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

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**КУРСОВОГО ПРОЕКТА:**

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

Тема Программное обеспечение «Интернет-магазин MU Store»

Исполнитель

студент 2 курса 4 группы Казакевич Дмитрий Петрович

Руководитель работы ассистент Рауба А.А.

Курсовой проект защищен с оценкой

Председатель Рауба А.А.

Минск 2020

# **Содержание**

[Содержание 2](#_Toc41227583)

[ВВЕДЕНИЕ 4](#_Toc41227584)

[1 Постановка задачи и обзор литературы 5](#_Toc41227585)

[1.1 Алгоритмы решения 5](#_Toc41227586)

[1.2 Обзор прототипов 6](#_Toc41227587)

[1.3 Актуальность задачи 9](#_Toc41227588)

[2 Проектирование архитектуры проекта 11](#_Toc41227589)

[2.1 Архитектура системы 11](#_Toc41227590)

[2.2 Проектирование логической структуры базы данных 11](#_Toc41227591)

[2.3 Проектирование структуры классов 12](#_Toc41227592)

[2.4 Проектирование структуры окон 14](#_Toc41227593)

[3 Разработка функциональной модели и модели данных ПС 16](#_Toc41227594)

[3.1 Функции, реализуемые программой 16](#_Toc41227595)

[3.2 Реализация классов 16](#_Toc41227596)

[3.3 Реализация работы с базой данных 17](#_Toc41227597)

[3.4 Реализация паттерна MVVM 18](#_Toc41227598)

[4 Тестирование 21](#_Toc41227599)

[4.1 Тест авторизации 21](#_Toc41227600)

[4.2 Тест регистрации 23](#_Toc41227601)

[4.3 Тест отправки сообщения 26](#_Toc41227602)

[4.4 Тест использования калькулятора 27](#_Toc41227603)

[ЗАКЛЮЧЕНИЕ 28](#_Toc41227604)

[Список используемых источников 29](#_Toc41227605)

[ПРИЛОЖЕНИЕ А 30](#_Toc41227606)

[ПРИЛОЖЕНИЕ Б 31](#_Toc41227607)

[ПРИЛОЖЕНИЕ В 32](#_Toc41227608)

[ПРИЛОЖЕНИЕ Г 33](#_Toc41227609)

[ПРИЛОЖЕНИЕ Д 36](#_Toc41227610)

[ПРИЛОЖЕНИЕ Е 37](#_Toc41227611)

[ПРИЛОЖЕНИЕ Ж 38](#_Toc41227612)

[ПРИЛОЖЕНИЕ З 41](#_Toc41227613)

[ПРИЛОЖЕНИЕ И 42](#_Toc41227614)

# **ВВЕДЕНИЕ**

Выбранная тема считается актуальной на сегодняшний день, так как сегодня миллионы людей ежедневно, не выходя из дому, покупают различные товары в электронных магазинах.  В мире огромными темпами растет количество пользователей Internet и, как следствие, количество «электронных» покупателей и потенциальных «электронных» покупателей.

Все большее количество фирм старается представить свою продукцию в online среде. При этом такое представление не ограничивается только лишь созданием промо-сайтов и размещением рекламных баннеров и статей в электронных журналах и на информационных порталах. С развитием Интернет-среды развивается и само предложение. Теперь люди могут не только получать интересующую их информацию, но и совершать покупки. При этом с помощью Интернет-магазинов можно приобретать товары совершенно разных категорий, как элементарные потребительские, так и высокотехнологичные. Такое положение вещей обусловлено, во-первых, постоянным, стабильным ростом аудитории пользователей глобальной сети. Основную часть аудитории сети Интернет составляют люди, работающие в офисах и проводящие большую часть времени за компьютером. Как правило, их образ жизни не позволяет им тратить большое количество времени на походы по offline магазинам в поисках именно того, что им нужно. Возможность заказать интересующий продукт в Интернете является для них действительно выходом из ситуации.

Целью моего проекта является создания приложения «MU Store» – это программное средство, предоставляющее сайт для клиентов, в котором они могут выбрать и купить интересующие их футбольную атрибутику футбольного клуба Манчестер Юнайтед, а также[предоставляющее] десктопное приложение для администратора, который может управлять базой данных сайта и тем самым обновлять на сайте, а также просматривать статистику заказов.

В качестве интерфейса прикладного программирования выбран обширный API-интерфейс – Windows Presentation Foundation (WPF), предназначенный для создания настольных программ с графически насыщенным пользовательским интерфейсом. В основе графической технологии WPF лежит мощная инфраструктура, основанная на DirectX. Это является одним из основных отличий WPF от более ранней технологии создания пользовательских интерфейсов – Windows Formsа также ASP.NET MVC Framework — фреймворк для создания веб-приложений, который реализует архитектурный шаблон Model-view-controller. Язык разработки проекта – C#. При выполнении курсового проекта будут использованы принципы и приемы ООП.

# **1 Постановка задачи и обзор литературы**

Основной задачей курсового проекта является разработка программного средства, которое позволяет хранить информацию в базе данных и добавлять новую информацию в уже имеющуюся.

Функционально приложение поддерживает:

Функции клиента:

* Выполнять регистрацию и авторизацию;
* Создание, редактирование и удаление заказов;
* Просматривать личную статистику;
* Фильтрация товаров;
* Выполнять поисковые запросы.

Функции администратора:

* Поддерживать работу с базой данных;
* Просматривать общую статистику заказанных товаров.

## **1.1 Алгоритмы решения**

При разработке приложения были использованы следующие технологии:

* WPF;
* Entity Framework;
* Microsoft SQL Server;
* ASP NET MVC;
* Ninject.

Windows Presentation Foundation (WPF) – это современная графическая система отображения для Windows. Это радикальное изменение от технологий, которые были до него, с инновационными функциями, такими как встроенные аппаратное ускорение и независимость от разрешения, которые вы будете изучать в этой главе.

WPF – это лучший набор инструментов, который можно использовать, если вы хотите создать богатое настольное приложение, работающее на Windows 7, Windows 8 и Windows 10 в режиме рабочего стола, на основе современных пользовательских интерфейсов, которые могут включать в себя мультимедиа, графика, документы и различные данные [1].

Entity Framework (EF) – это объектно-ориентированная и легко расширяемая технология доступа к данным. EF представляет собой более высокий уровень абстракции по сравнению с ADO.NET и позволяет абстрагироваться от самой базы данных и ее таблиц, работая с данными независимо от типа хранилища.

Entity Framework поддерживает множество различных систем баз данных. По умолчанию Microsoft предоставляет ряд встроенных провайдеров: для работы с MS SQL Server, для SQLite, для MySQL.

Центральной концепцией Entity Framework является понятие сущности или entity. Сущность представляет набор данных, ассоциированных с определенным объектом. Поэтому данная технология предполагает работу не с таблицами, а с объектами и их наборами. Отличительной чертой Entity Framework является использование запросов LINQ для выборки данных из БД. С помощью LINQ мы можем не только извлекать определенные строки, хранящие объекты из БД, но и получать объекты, связанные различными ассоциативными связями [2].

Microsoft SQL Server – это система управления реляционными базами данных, разработанная корпорацией Microsoft. Основной используемый язык запросов – Transact-SQL, который является реализацией стандарта ANSI/ISO по структурированному языку запросов (SQL) с расширениями. Используется для работы с базами данных размером от персональных до крупных баз данных масштаба предприятия.

Microsoft SQL Server в качестве языка запросов использует версию SQL являющуюся реализацией SQL-92 с множественными расширениями. T-SQL позволяет использовать дополнительный синтаксис для хранимых процедур и обеспечивает поддержку транзакций [3].

Инфраструктура ASP.NET MVC 5 представляет собой последнюю версию веб-платформы ASP.NET от Microsoft. Она предлагает высокопродуктивную модель программирования, которая способствует построению более чистой кодовой архитектуры, обеспечивает разработку через тестирование и поддерживает повсеместную расширяемость в комбинации со всеми преимуществами ASP.NET.

У инфраструктуры ASP.NET MVC есть множество преимуществ, по сравнению с классической платформой веб-разработки ASP.NET Web Forms. Ее встроенные вспомогательные методы HTML генерируют ясный и соответствующий стандартам код разметки, она предлагает мощную систему маршрутизации URL (теперь доступна и в ASP.NET Web Forms 4.5) которая позволяет создавать удобочитаемые URL-адреса. Компоненты ASP.NET MVC поддерживают расширяемость, а сама платформа поддерживает гибкую тестируемость в виде модульных и интеграционных тестов[4].

Ninject – это мощный и одновременно легкий в использовании IoC контейнер. Inversion of Control (инверсия управления) – это некий абстрактный принцип, набор рекомендаций для написания слабо связанного кода. Суть которого в том, что каждый компонент системы должен быть как можно более изолированным от других, не полагаясь в своей работе на детали конкретной реализации других компонентов.

Dependency Injection (внедрение зависимостей) – это одна из реализаций этого принципа (помимо этого есть еще Factory Method, Service Locator). Если более подробно рассмотреть Dependency Injection (DI), то это паттерн проектирования программного обеспечения, который позволяет разрабатывать слабосвязанный код[5].

## **1.2 Обзор прототипов**

Сайт «soccershop.by»

Данный сайт характеризуется наличием авторизации и аутентификации пользователей, возможностью выбирать товары как по футбольным клубам, так и по национальным сборным, возможностью добавлять товары в корзину и оформлять заказ. Его интерфейс представлен на рисунке 1.1.

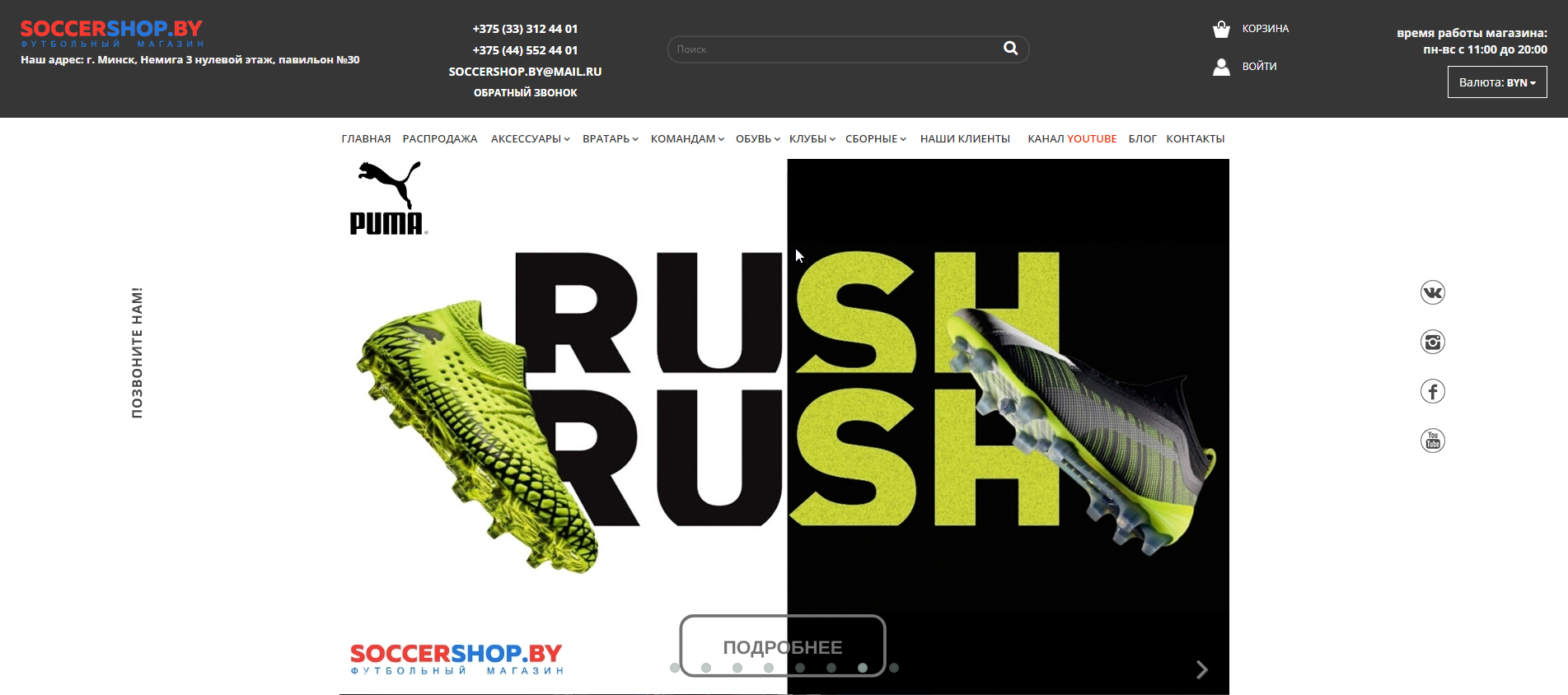


Рисунок 1.1 – Интерфейс сайта «soccershop.by»

Из недостатков необходимо отметить не самый приятный функционал и интуитивно непонятный дизайн программного средства.

Сайт «FanShop.by»

На данном сайте можно приобрести футбольную атрибутику и экипировку. Сайт позволяет быстро перемещаться по категориям и составлять заказ. Много доступной информации, которую плюс ко всему легко найти и очень быстрый отклик. Его интерфейс представлен на рисунке 1.2.

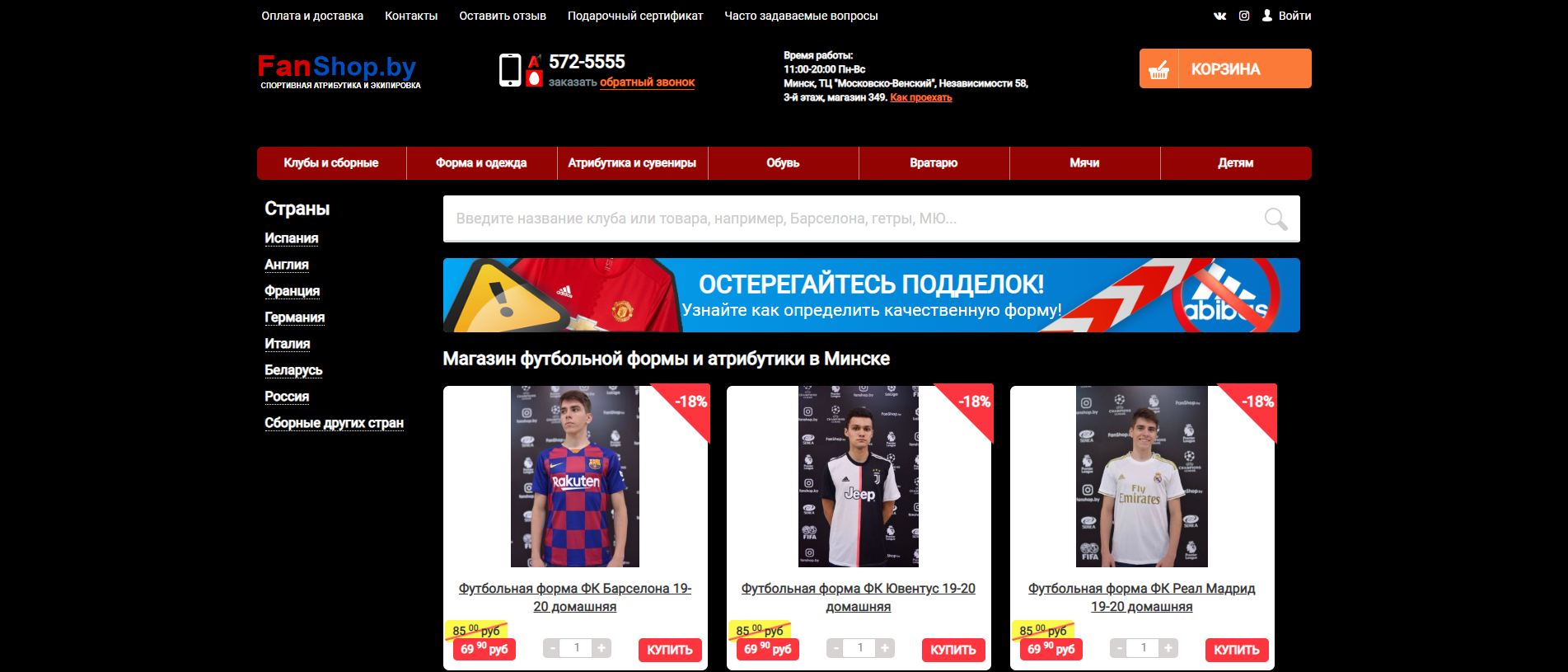


Рисунок 1.2 – Интерфейс сайта « FanShop.by »

Из недостатков необходимо отметить не самый приятный дизайн, т.к. он оформлен в чёрном и красном цвете, что не очень приятно для восприятия.

## **1.3 Актуальность задачи**

На сегодняшний день интернет-магазины являются важной частью нашей жизни. С помощью них удобно совершать покупки, не выходя из дома, находить необходимый товар, не тратя время на походы по магазинам, так как все можно заказать онлайн. Поэтому тема курсового проекта «Интернет- магазин футбольной атрибутики Manchester United» выбрана не случайно.

Данное приложение предоставляет выбор футбольной атрибутики и позволяет быстро совершать покупки онлайн.

MU-Store имеет ряд преимуществ:

* Привлекательный интерфейс и понятный функционал;
* Узнаваемость брендов;
* Широкий ассортимент товаров;
* Клиент имеет неограниченное время на выбор товаров;
* Возможность редактирования заказа;
* Если заказ ещё не отправлен, его можно легко отменить.

Помимо этого, в приложении присутствует возможность просмотра личной статистики клиента.

# **2 Проектирование архитектуры проекта**

## **2.1 Архитектура системы**

Для программного средства в курсовом проекте используется способ разработки программных модулей, основанный на клиентском приложении для администраторов, а также клиент-серверном приложении для пользователей. Приложение состоит из 3 частей:

* Клиентской части, в которой происходит основное взаимодействие пользователя. Реализацией данной части является сайт.
* Администрирующей части, в которой происходит основное взаимодействие с базой данных. Реализацией данной части является десктопное приложение.
* Базы данных. В данной БД хранятся товары и их характеристики, информация о пользователях, роли и соответствующие им пользователи.

## **2.2 Проектирование логической структуры базы данных**

При помощи подхода Code-First была создана база данных. Этот подход предполагает минимальное участие в проектировании сущностей базы данных программистом. Для её создания использовались система управления реляционными базами данных Microsoft SQL Server 2018.

База данных программной подсистемы состоит из 12 таблиц, 11 из которых взаимосвязанных друг с другом внешними ключами, ее схема представлена в приложении А.

Таблица Clothes хранит данные для входа. Включает 4 столбца: Article, Name, Description, Category, Price. Столбец Article является первичным ключом, инкрементируется при добавлении нового товара. Столбец Name хранит название товара. Столбец Description хранит описание товара. Столбец Category хранит категорию товара. Столбец Price хранит цену товара.

Таблица ClothesImages хранит картинки товаров. Включает 4 столбца: ClothesImagesID, FileId, ImageFile, ClothesId. Столбец ClothesImagesID является первичным ключом, инкрементируется при отправке нового сообщения. Столбец FileId хранит идентификатор файла. Столбец ImageFile хранит данные varbinary (max) для больших двоичных объектов (BLOB) в виде файлов в файловой систем. Столбец ClothesId хранит идентификатор товара, которому соответствует данный файл.

Таблица Sizes хранит размеры товаров. Включает 3 столбца: sizeId, size, ClothesId. Столбец sizeId является первичным ключом, инкрементируется при добавлении нового размера. Столбец size хранит размер в строковом виде. Столбец ClothesId хранит идентификатор товара, которому соответствует данный размер.

Таблица Orders хранит информацию о заказах пользователей. Включает 5 столбцов: Id, UserId, DateTime, TotalCost, Status. Столбец Id является первичным ключом, инкрементируется при добавлении нового заказа. Столбец UserId хранит идентификатор пользователя, совершившего заказ. Столбец DateTime хранит время, в которое был совершен заказ. Столбец TotalCost хранит суммарную стоимость заказа. Столбец Status хранит текущий статус заказа.

Таблица OrderDetails хранит детали заказов. Включает 5 столбцов: Id, OrderId, ProductId, Quantity, Size. Столбец Id является первичным ключом, инкрементируется при новом заказе. Столбец OrderId хранит идентификатор заказа. Столбец ProductId хранит идентификатор товара, который был заказан. Столбец Quantity количество единиц товара, который был заказан. Столбец Size хранит размер заказанного товара, если таковой имеется.

Таблица AspNetUsers хранит информацию о пользователях. Включает 12 столбцов: Id, Email, EmailConfirmed, PasswordHash, SecurityStamp, PhoneNumber, PhoneNumberConfirmed, TwoFactorEnabled, LockoutEndDateUtc, LockoutEnabled, AccessFailedCount, UserName. Столбец Id является первичным ключом, инкрементируется при регистрации нового пользователя. Столбец Email хранит электронную почту пользователя. Столбец EmailConfirmed хранит информацию о том, подтвердил ли пользователь свою почту. Столбец PasswordHash хранит хеш пароля пользователя. Столбец SecurityStamp хранит GUID, автоматически создаваемый в определенные моменты жизненного цикла объекта UserManager. Обычно создается и обновляется при смене пароля или добавлении/удалении логина через социальные сети. Метка безопасности (security stamp), как правило, создает снимок информации о пользователях и обеспечивает автоматический вход пользователей, если ничего не изменялось. Столбец PhoneNumber хранит номер телефона пользователя. Столбец PhoneNumberConfirmed информацию о том, подтвердил ли пользователь свой номер телефона. Столбец TwoFactorEnabled хранит информацию о том, доступна ли пользователю двухфакторная аутентификация. Стоблец LockoutEndDateUtc хранит дату разблокировки пользователя, если таковой был заблокирован. Столбец LockoutEnabled хранит информацию о том, может ли пользователь быть заблокирован. Столбец AccessFailedCount хранит количество неудачных попыток доступа пользователя. Столбец UserName хранит имя пользователя, в моей реализации данное поле совпадает с полем Email пользователя.

Таблица AspNetUserRoles хранит роли пользователей. Включает 2 столбца: UserId и RoleId. Столбец UserId хранит идентификатор пользователя. Столбец RoleId хранит идентификатор роли.

Таблица AspNetUserLogins служит для хранения информации о сторонних / внешних входах в систему, например, о пользователях, которые входят на ваш сайт через Google, Facebook, Twitter и т.д. Включает 3 столбца: LoginProvider, ProviderKey, UserId. Столбец LoginProvider хранит имя службы, предоставившей логин, например, «Facebook». Столбец ProviderKey хранит уникальный ключ, связанный с пользователем на стороннем ресурсе.

Таблица AspNetUserClaims служит дополнительной информацией о пользователях, на основе которой можно принимать решения по авторизации. Эти утверждения могут быть получены из внешних источников данных, а также из локальной базы данных Identity. Содержит 4 столбца: Id, UserId, ClaimType, ClaimValue. Столбец Id является первичным ключом. Стоблец UserId хранит идентификатор пользователя. Столбец ClaimType хранит тип утверждения. Столбец ClaimValue хранит значение утверждения.

Таблица AspNetRoles хранит роли пользователей. Содержит 2 столбца: Id и Name. Столбец Id является первичным ключом. Столбец Name хранит название роли.

## **2.3 Проектирование структуры классов**

Приложение курсового проекта состоит из 3 проектов: Domain, UnitedDirectManager, WebUI (рисунок 2.1).

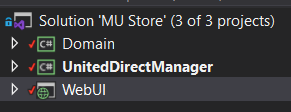


Рисунок 2.1 – Состав приложения

Проект Domain представляет собой библиотеку и включает в себя основные классы для работы с базой данных, модели, согласно которым генерируется база данных при помощи подхода Code-First. Также содержит абстракции в виде интерфейсов и абстрактных классов и их конкретные реализации. (рисунок 2.2).

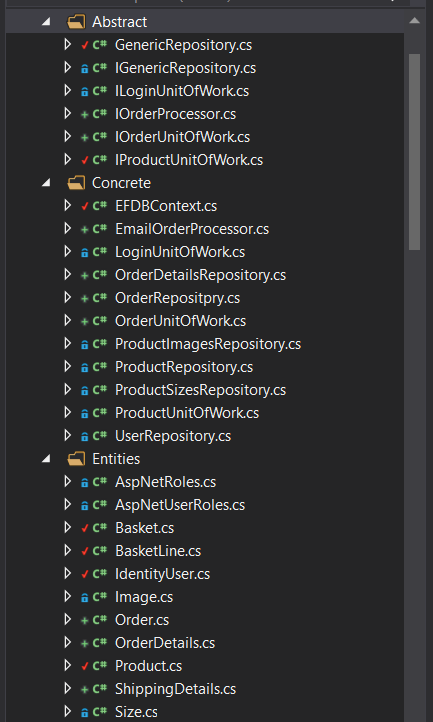


Рисунок 2.2 – Состав проекта Domain

Проект UnitedDirectManager состоит из страниц в папке View и соответствующих им классам с суффиксом ViewModel (рисунок 2.3).

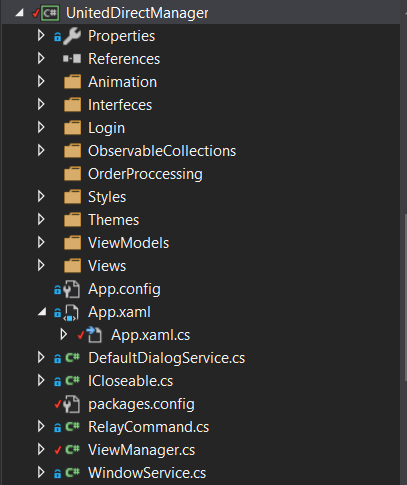


Рисунок 2.3 – Состав проекта UnitedDirectManager

Проект WebUI является веб-приложением для клиентов (рисунок 2.4).

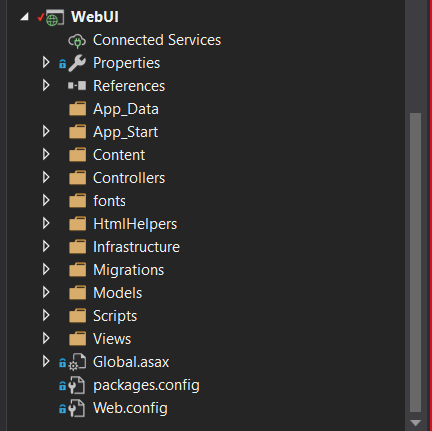


Рисунок 2.4 – Состав проекта WebUI

## **2.4 Проектирование структуры окон**

Настольное приложение включает одно главное окно и 10 страниц. Связи между ними представлены на рисунке 2.5.

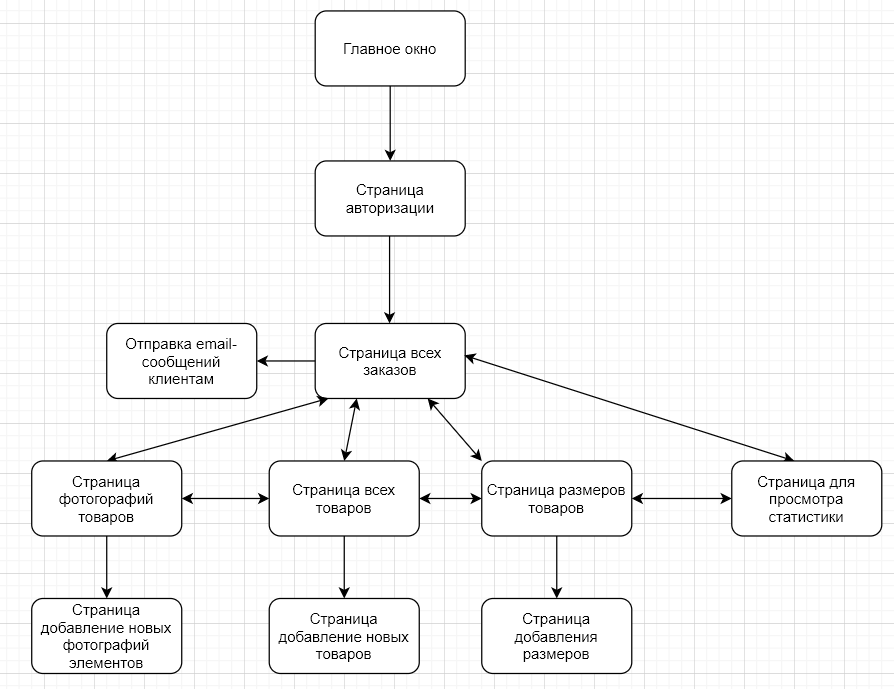


Рисунок 2.5 – Схема структуры окон

При запуске приложения появляется окно авторизации. При успешной авторизации админ попадает на страницу всех заказов, где может отправить email-сообщения клиентам, со страницы всех заказов админ имеет возможность перейти на страницу фотографий товаров, страницу всех товаров, страницу размеров товаров и страницу для просмотра статистики. На странице фотографий товаров админ может добавить новые фотографии, на странице добавления новых товаров админ может добавить новый товар, а на странице добавления размеров админ имеет возможность добавления размеров.

# **3 Разработка функциональной модели и модели данных ПС**

## **3.1 Функции, реализуемые программой**

Функционально приложение поддерживает:

Функции клиента:

* Выполнять регистрацию и авторизацию;
* Создание, редактирование и удаление заказов;
* Просматривать личную статистику;
* Фильтрация товаров;
* Выполнять поисковые запросы.

Функции администратора:

* Поддерживать работу с базой данных;
* Просматривать общую статистику заказанных товаров.

Диаграмма вариантов использование представлена в приложении Б.

## **3.2 Реализация классов**

Данный раздел содержит описание и реализацию некоторых классов приложения.

Класс EFDBContext, листинг которого представлен в приложении И, наследует класс DbContext, необходимый для управления подключением к базе данных.

## **3.3 Реализация работы с базой данных**

Для работы с базой данных были реализованы следующие паттерны: GeneticRepository и UnitOfWork.

GenericRepository — это паттерн проектирования при работе с базами данных, который позволяет отделить конкретные подключения к источникам данных и остальной программой [4].

Диаграмма классов, реализующих паттерн GenericRepository, представлена в приложении В.

Интерфейс репозитория IGenericRepository<T> включает функционал:

* IEnumerable<T> GetAll();
* IEnumerable<T> FindBy(Expression<Func<T, bool>> predicate);
* T Add(T entity);
* T Delete(T entity);
* void Edit(T entity);
* void Save();

Эти методы представляют собой реализацию CRUD и предоставляет асинхронную работу с базой данной.

Данный интерфейс реализует абстрактный класс GenericRepository<T>, в котором реализован весь функционал интерфейса.

Для каждой модели сущности был создан класс, наследующий абстрактный класс (рисунок 3.1).

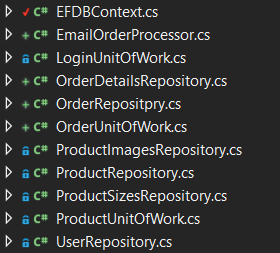


Рисунок 3.1 – Классы, реализующие абстрактный класс

UnitOfWork – это паттерн, который содержит набор репозиториев и гарантирует, что все репозитории будут использовать один и тот же контекст данных[4].

Реализация паттерна и диаграмма компонентов представлены в приложениях Г и Д соответственно.

## **3.4 Реализация паттерна MVVM**

В курсовом проект реализован паттерн MVVM (Model-View-ViewModel), который позволяет отделить логику приложения от визуальной части. Данный паттерн является архитектурным, то есть он задает общую архитектуру приложения [5].

MVVM удобно использовать вместо классического MVC и ему подобных в тех случаях, когда в платформе, на которой ведётся разработка, есть «связывание данных». В шаблонах проектирования MVC/MVP изменения в пользовательском интерфейсе не влияют непосредственно на Mодель, а предварительно идут через Контроллер (англ. Controller) или Presenter. В таких технологиях как WPF и Silverlight есть концепция «связывания данных», позволяющая связывать данные с визуальными элементами в обе стороны. Следовательно, при использовании этого приёма применение модели MVC становится крайне неудобным из-за того, что привязка данных к представлению напрямую не укладывается в концепцию MVC/MVP.

На рисунке 3.2 представлено размещение компонентов.

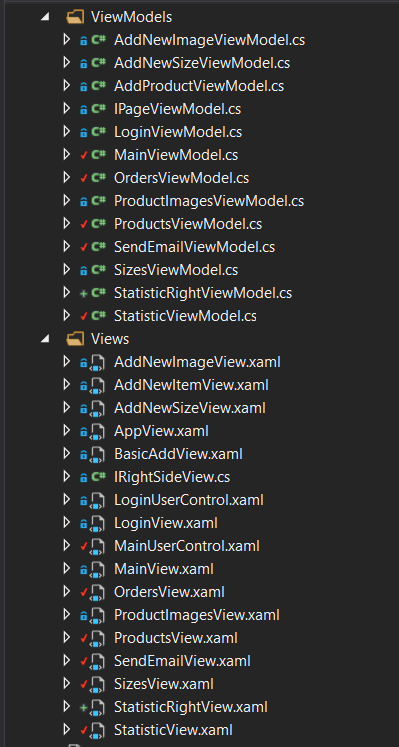


Рисунок 3.2 – Размещение компонентов

Все ViewModel классы реализуют интерфейс INotifyPropertyChanged (рисунок 3.3), уведомляющего клиента об изменении значения свойства.

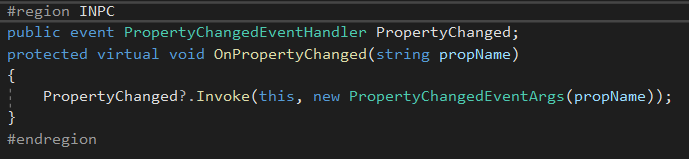


Рисунок 3.3 – Реализация интерфейса INotifyPropertyChanged

Для взаимодействия пользователя и приложения в MVVM используются команды. С этой целью создан класс RelayCommand, реализующий интерфейс ICommand. Листинг находится в приложении Е.

Для каждой странички создан свой класс с суффиксом ViewModel. В данном разделе рассмотрен класс LoginViewModel, листинг которого представлен в приложении Ж.

Класс содержит 2 команды:

* LoginCommand – команда для входа;
* CloseWindowCommand – команда закрытия окна.

Метод SignIn используется для авторизации пользователя. Если все необходимые данные введены, логин сверяется с базой данной. В случае обнаружения пользователя, сверяются хеши паролей, иначе выводится сообщение о некорректности логина. Если пароль введён верно, открывается главное окно приложения, в противном случае выводится сообщение о неправильном пароле. Диаграмма последовательности представлена в приложении К.

## **3.5 Реализация паттерна MVC.**

Основная идея этого паттерна в том, что и контроллер и представление зависят от модели, но модель никак не зависит от этих двух компонент.

Контроллер определяет, какие представление должно быть отображено в данный момент; События представления могут повлиять только на контроллер. Контроллер может повлиять на модель и определить другое представление. Возможно несколько представлений только для одного контроллера.

Контроллер перехватывает событие извне и в соответствии с заложенной в него логикой, реагирует на это событие изменяя модель, посредством вызова соответствующего метода. После изменения Модель использует событие о том что она изменилась, и все подписанные на это события Представления, получив его, обращаются к Модели за обновленными данными, после чего их и отображают.

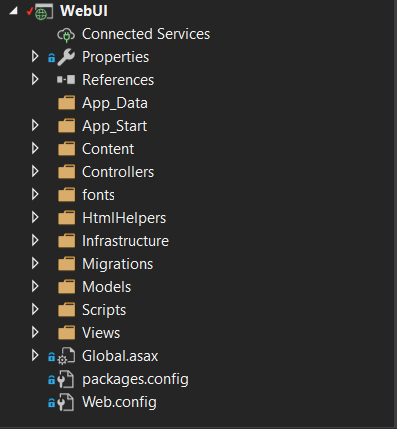


Рисунок 3.4 – Структура проекта ASP.NET MVC

# **4 Тестирование**

При использовании программного средства могут возникнуть такие ситуации, когда введенные пользователем данные приведут к ошибке приложения.

В программе предусмотрена обработка возможных исключений, связанных с работой пользователя. В данном разделе продемонстрированы следующие тесты:

* aвторизация;
* добавление новых элементов в базу данных.

## **4.1 Тест авторизации**

При авторизации может возникнуть ситуация, когда пользователь не введёт необходимые данные. В таком случае выводится это (рисунок 4.1):



Рисунок 4.1 – Тест авторизации при введённых неправильных данных

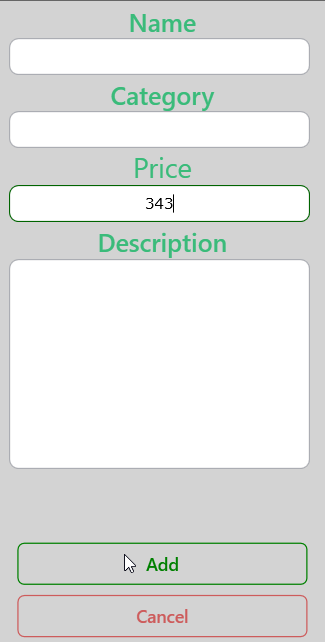


Рисунок 4.2 – Тест добавления товара при не всех веденных данных.

Если пароль не введен, то появляется следующее (рисунок 4.3):

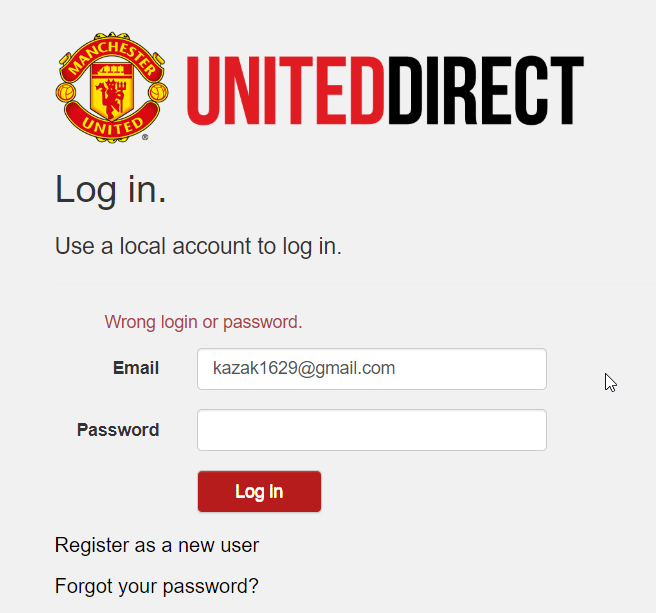


Рисунок 4.3 – Тест авторизации при некорректном пароле

При не введенном пароле пользователю выводится (рисунок 4.4):

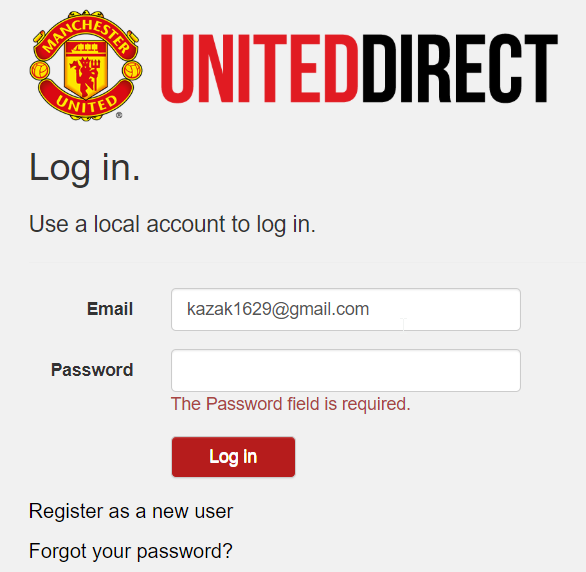


Рисунок 4.4 – Тест авторизации при не введенном пароле

При успешной авторизации пользователь попадает на главную страницу.

## **4.2 Тест регистрации**

При регистрации может возникнуть ситуация, когда пользователь не введёт необходимые данные, в этом случае, как и при авторизации, выводится сообщение (рисунок 4.5):

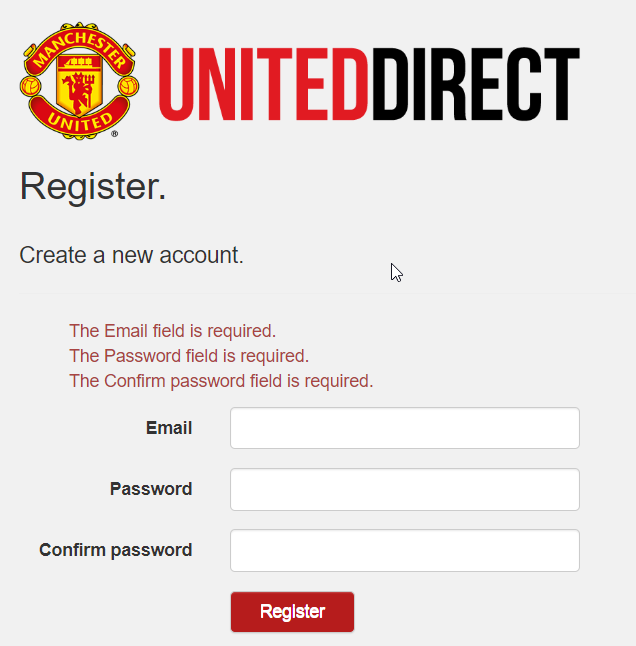


Рисунок 4.5 – Тест регистрации при не введённых данных

# **ЗАКЛЮЧЕНИЕ**

В ходе выполнения курсового проекта был изучен теоретический материал по теме проекта, рассмотрены аналоги, выявлены их преимущества и недостатки.

Для реализации программного средства были изучены и применены технологии такие, как WPF, Entity Framework, Microsoft SQL Server, ASP.NET MVC. Само приложение было реализовано на языке C# в среде Visual Studio. Для структуры настольного проекта применён архитектурный паттерн MVVM, для веб приложения архитектурный паттерн MVC, для работы с базой данной – Repository и UnitOfWork.

Цель курсового проекта – создание приложения «MU-Store», предоставляющее сайт для клиентов, в котором они могут выбрать и купить интересующие их товары: футбольную атрибутику футбольного клуба Манчестер Юнайтед, а также[предоставляющее] десктопное приложение для администратора, который может управлять базой данных сайта и тем самым обновлять данные на сайте, а также просматривать статистику заказов.

Главной задачей являлось создание максимально удобного и понятного приложения, для этого была спроектирована структура окон для комфортного использования.

Функционально проект поддерживает:

* Регистрацию и авторизацию;
* Отправку email-сообщений;
* Просмотр товаров;
* Поисковые запросы;
* Добавление и удаление товаров;
* Добавление и удаление характеристик товаров;
* Создание и удаление заказов;

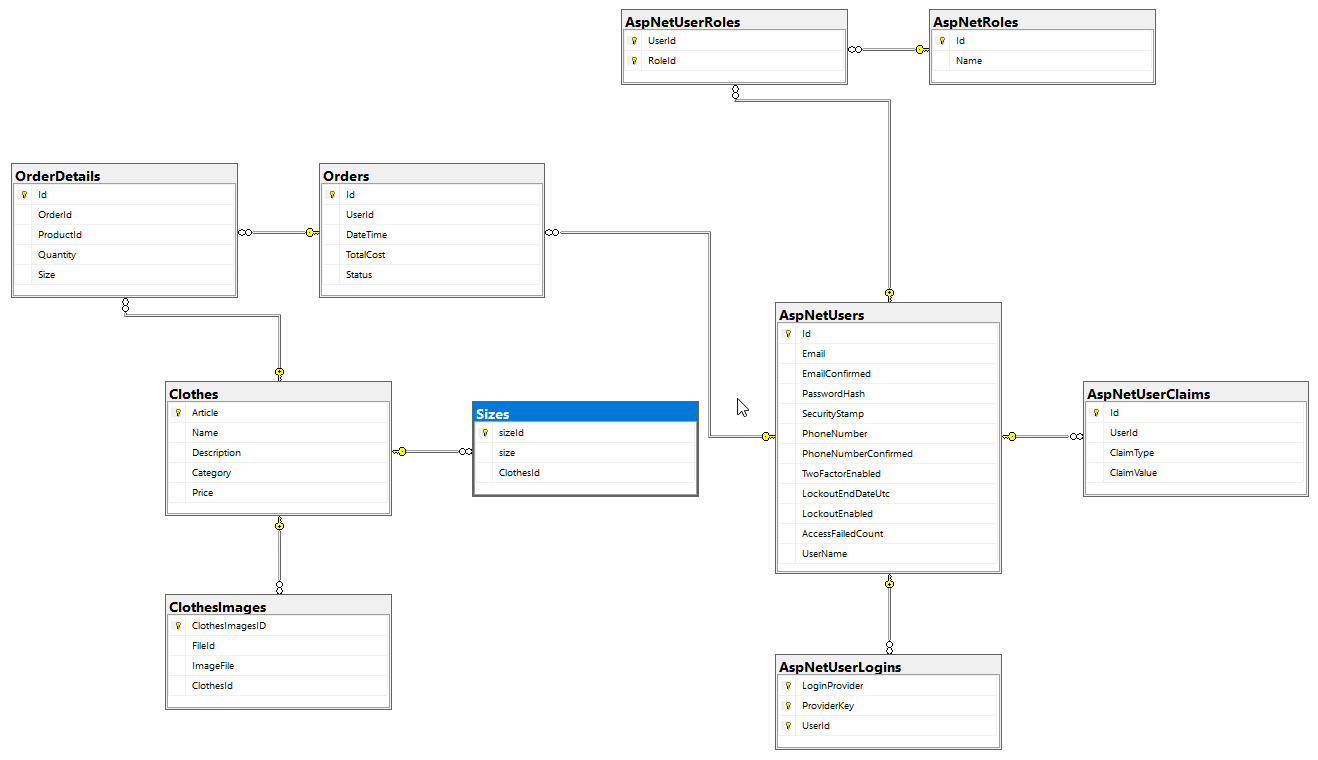
Все функции протестированы и выполняются корректно.

# **Список используемых источников**

1. Натан А. WPF 4. Подробное руководство. - Пер. с англ. - СПб.: Символ-Плюс, 2011. - 880 с.
2. Сайт о программировании Metanit: Введение в Entity Framework Core [Электронный ресурс] / metanit.com – Режим доступа: https://metanit.com/sharp/entityframeworkcore/1.1.php
3. Сайт о программировании Metanit: Введение в MS SQL Server [Электронный ресурс] / metanit.com – Режим доступа: <https://metanit.com/sql/sqlserver/1.1.php>
4. Пацей, Н.В. Курс лекций по языку программирования C# / Н.В. Пацей. – Минск: БГТУ, 2016. – 175 с.
5. Сайт о программировании Metanit: Определение паттерна MVVM [Электронный ресурс] / metanit.com – Режим доступа: <https://metanit.com/sharp/wpf/22.1.php>
6. Мэтью Макдональд – WPF: Windows Presentation Foundation в .NET 4.0 с примерами на C# / М. Макдональд. – Вильямс, 2011. – 1024 с.

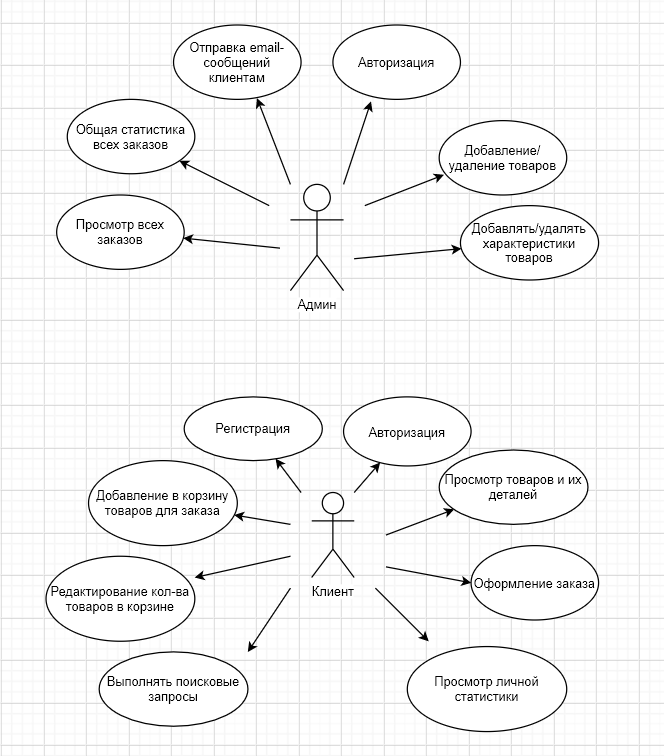
# **ПРИЛОЖЕНИЕ А**

Структура базы данных.



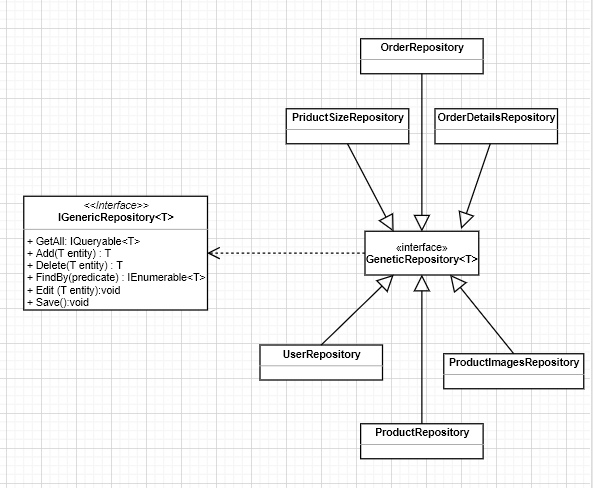
# **ПРИЛОЖЕНИЕ Б**

Диаграмма вариантов использования.

****

# **ПРИЛОЖЕНИЕ В**

Диаграмма классов, реализующих паттерн Repository.



# **ПРИЛОЖЕНИЕ Г**

Листинг 1. Реализация класса ProductUnitOfWork.

using Domain.Abstract;

using Domain.Entities;

using Ninject;

namespace Domain.Concrete

{

public class ProductUnitOfWork : IProductUnitOfWork

{

private GenericRepository<Size> \_sizesRepository;

private GenericRepository<Image> \_imagesRepository;

private GenericRepository<Product> \_productsRepository;

public ProductUnitOfWork([Named("Products")] GenericRepository<Product> productsRepo,

[Named("Images")] GenericRepository<Image> imagesRepo,

[Named("Sizes")] GenericRepository<Size> sizesRepo)

{

\_sizesRepository = sizesRepo;

\_imagesRepository = imagesRepo;

\_productsRepository = productsRepo;

}

public GenericRepository<Size> Sizes => \_sizesRepository;

public GenericRepository<Image> Images => \_imagesRepository;

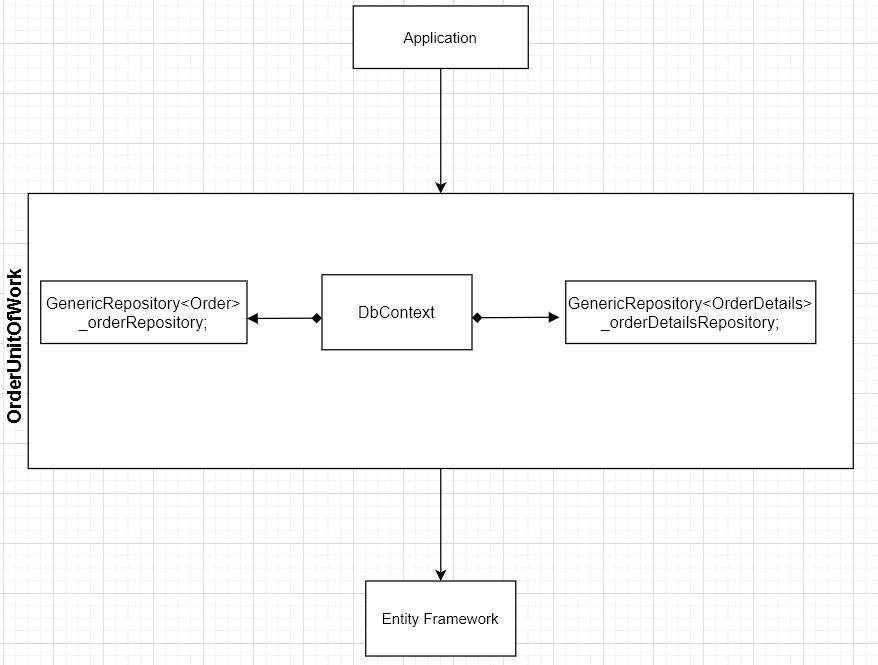
public GenericRepository<Product> Products => \_productsRepository;

}

}

# **ПРИЛОЖЕНИЕ Д**

Диаграмма компонентов с использованием паттерна Repository в связке с UnitOfWork паттерном.



# **ПРИЛОЖЕНИЕ Е**

Листинг 2. Реализация класса RelayCommand.

using System;

using System.Windows.Input;

namespace Models.Commands

{

public class RelayCommand : ICommand

{

private Action<object> execute;

private Func<object, bool> canExecute;

public event EventHandler CanExecuteChanged

{

add { CommandManager.RequerySuggested += value; }

remove { CommandManager.RequerySuggested -= value; }

}

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

{

this.execute = execute;

this.canExecute = canExecute;

}

public bool CanExecute(object parameter) => canExecute == null || canExecute(parameter);

public void Execute(object parameter) => execute(parameter);

}

}

# **ПРИЛОЖЕНИЕ Ж**

Листинг 3. Реализация класса LoginViewModel.

public class LoginViewModel : INotifyPropertyChanged, IPageViewModel

{

#region INPC

public event PropertyChangedEventHandler PropertyChanged;

protected virtual void OnPropertyChanged(string propName)

{

PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propName));

}

#endregion

private ILoginUnitOfWork \_loginUnitOfWork;

public IEnumerable<AspNetRoles> AspNetRoles { get; set; }

public IEnumerable<IdentityUser> IdentityUser { get; set; }

public IEnumerable<AspNetUserRoles> AspNetUserRoles { get; set; }

public LoginViewModel(ILoginUnitOfWork loginUnitOfWork)

{

CloseWindowCommand = new RelayCommand(x => CloseWindow((ICloseable)x));

\_loginUnitOfWork = loginUnitOfWork;

IdentityUser = \_loginUnitOfWork.Users.GetAll().ToList();

}

#region binding

private string \_login;

public string Login

{

get { return \_login; }

set

{

if (!string.Equals(\_login, value))

{

\_login = value;

OnPropertyChanged("Login");

}

}

}

#endregion

#region LoginCommand

private RelayCommand \_loginCommand;

public RelayCommand LoginCommand

{

get

{

return \_loginCommand ?? (\_loginCommand = new RelayCommand(param => SingIn(param), param => LoginCanExecute(param)));

}

}

public bool LoginCanExecute(object param)

{

if(!string.IsNullOrEmpty(\_login) && !string.IsNullOrEmpty(param.ToString()))

{

return true;

}

return false;

}

private void SingIn(object param)

{

var passwordBox = param as PasswordBox;

if (passwordBox == null)

{

return;

}

var password = passwordBox.Password;

var manager = IdentityUser.Where(user => user.Email == \_login && VerifyPassword.VerifyHashedPassword(user.PasswordHash, password)).FirstOrDefault();

if (manager == null)

{

MessageBox.Show("Wrong login or password.", "Login error", MessageBoxButton.OK, MessageBoxImage.Error, MessageBoxResult.OK, MessageBoxOptions.ServiceNotification);

return;

}

#region

//var isRoleExist = AspNetUserRoles.Where(userRole => userRole.UserId == manager.UserId).FirstOrDefault();

//if (isRoleExist == null)

//{

// MessageBox.Show("You do not have access to the application.", "Login error", MessageBoxButton.OK, MessageBoxImage.Error, MessageBoxResult.OK, MessageBoxOptions.ServiceNotification);

// return;

//}

//var concreteRole = AspNetRoles.Where(role => role.UserId == isRoleExist.RoleId).FirstOrDefault();

//if (concreteRole == null)

//{

// MessageBox.Show("You do not have access to the application.", "Login error", MessageBoxButton.OK, MessageBoxImage.Error, MessageBoxResult.OK, MessageBoxOptions.ServiceNotification);

// return;

//}

#endregion

///<summary>

///if login is successful pass true

///</summary>

ViewManager.GetInstance().ChangeViewModel(true);

}

#endregion

#region CloseWindowCommand

public RelayCommand \_closeWindowCommand;

public RelayCommand CloseWindowCommand { get; private set; }

private void CloseWindow(ICloseable window)

{

if (window != null)

{

window.Close();

}

**ПРИЛОЖЕНИЕ З**

Листинг 4. Реализация класса EFDBContext.

public class EFDBContext : DbContext

{

public DbSet<Product> Products { get; set; }

public DbSet<Image> ClothesImages { get; set; }

public DbSet<Size> Sizes { get; set; }

public DbSet<AspNetUserRoles> AspNetUserRoles { get; set; }

public DbSet<IdentityUser> IdentityUser { get; set; }

public DbSet<AspNetRoles> AspNetRoles { get; set; }

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

public DbSet<OrderDetails> OrdesDetails { get; set; }

protected override void OnModelCreating(DbModelBuilder modelBuilder)

{

base.OnModelCreating(modelBuilder);

modelBuilder.Entity<IdentityUser>().Ignore(c => c.AccessFailedCount)

.Ignore(c => c.Claims)

.Ignore(c => c.LockoutEnabled)

.Ignore(c => c.LockoutEndDateUtc)

.Ignore(c => c.PhoneNumber)

.Ignore(c => c.Logins)

.Ignore(c => c.PhoneNumberConfirmed)

.Ignore(c => c.TwoFactorEnabled);

}

}

# **ПРИЛОЖЕНИЕ И**

Диаграмма последовательности.

