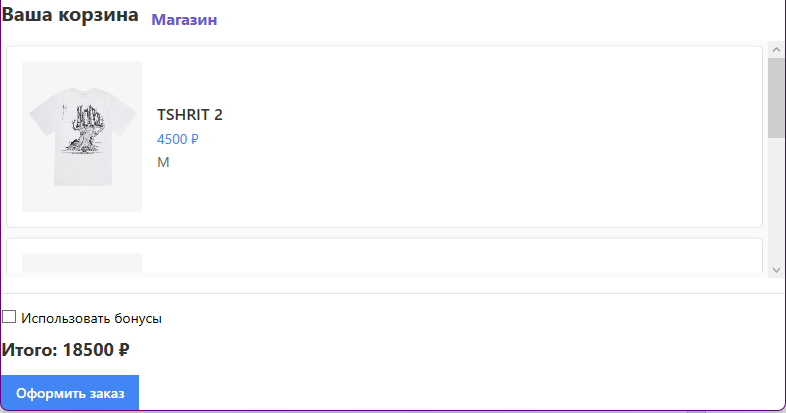


Рисунок 6.14 – Страница оплаты

Так же присуствует функциональность администратора, чтобы перейти к ней, нужно ввести при авторизации данные “Admin” в оба поля. Пример страницы приведен на рисунке 6.15.

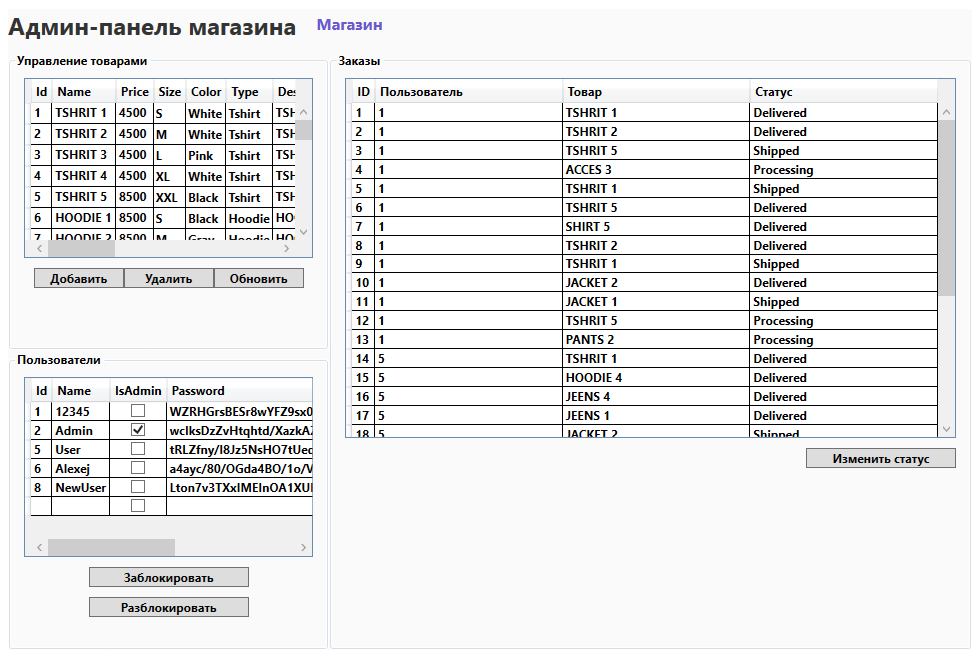


Рисунок 6.15 – Страница администратора

На данной странице администратор может добавлять, удалять и обновлять товары, изменять статус текущего заказа, заблокировать или разблокировать пользователя.

# **Выводы по разделу**

В разделе описан процесс использования программного средства. В нем пользователя инструктирую, как воспользоваться каждой из функции, которые предоставляет программное средство.

## **Заключение**

В ходе выполнения курсового проекта было разработано программное средство для управления компьютерным клубом на языке C# с использованием технологий Entity Framework Core, WPF.

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

* создана база данных;
* реализована возможность авторизации и создания новых аккаунтов;
* реализована возможность управлением пользователями;
* реализована возможность управлением товарами;
* реализована функцианолность дополнительных хранилищ товара;
* реализована возможность оплаты;
* реализована бонусная система;
* реализованы фильтры упрощенного поиска.

Тестирование программы показало, что она работает корректно и выполняет все свои функции.

В соответствии с полученным результатом работы программы можно сделать вывод, что разработанная программа работает, верно, а требования технического задания выполнены в полном объеме.

# **Список литературы**

1. Онлайн-ресурс “smartshell.gg” [Электронный ресурс] – Режим доступа smartshell.gg – Дата доступа: 25.03.2025
2. Онлайн-ресурс “astrumsoft.com” [Электронный ресурс] – Режим доступа astrumsoft.com – Дата доступа: 25.03.2025
3. Онлайн-ресурс “lightshell.ru” [Электронный ресурс] – Режим доступа lightshell.ru – Дата доступа: 27.03.2025
4. Онлайн-ресурс “ggLeap” [Электронный ресурс] – Режим доступа esportsentertainmentgroup.com – Дата доступа: 26.03.2025
5. Онлайн-ресурс “smartlaunch” [Электронный ресурс] – Режим доступа smartlaunch.com – Дата доступа: 26.03.2025
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: 28.03.2025
8. Работа с Entity Framework Core [Электронный ресурс] – https://professorweb.ru/my/entity-framework/6/level1/ – Дата доступа 23.04.2025

## **ПРИЛОЖЕНИЕ А**

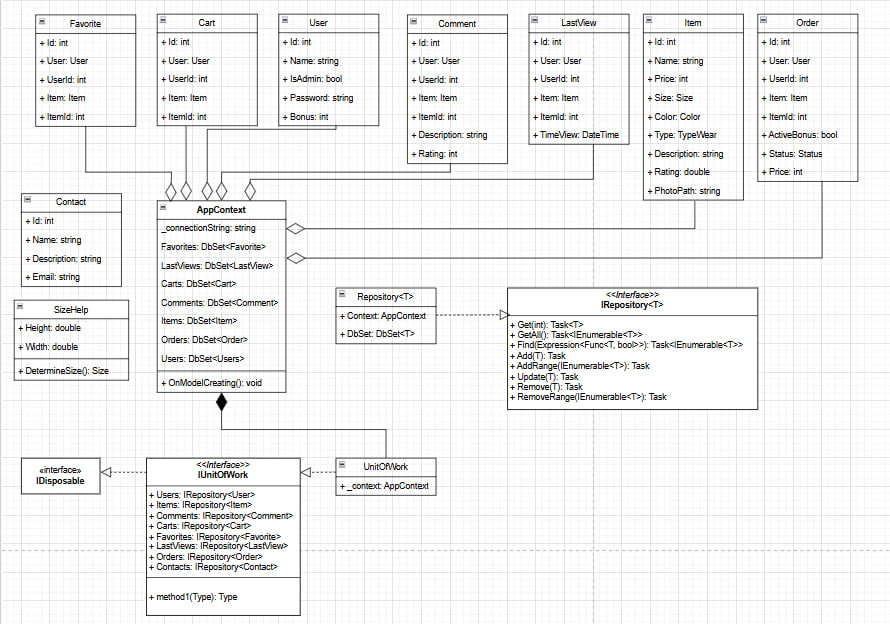


Рисунок А.1 – диаграмма классов

## **ПРИЛОЖЕНИЕ Б**



Рисунок Б.1 – диаграмма вариантов использования

## **ПРИЛОЖЕНИЕ В**

Рисунок В.1 – диаграмма последовательности

## **ПРИЛОЖЕНИЕ Г**

## Листинг 1: Класс SignViewModel

|  |
| --- |
| using OOP\_CourseWork.DataBase.Pattern.UnitOfWork;  using OOP\_CourseWork.Help;  using OOP\_CourseWork.Model.CurrentUser;  using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  using System.Windows.Input;  using System.Windows;  using OOP\_CourseWork.View;  namespace OOP\_CourseWork.ViewModel  {  public class SignViewModel : ViewModelBase  {  private readonly IUnitOfWork \_unitOfWork;  private string \_username;  public string Username  {  get => \_username;  set => Set(ref \_username, value);  }  private string \_password;  public string Password  {  get => \_password;  set => Set(ref \_password, value);  }  public ICommand SignInCommand { get; }  public SignViewModel(IUnitOfWork unitOfWork)  {  \_unitOfWork = unitOfWork;  SignInCommand = CreateAsyncCommand(SignInAsync, CanSignIn);  }  private bool CanSignIn()  {  return !string.IsNullOrWhiteSpace(Username) && !string.IsNullOrWhiteSpace(Password);  }  private async Task SignInAsync()  {  var users = await \_unitOfWork.Users.Find(u => u.Name == Username);  var user = users.FirstOrDefault();  if (user == null || !HashHelper.Verify(Password, user.Password))  {  MessageBox.Show("Неверное имя пользователя или пароль.");  return;  }  if (user.Block)  {  MessageBox.Show("Вы заблокированы");  return;  }  if (Username == "Admin" && Password == "Admin")  {  Application.Current.Dispatcher.Invoke(() =>  {  var adminWindow = new AdminWindow(\_unitOfWork);  adminWindow.Show();  Application.Current.Windows.OfType<SignWindow>().FirstOrDefault()?.Hide();  });  return;  }  CurrentUser.SetUser(user.Id, user.Bonus);  MessageBox.Show($"Добро пожаловать, {user.Name}!");  }  }  } |

## Листинг 1: Класс AdminViewModel

|  |
| --- |
| using Microsoft.EntityFrameworkCore;  using OOP\_CourseWork.DataBase.Pattern.UnitOfWork;  using OOP\_CourseWork.Model;  using System.Collections.ObjectModel;  using System.Windows;  using System.Windows.Input;  public class AdminViewModel : ViewModelBase  {  private readonly IUnitOfWork \_unitOfWork;  public ObservableCollection<Item> Items { get; } = new();  public ObservableCollection<User> Users { get; } = new();  public ICommand AddItem { get; }  public ICommand DeleteItem { get; }  public ICommand UpdateItem { get; }  public ICommand DeleteUser { get; }  public ICommand LoadAdminDataCommand { get; }  public ICommand ChangeOrderStatusCommand { get; }  public ICommand OnBlockUser1 { get; }  private Item \_selectedItem;  public Item SelectedItem  {  get => \_selectedItem;  set  {  \_selectedItem = value;  OnPropertyChanged(nameof(SelectedItem));  }  }  private User \_selectedUser;  public User SelectedUser  {  get => \_selectedUser;  set  {  \_selectedUser = value;  OnPropertyChanged(nameof(SelectedUser));  }  }  public ObservableCollection<Order> Orders { get; } = new();  private Order \_selectedOrder;  public Order SelectedOrder  {  get => \_selectedOrder;  set  {  \_selectedOrder = value;  OnPropertyChanged(nameof(SelectedOrder));  }  }  public AdminViewModel(IUnitOfWork unitOfWork)  {  \_unitOfWork = unitOfWork;  LoadAdminDataCommand = CreateAsyncCommand(LoadDataAsync);  AddItem = CreateAsyncCommand(AddItemAsync);  DeleteItem = CreateAsyncCommand(DeleteItemAsync);  UpdateItem = CreateAsyncCommand(UpdateItemAsync);  DeleteUser = CreateAsyncCommand(DeleteUserAsync);  ChangeOrderStatusCommand = CreateAsyncCommand(ChangeOrderStatusAsync);  OnBlockUser1 = CreateAsyncCommand(OnBlockUser);  LoadDataAsync();  }  private async Task LoadDataAsync()  {  Items.Clear();  var items = await \_unitOfWork.Items.GetAll();  foreach (var item in items)  Items.Add(item);  Users.Clear();  var users = await \_unitOfWork.Users.GetAll();  foreach (var user in users)  Users.Add(user);  Orders.Clear();  var orders = await \_unitOfWork.Orders.GetAll();  foreach (var order in orders)  Orders.Add(order);  }  private async Task AddItemAsync(object parameter)  {  if (parameter is not Item item)  {  MessageBox.Show("Не выбран товар");  return;  }  // Проверка заполнения всех полей  if (string.IsNullOrWhiteSpace(item.Name))  {  MessageBox.Show("Название товара не может быть пустым");  return;  }  if (item.Price <= 0)  {  MessageBox.Show("Цена должна быть больше 0");  return;  }  if (string.IsNullOrWhiteSpace(item.Description))  {  MessageBox.Show("Описание товара не может быть пустым");  return;  }  if (string.IsNullOrWhiteSpace(item.PhotoPath))  {  MessageBox.Show("Не указан путь к изображению");  return;  }  // Проверка enum-полей (хотя они всегда имеют значение по умолчанию)  if (!Enum.IsDefined(typeof(OOP\_CourseWork.Model.Size), item.Size))  {  MessageBox.Show("Указан недопустимый размер");  return;  }  if (!Enum.IsDefined(typeof(Color), item.Color))  {  MessageBox.Show("Указан недопустимый цвет");  return;  }  if (!Enum.IsDefined(typeof(TypeWear), item.Type))  {  MessageBox.Show("Указан недопустимый тип одежды");  return;  }  try  {  await \_unitOfWork.Items.Add(item);  await \_unitOfWork.CompleteAsync();  Items.Add(item);  MessageBox.Show("Товар успешно добавлен");  }  catch (Exception ex)  {  MessageBox.Show($"Ошибка при добавлении товара: {ex.Message}");  }  }  private async Task DeleteItemAsync(object parameter)  {  if (parameter is not Item item)  {  MessageBox.Show("No item selected");  return;  }  await \_unitOfWork.Items.Remove(item);  await \_unitOfWork.CompleteAsync();  Items.Remove(item);  }  private async Task UpdateItemAsync(object parameter)  {  if (parameter is not Item item)  {  MessageBox.Show("Не выбран товар для обновления");  return;  }  // Те же проверки, что и для добавления  if (string.IsNullOrWhiteSpace(item.Name))  {  MessageBox.Show("Название товара не может быть пустым");  return;  }  if (item.Price <= 0)  {  MessageBox.Show("Цена должна быть больше 0");  return;  }  if (string.IsNullOrWhiteSpace(item.Description))  {  MessageBox.Show("Описание товара не может быть пустым");  return;  }  if (string.IsNullOrWhiteSpace(item.PhotoPath))  {  MessageBox.Show("Не указан путь к изображению");  return;  }  if (!Enum.IsDefined(typeof(OOP\_CourseWork.Model.Size), item.Size))  {  MessageBox.Show("Указан недопустимый размер");  return;  }  if (!Enum.IsDefined(typeof(Color), item.Color))  {  MessageBox.Show("Указан недопустимый цвет");  return;  }  if (!Enum.IsDefined(typeof(TypeWear), item.Type))  {  MessageBox.Show("Указан недопустимый тип одежды");  return;  }  try  {  await \_unitOfWork.Items.Update(item);  await \_unitOfWork.CompleteAsync();  MessageBox.Show("Товар успешно обновлен");  }  catch (Exception ex)  {  MessageBox.Show($"Ошибка при обновлении товара: {ex.Message}");  }  }  private async Task DeleteUserAsync(object parameter)  {  try  {  if (parameter is not User user)  {  MessageBox.Show("Пользователь не найден.");  return;  }  user.Block = true; // помечаем как заблокированного  \_unitOfWork.Users.Update(user); // обновляем пользователя  await \_unitOfWork.CompleteAsync();  MessageBox.Show("Пользователь успешно заблокирован.");  }  catch (Exception ex)  {  MessageBox.Show($"Ошибка: {ex.Message}\n\n{ex.InnerException?.Message}");  }  }  private async Task OnBlockUser(object parameter)  {  try  {  if (parameter is not User user)  {  MessageBox.Show("Пользователь не найден.");  return;  }  user.Block = false; // помечаем как заблокированного  \_unitOfWork.Users.Update(user); // обновляем пользователя  await \_unitOfWork.CompleteAsync();  MessageBox.Show("Пользователь успешно заблокирован.");  }  catch (Exception ex)  {  MessageBox.Show($"Ошибка: {ex.Message}\n\n{ex.InnerException?.Message}");  }  }  private async Task ChangeOrderStatusAsync(object parameter)  {  if (SelectedOrder == null)  {  MessageBox.Show("Выберите заказ для изменения статуса.");  return;  }  SelectedOrder.Status = SelectedOrder.Status switch  {  Status.Processing => Status.Shipped,  Status.Shipped => Status.Delivered,  Status.Delivered => Status.Delivered,  \_ => SelectedOrder.Status  };  await \_unitOfWork.Orders.Update(SelectedOrder);  await \_unitOfWork.CompleteAsync();  OnPropertyChanged(nameof(Orders));  }  }  public static class SizeEnum  {  public static Array Values => Enum.GetValues(typeof(OOP\_CourseWork.Model.Size));  }  public static class ColorEnum  {  public static Array Values => Enum.GetValues(typeof(Color));  }  public static class TypeWearEnum  {  public static Array Values => Enum.GetValues(typeof(TypeWear));  } |

## Листинг 1: Класс ShopMainViewModel

|  |
| --- |
| using Microsoft.Extensions.Logging;  using OOP\_CourseWork.DataBase.Pattern.UnitOfWork;  using OOP\_CourseWork.Model;  using OOP\_CourseWork.Model.CurrentUser;  using OOP\_CourseWork.View;  using System;  using System.Collections.Generic;  using System.Collections.ObjectModel;  using System.Linq;  using System.Text;  using System.Text.RegularExpressions;  using System.Threading.Tasks;  using System.Windows;  using System.Windows.Controls;  using System.Windows.Input;  namespace OOP\_CourseWork.ViewModel  {  public class ShopMainViewModel : ViewModelBase  {  private readonly IUnitOfWork \_unitOfWork;  public ObservableCollection<Item> Items { get; } = new();  public OOP\_CourseWork.Model.Size? SelectedSize { get; set; }  public Color? SelectedColor { get; set; }  public TypeWear? SelectedType { get; set; }  public int? MinPrice { get; set; }  public int? MaxPrice { get; set; }  public double? MinRating { get; set; }  public Action? CloseAction { get; set; }  public ICommand LoadAllItemsCommand { get; }  public ICommand OpenItemPageCommand { get; }  public ICommand ApplyFilterCommand { get; }  private string \_searchText;  public string SearchText  {  get => \_searchText;  set  {  \_searchText = value;  OnPropertyChanged();  }  }  public ShopMainViewModel(IUnitOfWork unitOfWork)  {  \_unitOfWork = unitOfWork;  LoadAllItemsCommand = CreateAsyncCommand(LoadAllItemsAsync);  OpenItemPageCommand = CreateCommand(OpenItemPage);  ApplyFilterCommand = CreateAsyncCommand(ApplyFilterAsync);  LoadAllItemsCommand.Execute(null);  }  private async Task LoadAllItemsAsync()  {  Items.Clear();  var allItems = await \_unitOfWork.Items.GetAll();  foreach (var item in allItems)  Items.Add(item);  }  private async void OpenItemPage(object parameter)  {  if (parameter is Item item)  {  var window = new ShopItemWindow(\_unitOfWork, item);  var viewModel = (ShopItemViewModel)window.DataContext;  viewModel.SelectedItem = item;  window.Show();  Application.Current.Windows.OfType<ShopMainWindow>().FirstOrDefault()?.Hide();  Application.Current.Windows.OfType<CartWindow>().FirstOrDefault()?.Hide();  await viewModel.InitializeAsync();  }  }  private async Task ApplyFilterAsync()  {  var filteredItems = await \_unitOfWork.Items.GetAll();  if (SelectedSize.HasValue)  filteredItems = filteredItems.Where(item => item.Size == SelectedSize.Value).ToList();  if (SelectedColor.HasValue)  filteredItems = filteredItems.Where(item => item.Color == SelectedColor.Value).ToList();  if (SelectedType.HasValue)  filteredItems = filteredItems.Where(item => item.Type == SelectedType.Value).ToList();  if (MinPrice.HasValue)  filteredItems = filteredItems.Where(item => item.Price >= MinPrice.Value).ToList();  if (MaxPrice.HasValue)  filteredItems = filteredItems.Where(item => item.Price <= MaxPrice.Value).ToList();  if (MinRating.HasValue)  filteredItems = filteredItems.Where(item => item.Rating >= MinRating.Value).ToList();  if (!string.IsNullOrWhiteSpace(SearchText))  {  var regex = new Regex(SearchText, RegexOptions.IgnoreCase);  filteredItems = filteredItems  .Where(item => item.Name != null && regex.IsMatch(item.Name))  .ToList();  }  Items.Clear();  foreach (var item in filteredItems)  Items.Add(item);  }  }  public static class SizeEnum  {  public static Array Values => Enum.GetValues(typeof(OOP\_CourseWork.Model.Size));  }  public static class ColorEnum  {  public static Array Values => Enum.GetValues(typeof(Color));  }  public static class TypeWearEnum  {  public static Array Values => Enum.GetValues(typeof(TypeWear));  }  } |

## Листинг 1: Класс ShopItemViewModel

|  |
| --- |
| using OOP\_CourseWork.DataBase.Pattern.UnitOfWork;  using OOP\_CourseWork.Model.CurrentUser;  using OOP\_CourseWork.Model;  using System;  using System.Collections.Generic;  using System.Collections.ObjectModel;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  using System.Windows.Input;  using System.Windows.Navigation;  using OOP\_CourseWork.View;  using System.Windows;  namespace OOP\_CourseWork.ViewModel  {  public class ShopItemViewModel : ViewModelBase  {  private readonly IUnitOfWork \_unitOfWork;  public ObservableCollection<Item> Items { get; } = new();  public ObservableCollection<Comment> Comments { get; } = new();  private Item \_selectedItem;  public Item SelectedItem  {  get => \_selectedItem;  set  {  \_selectedItem = value;  OnPropertyChanged();  }  }  private string \_newCommentText;  public string NewCommentText  {  get => \_newCommentText;  set  {  \_newCommentText = value;  OnPropertyChanged();  }  }  private int \_newCommentRating = 1;  public int NewCommentRating  {  get => \_newCommentRating;  set  {  if (value < 1) value = 1;  if (value > 5) value = 5;  \_newCommentRating = value;  OnPropertyChanged();  }  }  public Action? CloseAction { get; set; }  public ICommand AddCommentCommand { get; }  public ICommand BackToShopCommand { get; }  public ICommand LoadItemCommand { get; }  public ICommand LoadCommentsCommand { get; }  public ICommand AddToCartCommand { get; }  public ICommand AddToFavoriteCommand { get; }  public ShopItemViewModel(IUnitOfWork unitOfWork,Item selectedItem)  {  \_unitOfWork = unitOfWork;  SelectedItem = selectedItem;  LoadItemCommand = CreateAsyncCommand(LoadItemAsync);  LoadCommentsCommand = CreateAsyncCommand(LoadCommentAsync);  AddToCartCommand = CreateAsyncCommand(AddToCartAsync);  AddToFavoriteCommand = CreateAsyncCommand(AddToFavoriteAsync);  AddCommentCommand = CreateAsyncCommand(AddCommentAsync);  BackToShopCommand = CreateAsyncCommand(BackToShop);  }  public async Task InitializeAsync()  {  await LoadItemAsync();  await LoadCommentAsync();  }  private async Task BackToShop()  {  var currentWindow = Application.Current.Windows  .OfType<Window>()  .FirstOrDefault(w => w.DataContext == this);  var mainWindow = Application.Current.Windows.OfType<ShopMainWindow>().FirstOrDefault();  if (mainWindow != null)  {  mainWindow.DataContext = new ShopMainViewModel(\_unitOfWork);  mainWindow.Title = "Магазин";  }  Application.Current.MainWindow = mainWindow;  mainWindow?.Show();  currentWindow?.Hide();  }  private async Task LoadItemAsync()  {  Items.Clear();  if (SelectedItem == null)  {  MessageBox.Show("SelectedItem == null, загрузка невозможна");  return;  }  if (\_unitOfWork.LastViews == null)  {  MessageBox.Show("LastViews репозиторий не инициализирован");  return;  }  var itemId = CurrentItem.ItemId;  try  {  if (CurrentUser.UserId != 0)  {  var lastview = new LastView  {  UserId = CurrentUser.UserId,  ItemId = SelectedItem.Id,  TimeView = DateTime.Now  };  await \_unitOfWork.LastViews.Add(lastview);  await \_unitOfWork.CompleteAsync();  }  }  catch (Exception ex)  {  MessageBox.Show($"Ошибка при добавлении LastView: {ex.Message}");  return;  }  var favorites = await \_unitOfWork.Items.Find(f => f.Id == itemId);  foreach (var fav in favorites)  {  Items.Add(fav);  }  }  private async Task LoadCommentAsync()  {  Comments.Clear();  var itemId = SelectedItem?.Id ?? 0;  var favorites = await \_unitOfWork.Comments.Find(f => f.ItemId == itemId);  if (itemId == 0)  {  MessageBox.Show("SelectedItem не установлен, невозможно загрузить комментарии");  return;  }  foreach (var fav in favorites)  {  Comments.Add(fav);  }  }  private async Task AddToCartAsync()  {  if (SelectedItem == null)  {  MessageBox.Show("Товар не выбран.");  return;  }  if (CurrentUser.UserId == 0)  {  MessageBox.Show("Вы не авторизованы.");  return;  }  var userId = CurrentUser.UserId;  var itemId = SelectedItem.Id;  var existing = await \_unitOfWork.Carts.Find(c => c.UserId == userId && c.ItemId == itemId);  if (existing.Any())  {  MessageBox.Show("Этот товар уже в корзине.");  return;  }  var cartItem = new Cart  {  UserId = userId,  ItemId = itemId  };  await \_unitOfWork.Carts.Add(cartItem);  await \_unitOfWork.CompleteAsync();  }  private async Task AddToFavoriteAsync()  {  if (SelectedItem == null)  {  MessageBox.Show("Товар не выбран.");  return;  }  if (CurrentUser.UserId == 0)  {  MessageBox.Show("Вы не авторизованы.");  return;  }  var userId = CurrentUser.UserId;  var itemId = SelectedItem.Id;  var existing = await \_unitOfWork.Favorites.Find(c => c.UserId == userId && c.ItemId == itemId);  if (existing.Any())  {  MessageBox.Show("Этот товар уже в избранном.");  }  var cartItem = new Favorite  {  UserId = userId,  ItemId = itemId  };  await \_unitOfWork.Favorites.Add(cartItem);  await \_unitOfWork.CompleteAsync();  }  private async Task UpdateItemRatingAsync(int itemId)  {  var comments = await \_unitOfWork.Comments.Find(c => c.ItemId == itemId);  if (comments.Any())  {  double avgRating = comments.Average(c => c.Rating);  var items = await \_unitOfWork.Items.Find(i => i.Id == itemId);  var item = items.FirstOrDefault();  if (item != null)  {  item.Rating = avgRating;  \_unitOfWork.Items.Update(item);  await \_unitOfWork.CompleteAsync();  SelectedItem.Rating = avgRating;  OnPropertyChanged(nameof(SelectedItem));  }  }  }  private async Task AddCommentAsync()  {  if (string.IsNullOrWhiteSpace(NewCommentText))  {  MessageBox.Show("Комментарий не может быть пустым.");  return;  }  if (NewCommentRating < 1 || NewCommentRating > 5)  {  MessageBox.Show("Рейтинг должен быть от 1 до 5.");  return;  }  if (CurrentUser.UserId == 0)  {  MessageBox.Show("Только авторизованные пользователи могут оставлять комментарии.");  return;  }  var comment = new Comment  {  Description = NewCommentText,  UserId = CurrentUser.UserId,  ItemId = SelectedItem.Id,  Rating = NewCommentRating  };  await \_unitOfWork.Comments.Add(comment);  await \_unitOfWork.CompleteAsync();  await UpdateItemRatingAsync(SelectedItem.Id);  NewCommentText = string.Empty;  NewCommentRating = 1;  await LoadCommentAsync();  }  }  } |

## Листинг 1: Класс CartViewModel

|  |
| --- |
| using OOP\_CourseWork.DataBase.Pattern.UnitOfWork;  using OOP\_CourseWork.Model;  using OOP\_CourseWork.Model.CurrentUser;  using System;  using System.Collections.Generic;  using System.Collections.ObjectModel;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  using System.Windows;  using System.Windows.Input;  namespace OOP\_CourseWork.ViewModel  {  public class CartViewModel : ViewModelBase  {  private readonly IUnitOfWork \_unitOfWork;  public ICommand LoadCartCommand { get; }  public ICommand RemoveCartCommand { get; }  private ObservableCollection<Item> \_cartItems = new ObservableCollection<Item>();  public ObservableCollection<Item> CartItems  {  get => \_cartItems;  set  {  \_cartItems = value;  OnPropertyChanged(nameof(CartItems));  }  }  public CartViewModel(IUnitOfWork unitOfWork)  {  \_unitOfWork = unitOfWork;  LoadCartCommand = CreateAsyncCommand(LoadCartAsync);  RemoveCartCommand = CreateAsyncCommand(RemoveCartAsync);  LoadCartCommand.Execute(null);  }  private async Task LoadCartAsync()  {  try  {  CartItems.Clear();  var carts = await \_unitOfWork.Carts.GetAll();  var items = carts.Where(c => c.UserId == CurrentUser.UserId).Select(x => x.ItemId).ToList();  var allItems = await \_unitOfWork.Items.GetAll();  var filteredItems = allItems.Where(item => items.Contains(item.Id)).ToList();  foreach (var item in filteredItems)  {  if (item.Name != null)  {  CartItems.Add(item);  }    }  OnPropertyChanged(nameof(CartItems));  }  catch (Exception ex)  {  MessageBox.Show($"Error: {ex.Message}");  }  }  private async Task RemoveCartAsync(object parameter)  {  if (parameter is Item itemToRemove)  {  var carts = await \_unitOfWork.Carts.GetAll();  var cartToRemove = carts.FirstOrDefault(c => c.ItemId == itemToRemove.Id && c.UserId == CurrentUser.UserId);  if (cartToRemove != null)  {  await \_unitOfWork.Carts.Remove(cartToRemove);  CartItems.Remove(itemToRemove);  await \_unitOfWork.CompleteAsync();  }  }  }  }  } |

## Листинг 1: Класс PaymentViewModel

|  |
| --- |
| using OOP\_CourseWork.DataBase.Pattern.UnitOfWork;  using OOP\_CourseWork.Model;  using OOP\_CourseWork.Model.CurrentUser;  using System.Collections.ObjectModel;  using System.Linq;  using System.Threading.Tasks;  using System.Windows;  using System.Windows.Input;  public class PaymentViewModel : ViewModelBase  {  private readonly IUnitOfWork \_unitOfWork;  public ObservableCollection<Item> CartItems { get; } = new();  public ICommand PayCommand { get; }  public ICommand LoadPaymentDataCommand { get; }  private bool \_useBonus;  public bool UseBonus  {  get => \_useBonus;  set  {  \_useBonus = value;  OnPropertyChanged(nameof(UseBonus));  OnPropertyChanged(nameof(TotalPrice));  }  }  public int RawTotalPrice => CartItems.Sum(i => i.Price);  public int TotalPrice  {  get  {  if (!UseBonus)  return RawTotalPrice;  if (RawTotalPrice < 10000 || CurrentUser.Bonus == 0)  return RawTotalPrice;  int maxBonuses = Math.Min(CurrentUser.Bonus, 10);  double discount = 0.05 \* maxBonuses;  return (int)(RawTotalPrice \* (1 - discount));  }  }  public PaymentViewModel(IUnitOfWork unitOfWork)  {  \_unitOfWork = unitOfWork;  LoadPaymentDataCommand = CreateAsyncCommand(LoadDataAsync);  PayCommand = CreateAsyncCommand(ExecutePaymentAsync);  LoadDataAsync();  }  private async Task LoadDataAsync()  {  CartItems.Clear();  var carts = await \_unitOfWork.Carts.GetAll();  var items = carts  .Where(c => c.UserId == CurrentUser.UserId)  .Select(c => c.Item)  .ToList();  foreach (var item in items)  CartItems.Add(item);  OnPropertyChanged(nameof(TotalPrice));  }  private async Task ExecutePaymentAsync(object? obj)  {  if (CartItems.Count == 0)  {  MessageBox.Show("Корзина пуста");  return;  }  int rawTotal = RawTotalPrice;  int userBonus = CurrentUser.Bonus;  var users = await \_unitOfWork.Users.Find(x => x.Id == CurrentUser.UserId);  var user = users.FirstOrDefault();  if (user == null)  {  MessageBox.Show("Пользователь не найден");  return;  }  foreach (var item in CartItems)  {  var order = new Order  {  UserId = CurrentUser.UserId,  ItemId = item.Id,  Price = item.Price,  ActiveBonus = UseBonus,  Status = Status.Processing  };  await \_unitOfWork.Orders.Add(order);  }  if (UseBonus && rawTotal >= 10000 && userBonus > 0)  {  int bonusesUsed = Math.Min(userBonus, 10);  user.Bonus -= bonusesUsed;  }  else if (!UseBonus && rawTotal >= 8000)  {  int extra = rawTotal - 8000;  int bonusEarned = extra / 500;  user.Bonus += bonusEarned;  }  await \_unitOfWork.Users.Update(user);  var allCarts = await \_unitOfWork.Carts.GetAll();  var userCarts = allCarts.Where(c => c.UserId == CurrentUser.UserId).ToList();  foreach (var cart in userCarts)  {  await \_unitOfWork.Carts.Remove(cart);  }  await \_unitOfWork.CompleteAsync();  CurrentUser.Bonus = user.Bonus;  CartItems.Clear();  OnPropertyChanged(nameof(TotalPrice));  MessageBox.Show("Оплата прошла успешно");  }  } |

## Листинг 1: Класс OrderViewModel

|  |
| --- |
| using OOP\_CourseWork.DataBase.Pattern.UnitOfWork;  using OOP\_CourseWork.Model.CurrentUser;  using OOP\_CourseWork.Model;  using System;  using System.Collections.Generic;  using System.Collections.ObjectModel;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  using System.Windows.Input;  using System.Windows;  namespace OOP\_CourseWork.ViewModel  {  public class OrdersViewModel : ViewModelBase  {  private readonly IUnitOfWork \_unitOfWork;  public ObservableCollection<Item> Orders = new ObservableCollection<Item>();  public ICommand LoadOrdersCommand { get; }  public ObservableCollection<Item> OrdersItems  {  get => Orders;  set  {  Orders = value;  OnPropertyChanged(nameof(OrdersItems));  }  }  public OrdersViewModel(IUnitOfWork unitOfWork)  {  \_unitOfWork = unitOfWork;  LoadOrdersCommand = CreateAsyncCommand(LoadOrdersAsync);  LoadOrdersAsync();  }  private async Task LoadOrdersAsync()  {  try  {  OrdersItems.Clear();  var allOrders = await \_unitOfWork.Orders.GetAll();  var userOrders = allOrders  .Where(x => x.UserId == CurrentUser.UserId &&  (x.Status == Status.Processing || x.Status == Status.Shipped))  .ToList();  var allItems = await \_unitOfWork.Items.GetAll();  var itemWithStatusList = userOrders  .Join(allItems,  order => order.ItemId,  item => item.Id,  (order, item) => new  {  item.PhotoPath,  item.Name,  item.Price,  item.Size,  order.Status  })  .ToList();  foreach (var x in itemWithStatusList)  {  OrdersItems.Add(new Item  {  Name = $"{x.Name} ({x.Status})",  PhotoPath = x.PhotoPath,  Price = x.Price,  Size = x.Size  });  }  OnPropertyChanged(nameof(OrdersItems));  }  catch (Exception ex)  {  MessageBox.Show($"Error: {ex.Message}");  }  }  }  } |

## 