**Содержание**

[Введение 2](#_Toc104253441)

[1. Аналитический обзор литературы 3](#_Toc104253442)

[1.1. Аналитический обзор источников 3](#_Toc104253443)

[1.2. Обзор аналогов 3](#_Toc104253444)

[2. Анализ требований к программному средству и разработка функциональных требований 5](#_Toc104253445)

[2.1. Спецификация функциональных требований 5](#_Toc104253446)

[2.1.1. Требования к параметрам технических и программных средств 5](#_Toc104253447)

[2.1.2. Требования к организации входных данных 5](#_Toc104253448)

[2.1.3. Требования к надежности 5](#_Toc104253449)

[2.2. Описание функциональности программного средства 6](#_Toc104253450)

[3. Проектирование программного средства 7](#_Toc104253451)

[3.1. Проектирование базы данных 7](#_Toc104253452)

[3.2. Проектирование архитектуры проекта 12](#_Toc104253453)

[4. Создание программного средства 15](#_Toc104253454)

[4.1. Класс CommandDelegate 15](#_Toc104253455)

[4.2. Класс BoolToVisibilityConverter 15](#_Toc104253456)

[4.3. Класс MathConverter 15](#_Toc104253457)

[4.4. Класс MultiValueEqualityConverter 15](#_Toc104253458)

[4.5. Класс ScrollViewerExtensions 15](#_Toc104253459)

[4.6. Класс SaltedHash 15](#_Toc104253460)

[4.7. Классы Character, CharacterClass, Container, ContainerItem, Item, ItemType, Message, TradeOffer, User, UserRole, UserStatus 16](#_Toc104253461)

[4.8. Класс GameDBContainer 16](#_Toc104253462)

[4.9. Класс WindowMode 16](#_Toc104253463)

[4.10. Класс EnumExtensions 16](#_Toc104253464)

[4.11. Классы ViewModel 16](#_Toc104253465)

[4.12. Валидация данных 16](#_Toc104253466)

[5. Тестирование 17](#_Toc104253467)

[6. Методика использования программного средства 28](#_Toc104253468)

[Заключение 38](#_Toc104253469)

[СПИСОК ЛИТЕРАТУРЫ 39](#_Toc104253470)

[Приложения 40](#_Toc104253471)

[Приложение А 40](#_Toc104253472)

[Приложение Б 41](#_Toc104253473)

[Приложение В 45](#_Toc104253474)

[Приложение Г 47](#_Toc104253475)

[Приложение Д 48](#_Toc104253476)

# Введение

Компьютерная ролевая игра – это программа на ПК, позволяющая пользователю провести свой досуг, коллекционируя вещи, улучшая своего персонажа, и накапливая игровую валюту. Представляет собой игровой лаунчер, написанный с помощью библиотеки WPF, где игрок может создавать персонажа, выбрать активного персонажа, изменять его экипировку из имеющихся предметов инвентаря. Однако, чтобы получить возможность сделать это, игроку придется поучаствовать в несложном игровом процессе по сбору валюты и прохождения игровой локации. Эта часть представлена в виде приложения, написанного на платформе Unity в жанре «инди».

Основной идеей подобного приложения является предоставление игрового досуга пользователю, а также возможность развития собственного персонажа и накопления многочисленных игровых ресурсов, таких как вещи, оружие и валюта. Получить предметы в инвентарь своего персонажа вы можете, покупая их за игровую валюту в лавке, однако также игрокам будет доступна возможность составлять предложения обмена к другим игрокам, которые те в свою очередь могут принять или отказать. Но и это еще не все, чтобы сделать предложение нужна будет честная сделка, а значит и потребуется возможность договориться о ней с другим игроком, в данном приложении также присутствует возможность общаться с другими пользователями, начинать чаты и отправлять им свои сообщения, пусть для вас не станет одиноким ваше времяпрепровождение если вы этого пожелаете.

Цель моего курсового проекта - разработать приложение для увлекательного времяпрепровождения, которое затянет пользователя надолго, а также испытать удачу, проходя локацию и повышая свой уровень, или приобретать и обменивать вещи и товары, при этом имея возможность общаться и писать сообщения с другими игроками.

Язык разработки проекта – C#. При выполнении курсового проекта будут использованы принципы и приемы ООП. Также будут использоваться технологии Windows Presentation Foundation(WPF).

1. **Аналитический обзор литературы**
   1. **Аналитический обзор источников**

В ходе подготовки пояснительной записки была изучена специальная техническая, учебно-методическая и справочная литература, статьи и материалы, опубликованные в сети интернет.

Принцип создания окон для регистрации и авторизации был взят из статьи «WPF – система авторизации и регистрации». В статье были рассмотрены алгоритм работы системы авторизации и регистрации и пример создания окон регистрации и авторизации.

Алгоритм работы окна авторизации:

1. выполняется проверка заполняемости всех полей;
2. выполняется проверка существования записи в базе данных;
3. если запись существует, пользователь авторизуется.

Алгоритм работы окна регистрации:

1. выполняется проверка заполняемости всех полей;
2. выполняется проверка на допустимость введенных данных
3. выполняется проверка существования записи с таким же логином в базе данных;
4. если записи не существует, добавляется новая запись.

Принцип работы с SQL были получены из статьи «Подключение к базе данных». В статье было рассмотрено подключение необходимых библиотек, работа с SQL.

Дополнительная информации о принципах работы с WPF была получена из интернет-источника «Metanit», содержащего практические советы по работе с технологией.

* 1. **Обзор аналогов**

Для создания принципиально нового решения в виде программного продукта, используемого в качестве игрового лаунчера для игры в жанре «инди», необходимо проанализировать уже существующее программные средства в данной сфере. Анализ достоинств и недостатков этих аналогов позволит сформировать требования к проектируемому программному средству, учитывающие опыт существующих разработок и внести в них улучшения или изменения.

В качестве исследуемых аналогов были выбраны такие программные продукты, которые можно отнести к группе игр, игровых лаунчеров и приложений, предназначенных для непосредственного управления инвентарем пользователя, как наиболее близкие по области применения к разрабатываемому программному средству. Источником информации послужили электронные базы в сети Интернет.

В результате поиска были обнаружены следующие ресурсы:

* «Dark souls» – компьютерная игра в жанре action/RPG, разработанная японской компанией From Software и выпущенная в 2011 году для PlayStation 3 и Xbox 360. Действие игры разворачивается в вымышленном королевстве Лордран, напоминающем средневековую Европу. Лордран поражен проклятием, обращающим его жителей в нежить; на главного героя, также живого мертвеца, возлагается задача заново разжечь священный огонь и восстановить порядок вещей. Игровой процесс Dark Souls завязан на исследовании мира, нахождении ключевых предметов и уничтожении боссов; игру отличает высокий уровень сложности — подразумевается, что игровой персонаж будет раз за разом погибать, а игрок — учиться на своих ошибках. У данного программного средства были позаимствованы такие идеи, как:

1. меню инвентаря персонажа.
2. возможность экипировки предметов.
3. панель и система покупки игровых предметов.

* «World of Warcraft» – массовая многопользовательская ролевая онлайн-игра, разработанная и издаваемая компанией Blizzard Entertainment. Действие World of Warcraft происходит в фэнтезийной вселенной Warcraft. Игра тесно связана с предыдущими играми серии — стратегиями в реальном времени; каждый игрок управляет одним персонажем и может взаимодействовать с другими игроками в общем виртуальном мире. Благодаря игровому лаунчеру данной игры, была реализована панель и система создания персонажа, а также частично позаимствовано классовое деление персонажей и создано общее понимание итогового продукта
* Десктопное приложение «Steam» – онлайн-сервис цифрового распространения компьютерных игр и программ, разработанный и поддерживаемый компанией Valve. Steam выполняет роль средства технической защиты авторских прав, платформы для многопользовательских игр и потокового вещания, а также социальной сети для игроков. Программный клиент Steam также обеспечивает установку и регулярное обновление игр, облачные сохранения игр, текстовую и голосовую связь между игроками. Данное программное средство стало основой для таких фундаментальных функций приложения, как:

1. мессенджер, с помощью которого пользователи могут выбирать собеседника, отправлять ему сообщения и читать сформировавшийся чат.
2. система предложений обмена – известная и популярная функция данного онлайн-сервиса, которая послужила качественным примером для реализации подобного функционала.

# Анализ требований к программному средству и разработка функциональных требований

## Спецификация функциональных требований

* + 1. **Требования к параметрам технических и программных средств**

Программное средство может функционировать на ЭВМ со следующими минимальными характеристиками:

1. процессор Intel® Core™ i3 с тактовой частотой 2.4 ГГц и лучше;
2. оперативная память 1 ГБ и более;
3. операционная система: Windows 10.
   * 1. **Требования к организации входных данных**

Входные данные для программного средства должны быть представлены в виде вводимого пользователем с клавиатуры текста: логин, пароль. После аутентификации пользователю предоставляется возможность работы с системой.

Данные, вводимые пользователем, должны проверяться на корректность в процессе аутентификации.

* + 1. **Требования к надежности**

Надежное (устойчивое) функционирование программы должно быть обеспечено выполнением совокупности организационно-технических мероприятий, перечень которых приведен ниже:

1. надежное (устойчивое) функционирование программы должно быть обеспечено выполнением совокупности организационно-технических мероприятий, перечень которых приведен ниже:
2. организацией бесперебойного питания технических средств,   
   выполнением требований «ГОСТ 31078-2002. Защита информации. Испытания программных средств на наличие компьютерных вирусов. Типовое руководство»;
3. необходимым уровнем квалификации сотрудников профильных подразделений.

Время восстановления после отказа, вызванного сбоем электропитания технических средств (иными внешними факторами), не фатальным сбоем (не крахом) операционной системы, не должно превышать времени, необходимого на перезагрузку операционной системы и запуск программы, при условии соблюдения условий эксплуатации технических и программных средств.

Время восстановления после отказа, вызванного неисправностью технических средств, фатальным сбоем (крахом) операционной системы, не должно превышать времени, требуемого на устранение неисправностей технических средств и переустановки программных средств.

## Описание функциональности программного средства

На основании анализа исходных данных для проектируемого программного средства можем выделить, что основной целью является создание качественного программного продукта, позволяющего решить существующие проблемы таких как:

* создание игрового персонажа;
* развитие персонажа в ходе игрового процесса;
* получение игровой валюты в ходе игрового процесса;
* приобретение игровых предметов для персонажа;
* обмен игровых предметов с другими пользователями;
* отправка сообщений другим пользователям.

Диаграмма классов представлена в Приложении А.

# Проектирование программного средства

## Проектирование базы данных

Для данного курсового проекта была спроектирована база данных, которая получила название «GameDB». Для управления базой данных используется ПО Microsoft SQL Server 2018. Логическая схема данных представлена на рисунке 3.1.

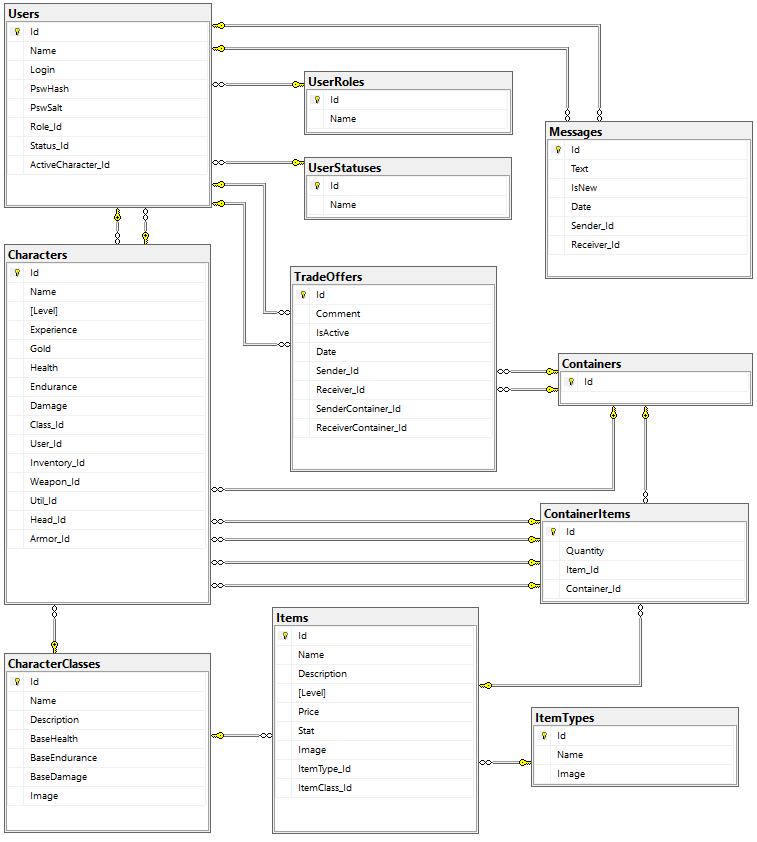
****

Рисунок 3.1 – Логическая схема данных

В базе данных находится 11 таблиц. В таблице Users хранятся данные для идентификации пользователей, зарегистрированные в приложении, UserRoles – роли пользователей, UserStatuses – статусы пользователей, Characters – персонажы, которые создаются пользователями, Items – список игровых предметов, CharacterClasses – классы персонажей, ItemTypes – типы предметов, Containers – список контейнеров, для хранения экземпляров предметов инвентаря, ContainerItems – предметы инвентаря персонажей, Messages – сообщения отправленные пользователями, и TradeOffers, которая содержит созданные пользователям предложения обмена.

Таблица Users состоит из следующих столбцов:

* Id;
* Name;
* Login;
* PswHash;
* PswSalt;
* Role\_Id;
* Status\_Id;
* ActiveCharacter\_Id.

Столбец Id содержит идентификатор пользователя, Name содержит имя пользователя, Login – его логин для идентифицирования и авторизации, PswHash – хешированный алгоритмом SHA256 пароль, Salt – соль, используемая при хешировании пароля, Role\_Id – внешний ключ, указывающий на роль пользователя, Status\_Id – внешний ключ, указывающий на статус пользователя, ActiveCharacter\_Id – внешний ключ, указывающий на активного персонажа пользователя.

Таблица UserRoles состоит из столбцов:

* Id;
* Name.

Столбец Id содержит идентификатор роли пользователя, Name – название роли пользователя.

Таблица UserStatuses состоит из столбцов:

* Id;
* Name.

Столбец Id содержит идентификатор статуса пользователя, Name – название статуса пользователя.

Таблица Characters состоит из следующих столбцов:

* Id;
* Name;
* Level;
* Experience;
* Gold;
* Health;
* Endurance;
* Damage
* Class\_Id;
* User\_Id;
* Inventory\_Id;
* Weapon\_Id;
* Util\_Id;
* Head\_Id;
* Armor\_Id.

Столбец Id содержит идентификатор персонажа, Name содержит имя персонажа, Level – хранит текущий уровень персонажа, Experience – содержит количество опыта до следующего уровня, Gold – хранит количество игровой валюты у персонажа, Health – количество здоровья персонажа, Endurance – количество выносливости персонажа, Damage – количество урона этого персонажа, Class\_Id – внешний ключ, указывающий на класс персонажа, User\_Id – внешний ключ, указывающий на владельца этого персонажа, Inventory\_Id – внешний ключ, указывающий на контейнер инвентаря этого персонажа, Weapon\_Id – внешний ключ, указывающий на экипированное оружие, Util\_Id – внешний ключ, указывающий на экипированный расходуемый предмет, Head\_Id – внешний ключ, указывающий на экипированный головной убор, Armor\_Id – внешний ключ, указывающий на экипированную броню персонажа.

Таблица Items состоит из следующих столбцов:

* Id;
* Name;
* Description;
* Level;
* Price;
* Stat;
* Image;
* ItemType\_Id;
* ItemClass\_Id.

Столбец Id содержит идентификатор предмета, Name содержит название предмета, Description – его описание, Level – уровень предмета, Price – стоимость предмета, Stat – значение характеристики предмета, зависящей от типа предмета, Image – путь к изображению этого предмета, ItemType\_Id – внешний ключ, указывающий на тип предмета, ItemClass\_Id – внешний ключ, указывающий на класс предмета, соответствующий классам персонажей.

Таблица CharacterClasses состоит из следующих столбцов:

* Id;
* Name;
* Description;
* BaseHealth;
* BaseEndurance;
* BaseDamage;
* Image.

Столбец Id содержит идентификатор класса персонажа, Name содержит название класса персонажа, Description – его описание, BaseHealth – базовое здоровье у данного класса персонажа, BaseEndurance – базовая выносливость у этого класса персонажа, BaseDamage – базовый урон у этого класса персонажа, Image – путь к изображению этого класса персонажа.

Таблица ItemTypes состоит из следующих столбцов:

* Id;
* Name;
* Image.

Столбец Id содержит идентификатор типа предмета, Name содержит название типа предмета, Image – путь к изображению этого типа предмета.

Таблица Containers состоит из следующих столбцов:

* Id.

Столбец Id содержит идентификатор контейнера.

Таблица ContainerItems состоит из следующих столбцов:

* Id;
* Quantity;
* Item\_Id;
* Container\_Id.

Столбец Id содержит идентификатор элемента контейнера, Quantity – хранит количество предметов этого элемента контейнера, Item\_Id – внешний ключ, указывающий на предмет из данного элемента контейнера, Container\_Id – внешний ключ, указывающий на контейнер, к которому привязан данный элемент контейнера.

Таблица Messages состоит из следующих столбцов:

* Id;
* Text;
* IsNew;
* Date
* Sender\_Id;
* Receiver\_Id.

Столбец Id содержит идентификатор сообщения, Text содержит текст сообщения, IsNew – логический показатель на то, является данное сообщение непрочитанным получателем, Date – дата отправки сообщения, Sender\_Id – внешний ключ, указывающий на отправителя данного сообщения, Receiver\_Id – внешний ключ, указывающий на получателя данного сообщения.

Таблица TradeOffers состоит из следующих столбцов:

* Id;
* Comment;
* IsActive;
* Date
* Sender\_Id;
* Receiver\_Id;
* SenderContainer\_Id;
* ReceiverContainer\_Id.

Столбец Id содержит идентификатор предложения обмена, Comment содержит комментарий к предложению обмена, IsActive – логический показатель на то, является данное предложение активным (не принятым или не отмененным), Date – дата отправки предложения обмена, Sender\_Id – внешний ключ, указывающий на отправителя данного предложения обмена, Receiver\_Id – внешний ключ, указывающий на получателя данного предложения обмена, SenderContainer\_Id – внешний ключ, указывающий на список предметов отправителя, ReceiverContainer\_Id – внешний ключ, указывающий на список предметов. Скрипты создания базы данных приложения представлены в Приложении Б.

## Проектирование архитектуры проекта

На рисунке 3.2 изображена диаграмма вариантов использования приложения.

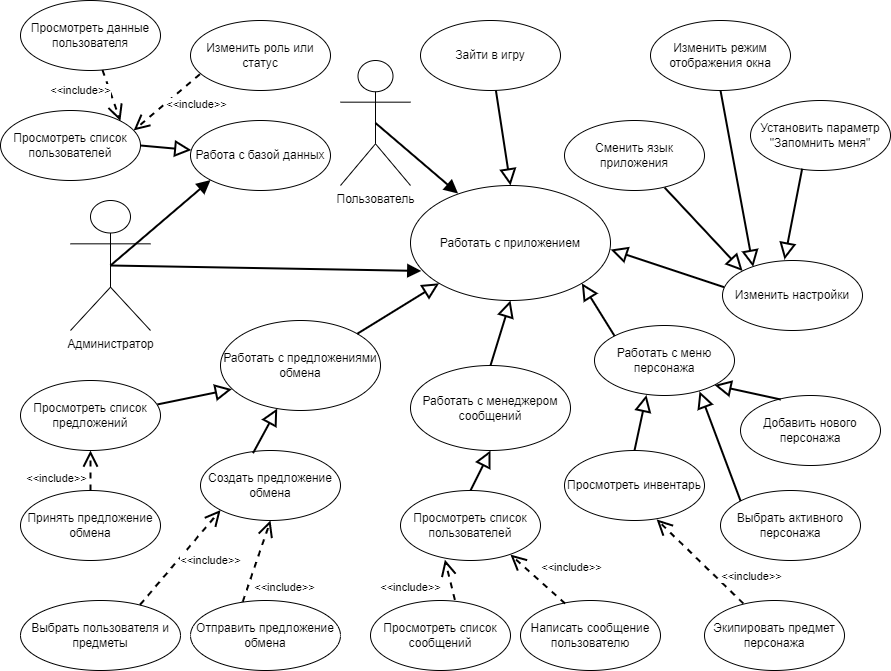


Рисунок 3.2 – Диаграмма использования

На рисунке 3.3 была отображена диаграмма последовательности для авторизации. На диаграмме последовательности отображаются только те объекты, которые непосредственно принимают участие во взаимодействии.

****

Рисунок 3.3 – Диаграмма последовательности для авторизации

На рисунке 3.4 изображена диаграмма компонентов приложения.



Рисунок 3.4 – Диаграмма компонентов

На рисунке 3.5 представлена блок-схема для алгоритма регистрации пользователя.



Рисунок 3.5 – Схема алгоритма регистрации

Логика перехода между страницами реализована в классе MainViewModel. Листинг класса представлен в Приложении В. Для получения и записи данных в БД используется класс GameDBContainer, который создает достаточный уровень абстракции для быстро доступа к БД. Для создания подключения и работы с БД используется технология Entity Framework Core. Для администрирования приложения путем доступа к БД был создан пользователь с логином admin, обладающий правами администратора.

При разработке приложения был реализован архитектурный паттерн MVVM. Он используется для разделения модели и ее представления, что необходимо для их изменения отдельно друг от друга.

# Создание программного средства

В данной главе детальнее разбираются некоторые классы приложения, которые используются для выполнения основных операций.

## Класс CommandDelegate

Данный класс позволяет делегировать командную логику методами, передаваемыми в качестве параметров, и позволяет View связывать команды с объектами, которые не являются частью дерева элементов. Листинг класса представлен в Приложении Г.

## Класс BoolToVisibilityConverter

Этот класс реализует интерфейс IValueConverter и используется в качестве конвертера значений, который позволяет View привязать свойство Visibility визуального элемента к логическому свойству из контекста данных.

## Класс MathConverter

Этот класс реализует интерфейс IMultiValueConverter и используется в качестве конвертера массива значений, который позволяет View получить значение, передав в качестве параметра операцию, которая будет применена к каждому из исходных значений.

## Класс MultiValueEqualityConverter

Этот класс реализует интерфейс IMultiValueConverter и используется в качестве конвертера массива значений, который позволяет View сравнить на полное равенство переданных исходных объектов.

## Класс ScrollViewerExtensions

Данный класс содержит свойство зависимостей, которое может быть применено для элемента ScrollViewer, которое позволит ему оставаться в прокрученном вниз положении.

## Класс SaltedHash

Этот класс содержит методы для генерации соли, определения, является ли пароль, введенным пользователем, тем, который хранится в базе данных приложения, и для хеширования паролей. Листинг класса представлен в Приложении Д.

## Классы Character, CharacterClass, Container, ContainerItem, Item, ItemType, Message, TradeOffer, User, UserRole, UserStatus

Данные классы представляют из себя модели сущностей, содержащие поля, аналогичные тем, которые располагаются в базе данных приложения.

## Класс GameDBContainer

В любом приложении, работающим с БД через Entity Framework Core, необходимо использовать контекст (класс производный от DbContext) и набор данных DbSet, через который можно взаимодействовать с таблицами из БД. Данный класс является реализацией паттерна UnitOfWork для приложения. Паттерн UnitofWork позволяет упростить работу с различными репозиториями и дает уверенность, что все репозитории будут использовать один и тот же контекст данных. Репозиторий позволяет абстрагироваться от конкретных подключений к источникам данных, с которыми работает программа, и является промежуточным звеном между классами, непосредственно взаимодействующими с данными, и остальной программой.

## Класс WindowMode

Данный класс используется для хранения в настройках приложения пользователя данных об используемом состоянии окна WindowState и стиле окна WindowStyle.

## Класс EnumExtensions

Данный статический класс инкапсулирует методы расширения для используемых в приложении перечислений, таких как получение модели, привязанной к этому экземпляру перечисления или получения атрибутов, указанных в определении этого перечисления

## Классы ViewModel

Классы с суффиксом ViewModel содержат: закрытые поля, открытые свойства и команды, привязываемые к элементам управления на страницах View. ViewModel также содержит логику по получению данных из модели, которые потом передаются в представление. И также VewModel определяет логику по обновлению данных в модели.

## Валидация данных

Чтобы избежать ошибок при вводе данных пользователем в поля приложения, используются ограничения, накладываемые на команды, выполняемые при нажатии кнопок, подтверждающие введенные в поля данные: кнопки не доступны до того момента, пока в поля не будут внесены верные данные. В некоторых окнах предусмотрен вывод предупреждающих сообщений.

# Тестирование

Основной целью тестирования приложения было стремление доказать невозможность введения пользователем данных, которые бы могли привести приложение в неработоспособное состояние. Были проведены:

* тесты на валидацию данных при авторизации и регистрации;
* тесты на невозможность покупки предмета из магазина без активного персонажа;
* тесты на невозможность покупки большего количества предметов из магазина чем количество валюты у персонажа;
* тесты на невозможность экипировки предмета не того же класса с персонажем или большего уровня чем уровень персонажа;
* тесты на невозможность добавления большего количества предметов в предложение обмена чем есть у персонажа пользователя;
* тесты на невозможность подтверждения предложения обмена, если оно содержит в себе неверные данные об инвентаре персонажа.

Данные, которые пользователь будет вводить на этапе авторизации будут проверяться на допустимость с помощью класса валидатора, представленного в листинге 5.1.

using System.Text.RegularExpressions;

namespace GameWPF.Util.Validation

{

public class UserValidator

{

public static bool IsPasswordValid(string psw)

=> Regex.IsMatch(psw, "^(?=.\*\\d)(?=.\*[a-z])(?=.\*[A-Z]).{8,}$");

public static bool IsLoginValid(string login)

=> Regex.IsMatch(login, "^[A-Za-z]\\w{5,20}$");

public static bool IsNameValid(string name)

=> Regex.IsMatch(name, "^(\\p{L})+([(. )'-](\\p{L})+)\*$");

}

}

Листинг 5.1 – класс UserValidator, имеющий статические методы, предназначенные для проверки данных пользователя на допустимость

При вводе некорректного пароля будет отображаться подсказка со списком требований. Такая ситуация продемонстрирована на рисунке 5.1.

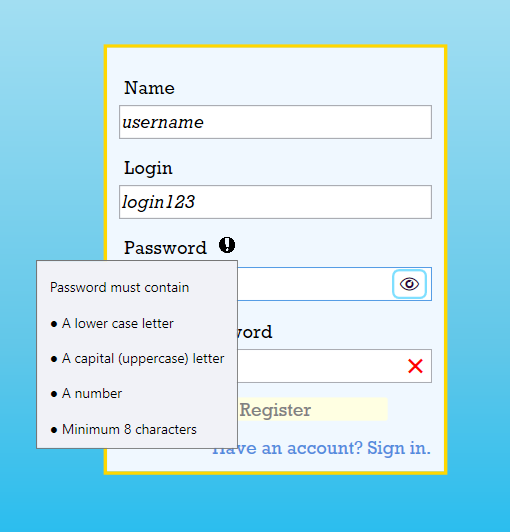


Рисунок 5.1 – Подсказка о требованиях вводимого пароля

В момент регистрации возможна ситуация, когда пользователь вводит уже существующий логин. Обработка данной ситуации продемонстрирована на рисунке 5.2.

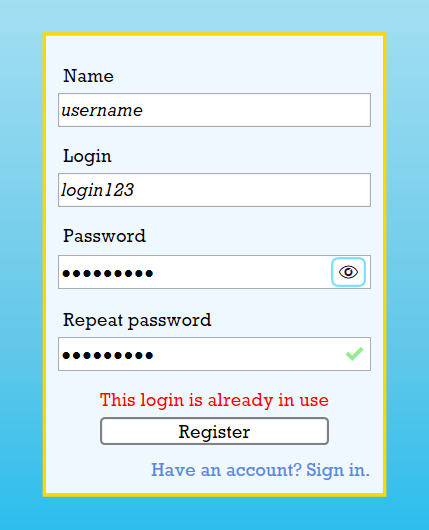


Рисунок 5.2 – Пользователь с таким логином уже существует

Предупреждение о неверно введенном пароле или логине продемонстрировано на рисунке 5.3.

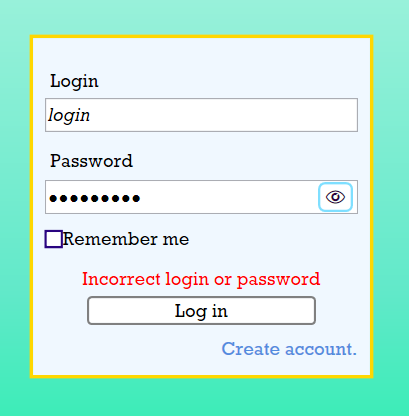


Рисунок 5.3 – Логин или пароль неверный

После авторизации пользователь попадает в меню, отсюда зайдя в раздел магазина, но здесь он сразу увидит сообщение о том, что активный персонаж не выбран. Результат действия приведен на рисунке 5.4.

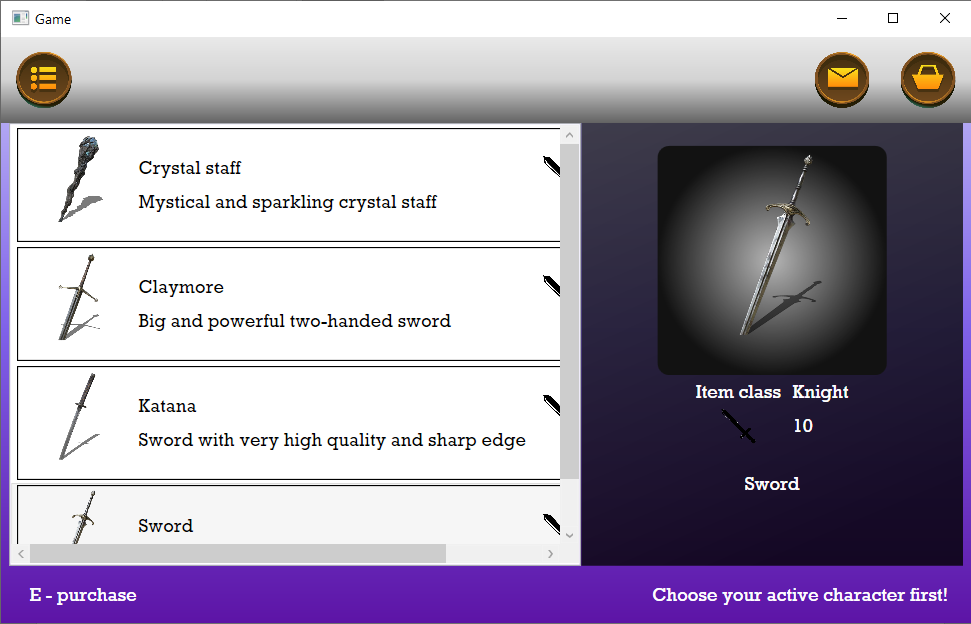


Рисунок 5.4 – Сообщение о том, что активный персонаж не выбран

Пользователь может попробовать приобрести предмет из списка, однако такая возможность заблокируется с соответствующим уведомлением. Результат метода проверки на рисунке 5.5.

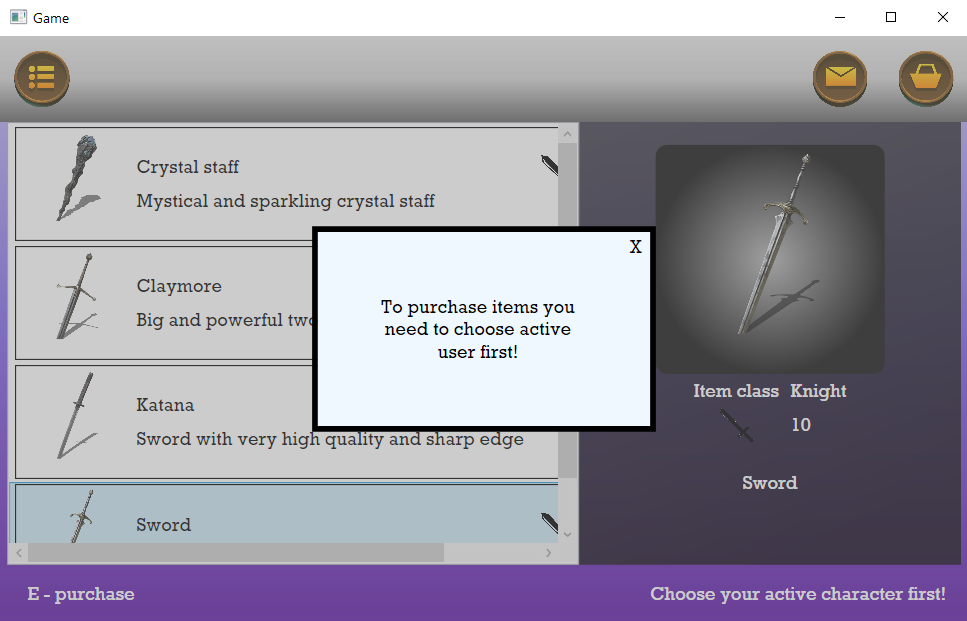


Рисунок 5.5 – Уведомление при попытке приобрести предмет без выбранного активного персонажа

В случае же если пользователь выбрал активного персонажа, но попытается ввести количество предметов большее, чем он сможет купить, потратив свою валюту, возможность совершить покупку заблокируется. Результат метода проверки на рисунке 5.6.

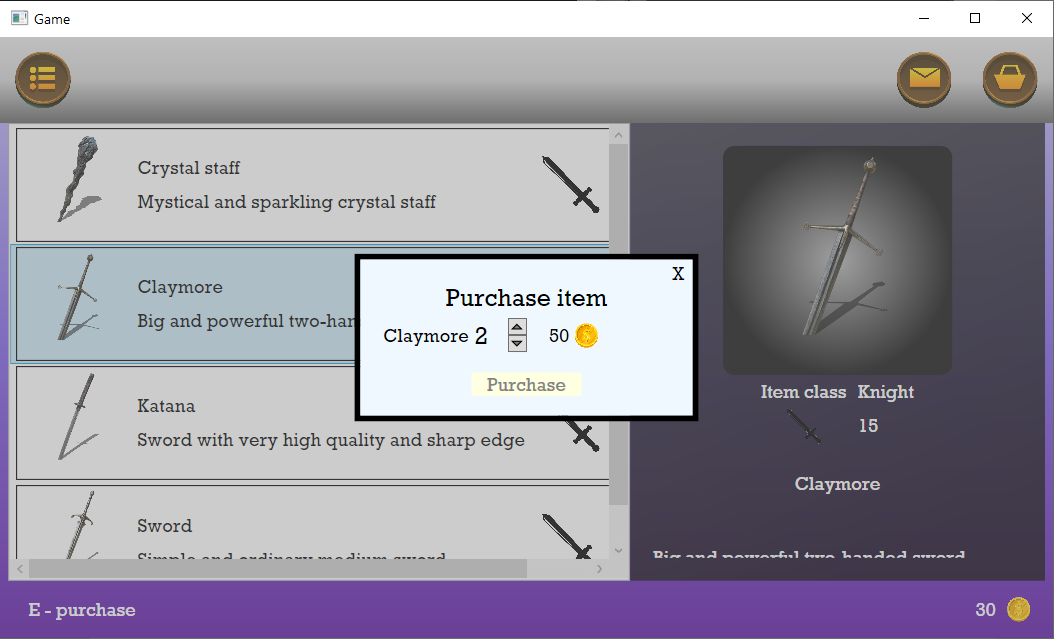


Рисунок 5.6 – Кнопка покупки большого количества предметов заблокирована

Подобное ограничение реализовано в листинге 5.2.

public ICommand ConfirmPurchaseCommand

{

get => confirmPurchaseCommand ?? (confirmPurchaseCommand = new CommandDelegate(ConfirmPurchase,

parameter => User.ActiveCharacter != null && SelectedItem != null

&& MainViewModel.User.ActiveCharacter.Gold >= SelectedItem.Price \* ItemQuantity));

Листинг 5.2 – команда, привязанная к кнопке покупки товара, с методом, ограничивающим доступ

В меню персонажа у пользователя появляется возможность экипировать на своего персонажа предмет из его инвентаря, однако это сделать будет невозможным, если у предмета и персонажа разные классы или если предмет имеет больший уровень чем персонаж. Данное ограничение представлено на рисунке 5.7.

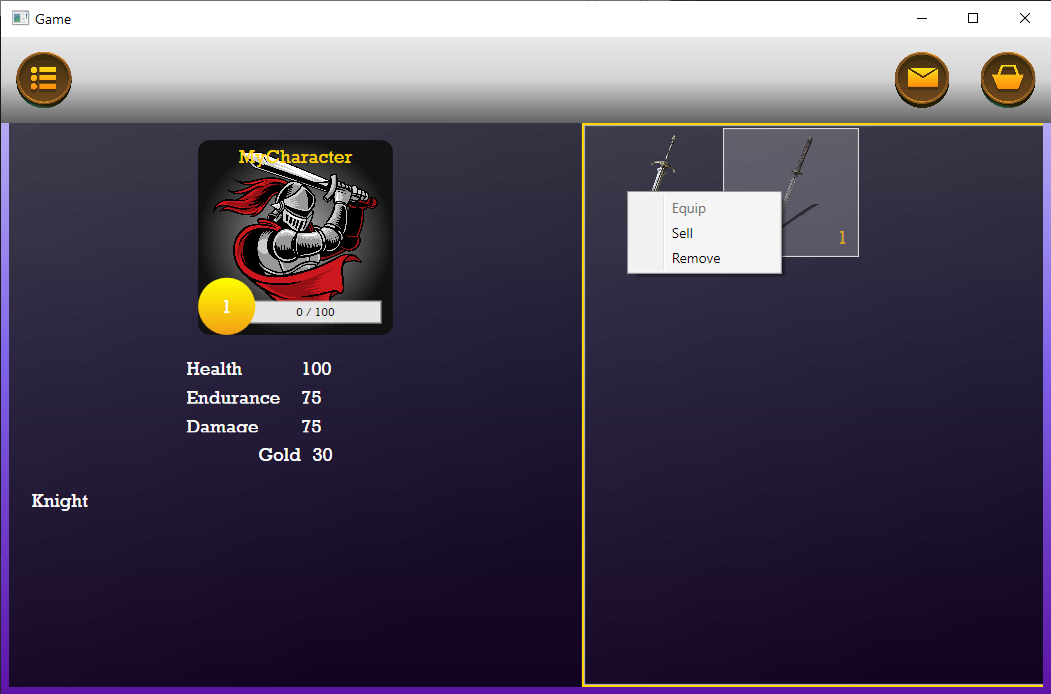


Рисунок 5.7 – Невозможно экипировать предмет большего уровня чем уровень персонажа

На листинге 5.3 представлен метод, накладывающий одно из ограничений.

public ICommand ToggleEquipItemCommand

{

get => toggleEquipItemCommand ?? (toggleEquipItemCommand = new CommandDelegate<ContainerItem>(ToggleEquipItem, containerItem =>

SelectedCharacter != null & containerItem != null &&

SelectedCharacter.Level >= containerItem.Item.Level && SelectedCharacter.Class == containerItem.Item.ItemClass));

}

Листинг 5.3 – команда, привязанная к кнопке экипировки предмета, с методом, ограничивающим доступ

На панели предложений обмена у пользователя есть возможность создать новое предложение обмена. Пользователь не сможет запросить больше предметов, чем есть в инвентаре у получателя и отправить больше предметов, чем есть в инвентаре его активного персонажа. Данное ограничение представлено на рисунке 5.8.

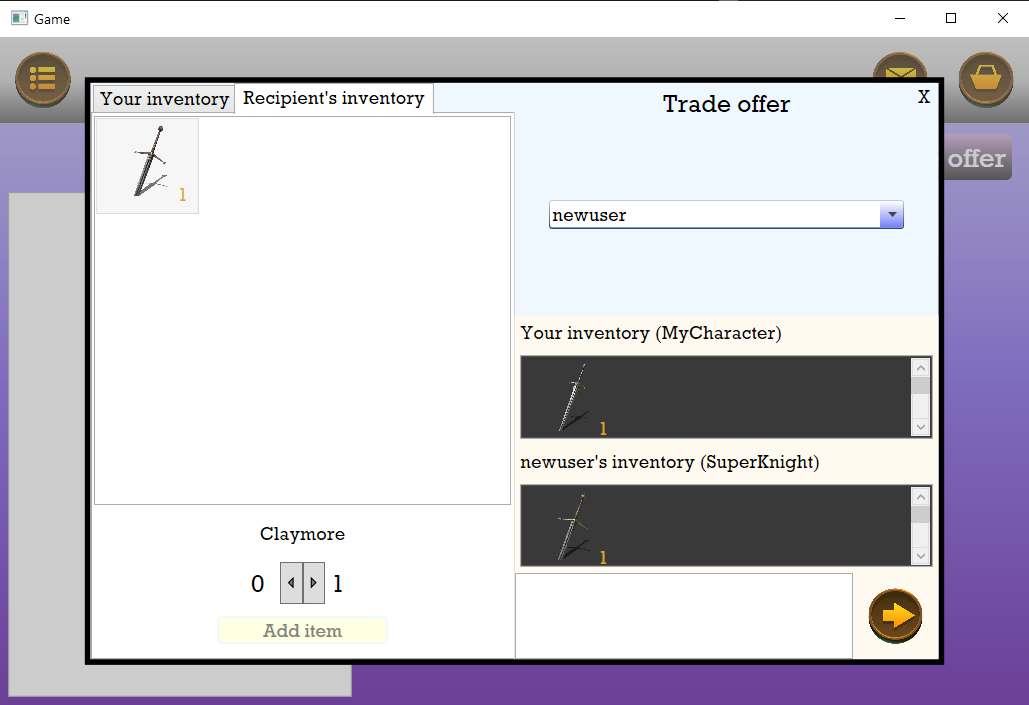
****

Рисунок 5.8 – Невозможно добавить в предложение обмена большее количество предметов, чем есть в инвентаре

Также предусмотрено валидационное сообщение, если предложение обмена содержит неверные или устаревшие данные об инвентаре персонажа получателя или отправителя, которое представлено на рисунке 5.9.

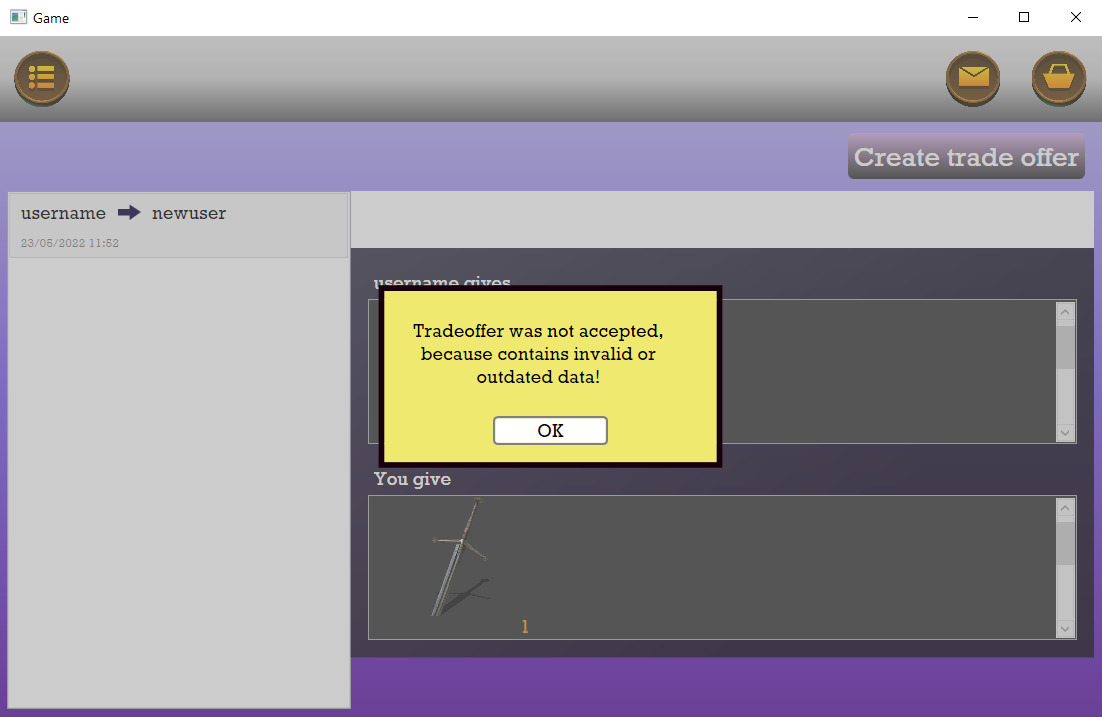


Рисунок 5.9 – Предложение обмена содержит неверные или устаревшие данные

# Методика использования программного средства

Окно входа содержит два поля для ввода логина и пароля. После ввода верных данных можно установить флажок на сохранения пользователя в свойствах приложения для дальнейшего автоматического входа. Далее необходимо нажать на кнопку вход. Окно входа в аккаунт представлено на рисунке 6.1.

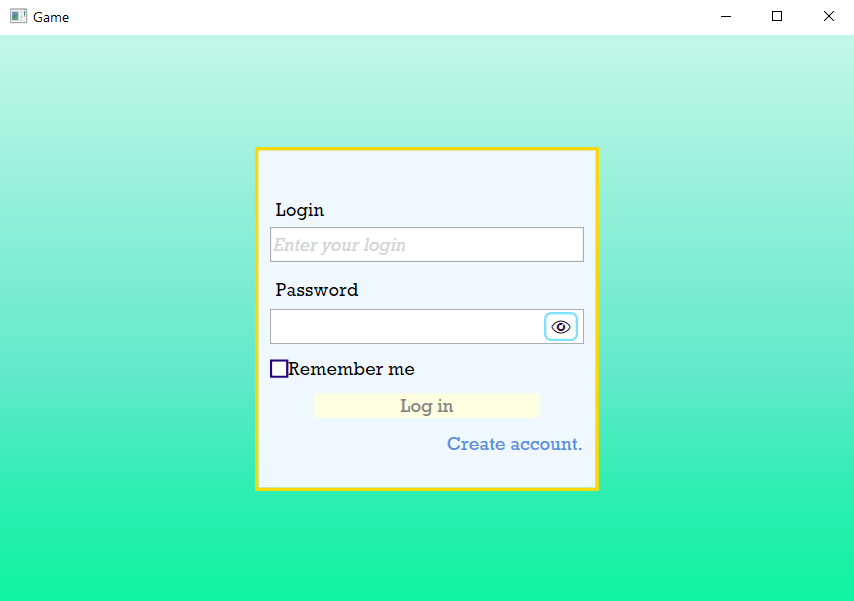


Рисунок 6.1 – Окно входа в аккаунт

Окно регистрации содержит поля для ввода имени, логина, пароля и повтора пароля. Введенная информация должна соответствовать требованиям и пароли должны совпадать. Окно регистрации представлено на рисунке 6.2.

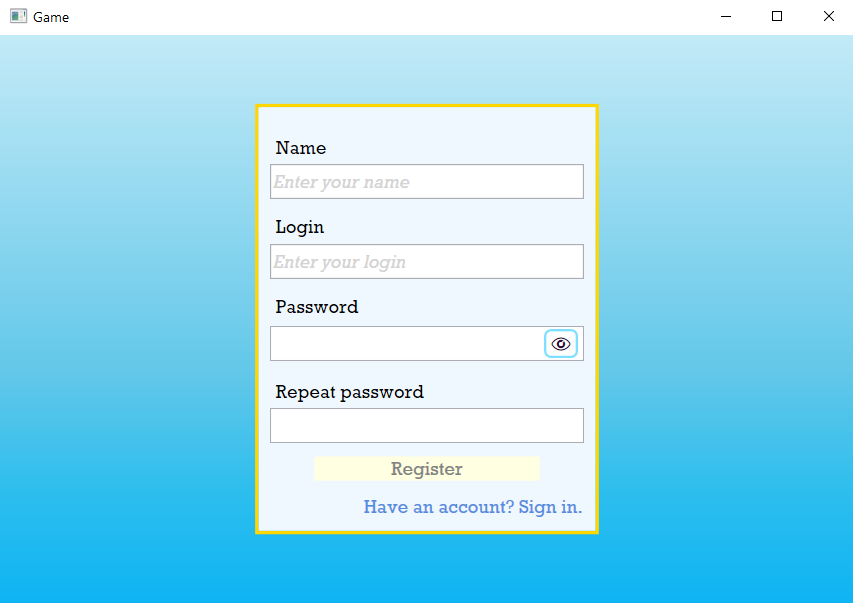


Рисунок 6.2 – Окно регистрации

Первая страница, которая открывается при входе в аккаунт, – это главное меню, откуда пользователь может попасть на остальные активные страницы. Данная страница представлена на рисунке 6.3.

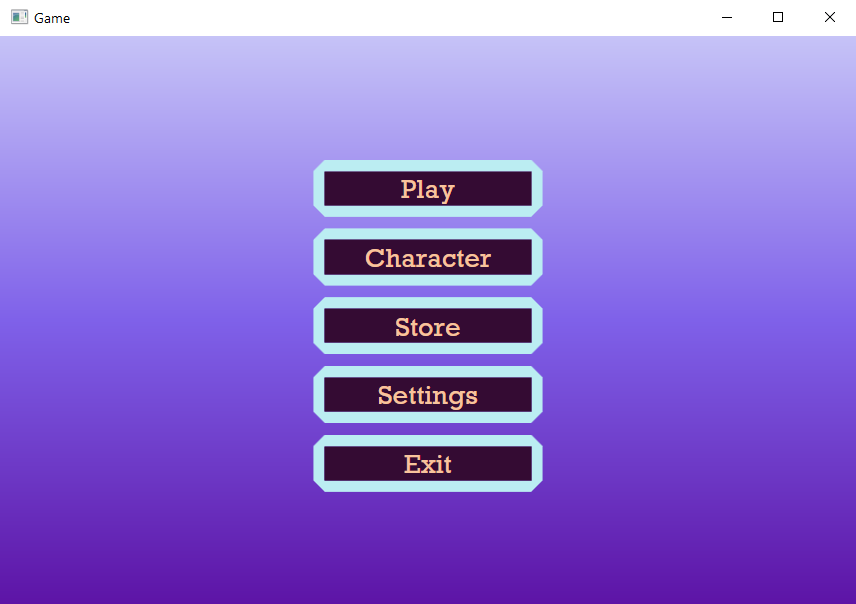


Рисунок 6.3 – Главное меню

На странице персонажа пользователь может создать нового персонажа, просмотреть имеющихся и выбрать активного. Также здесь пользователь может просмотреть инвентарь этого персонажа, информацию, характеристики и экипировать предметы из инвентаря. Страница персонажа представлена на рисунке 6.4.

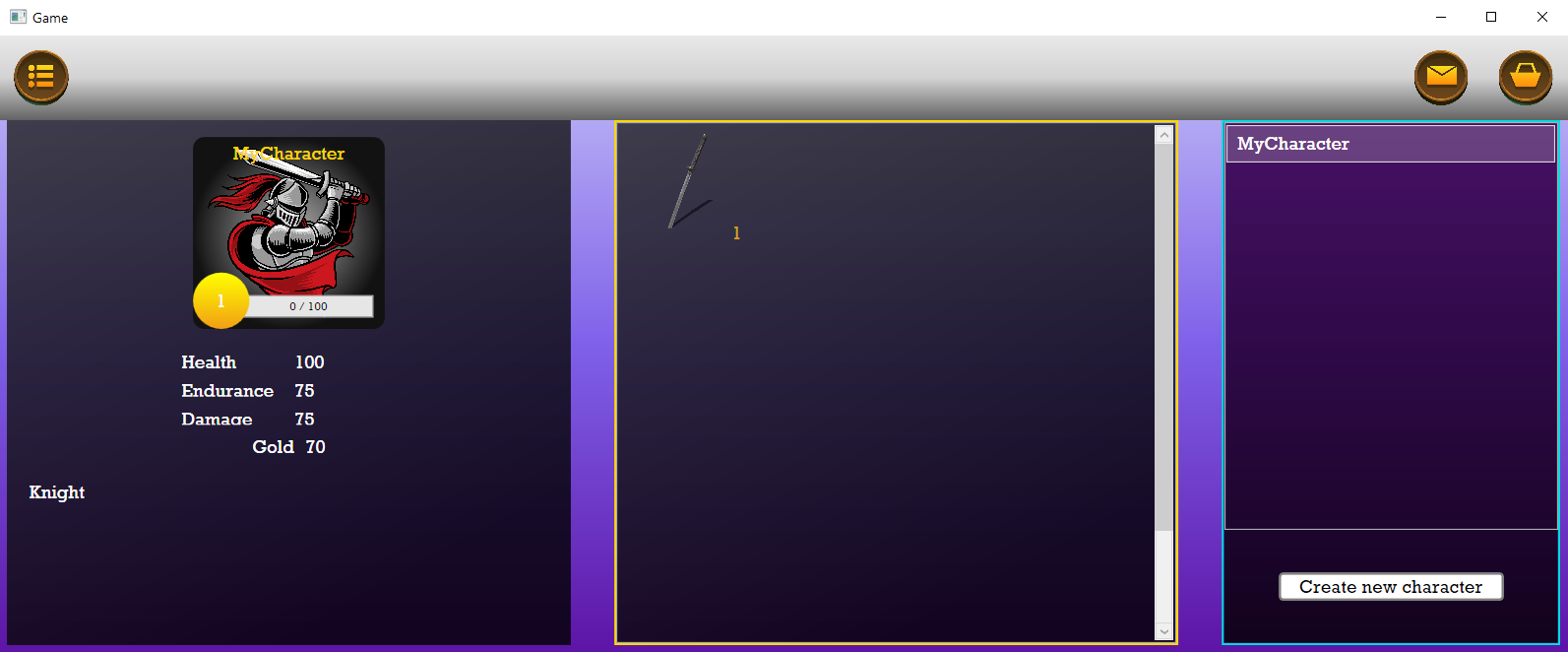


Рисунок 6.4 – Страница персонажа

При нажатии на кнопку «Создать персонажа» появляется модальное окно, позволяющее пользователю выбрать класс нового персонажа и его имя, отслеживая базовые характеристики создаваемого персонажа. На протяжении всей работы с данным окном содержимое основного окна блокируется. Данное окно представлено на рисунке 6.5.

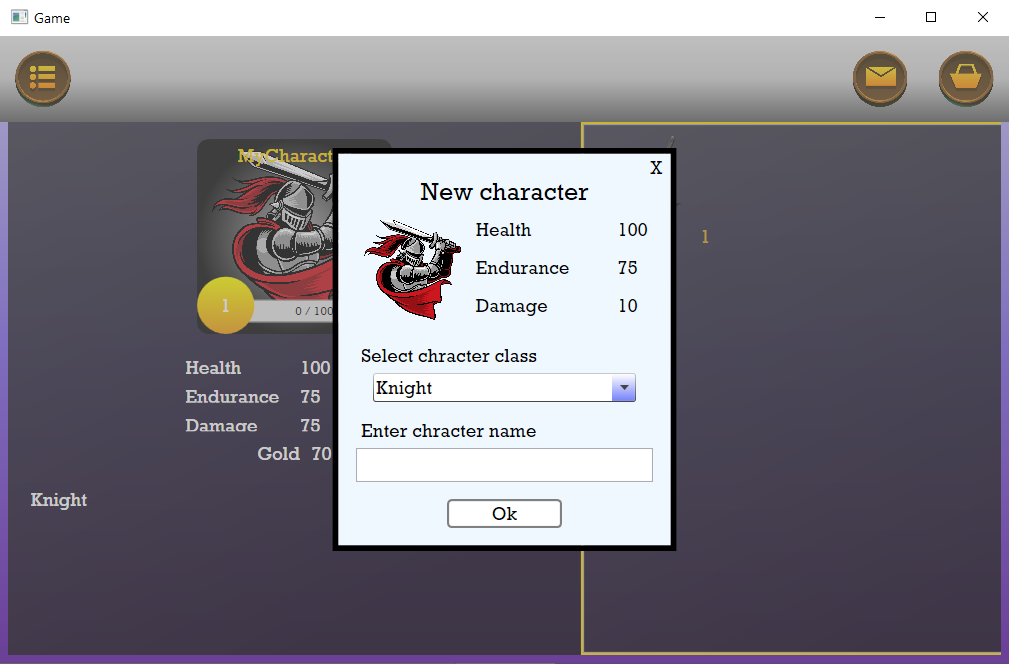


Рисунок 6.5 – Окно создания персонажа

Страница магазина позволяет просмотреть весь список доступных предметов, изучить информацию о выбранном предмете и приобретать пользователю предметы, используя активного персонажа. Данная страница представлена на рисунке 6.6.

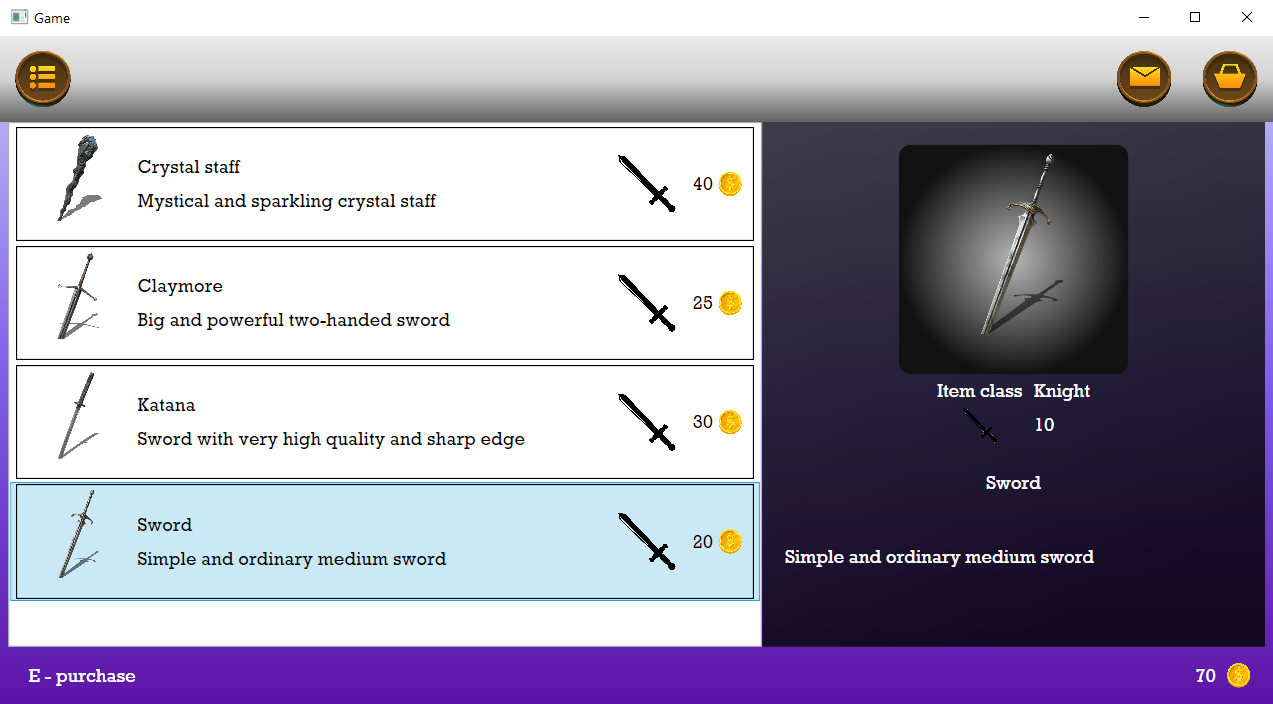
****

Рисунок 6.6 – Страница магазина

При нажатии на клавишу «E» и выбранном предмете появляется модальное окно, позволяющее пользователю выбрать необходимое количество предметов и подтвердить покупку. На протяжении всей работы с данным окном содержимое основного окна блокируется. Данное окно представлено на рисунке 6.7.

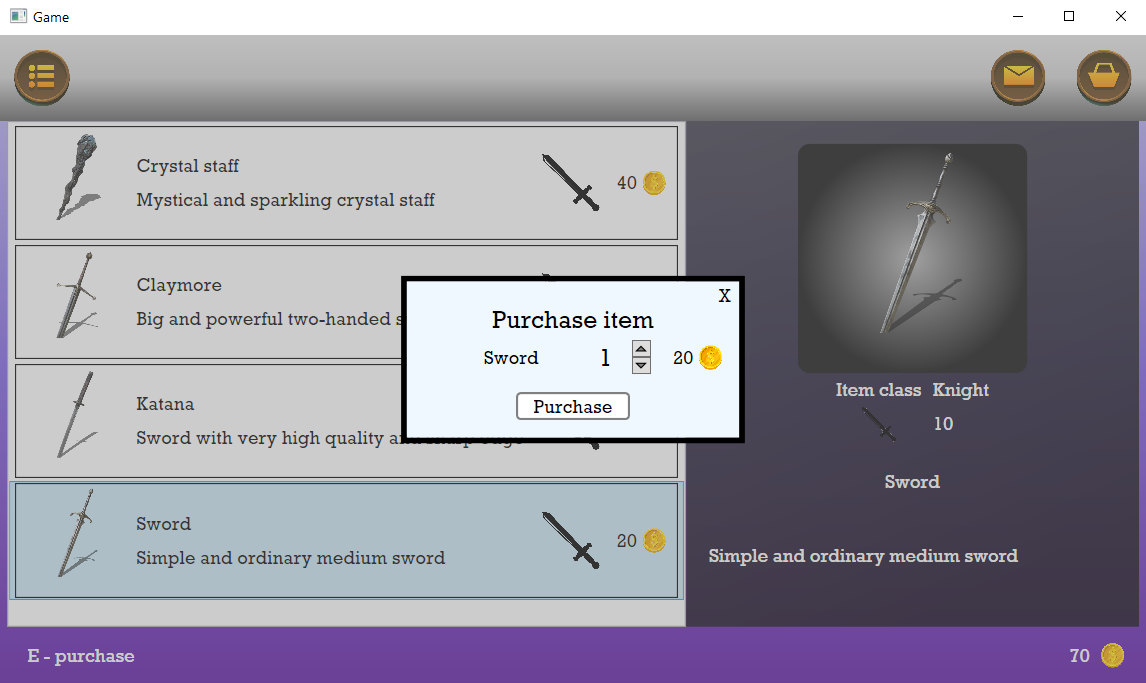


Рисунок 6.7 – Окно покупки предметов

На панели, которая располагается вверху всех основных страниц, можно перейти на страницу менеджера сообщений, которая содержит список всех пользователей, отсортированный по последнему отправленному этим пользователем сообщению, и панель, на которой отобразится сформированный чат, при выборе пользователя из списка. Внизу этой панели пользователь сможет написать новое. Страница сообщений представлена на рисунке 6.8.



Рисунок 6.8 – Страница сообщений

Также с помощью верхней панели, можно перейти на страницу предложений обмен, которая содержит список всех предложений обмена, которые адресованы или отправлены текущим пользователем. Список будет отсортирован по дате отправки. Также при выборе предложения обмена можно просмотреть информацию о нем и активные предметы. Если предложение обмена получил текущий пользователь и оно активно, то он сможет его принять, получив указанные предметы, но и отдав свои. В углу страницы располагается кнопка создания предложения обмена. Страница предложений обмена представлена на рисунке 6.9.

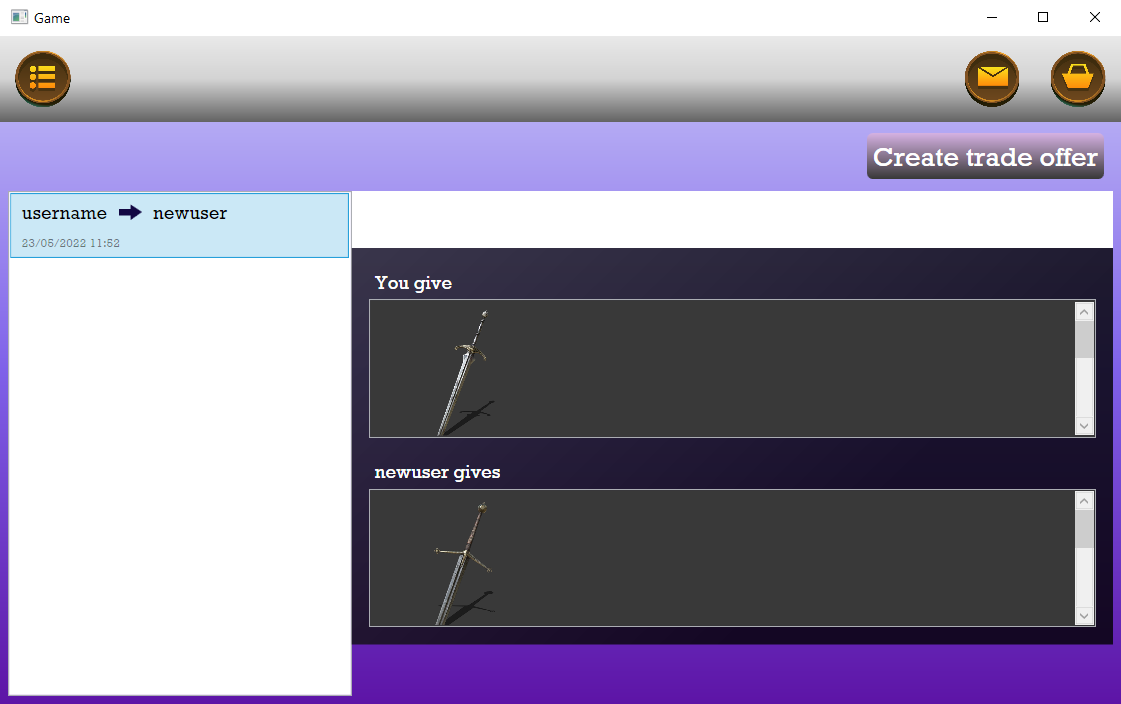
****

Рисунок 6.9 – Страница предложений обмена

При нажатии на кнопку «Создать предложение обмена» появляется модальное окно, позволяющее пользователю выбрать получателя, выбрать предметы, которые он хочет получить из его инвентаря и предметы, которые он готов отдать из своего инвентаря, возможно написать комментарий и отправить предложение обмена. Данное окно представлено на рисунке 6.10.

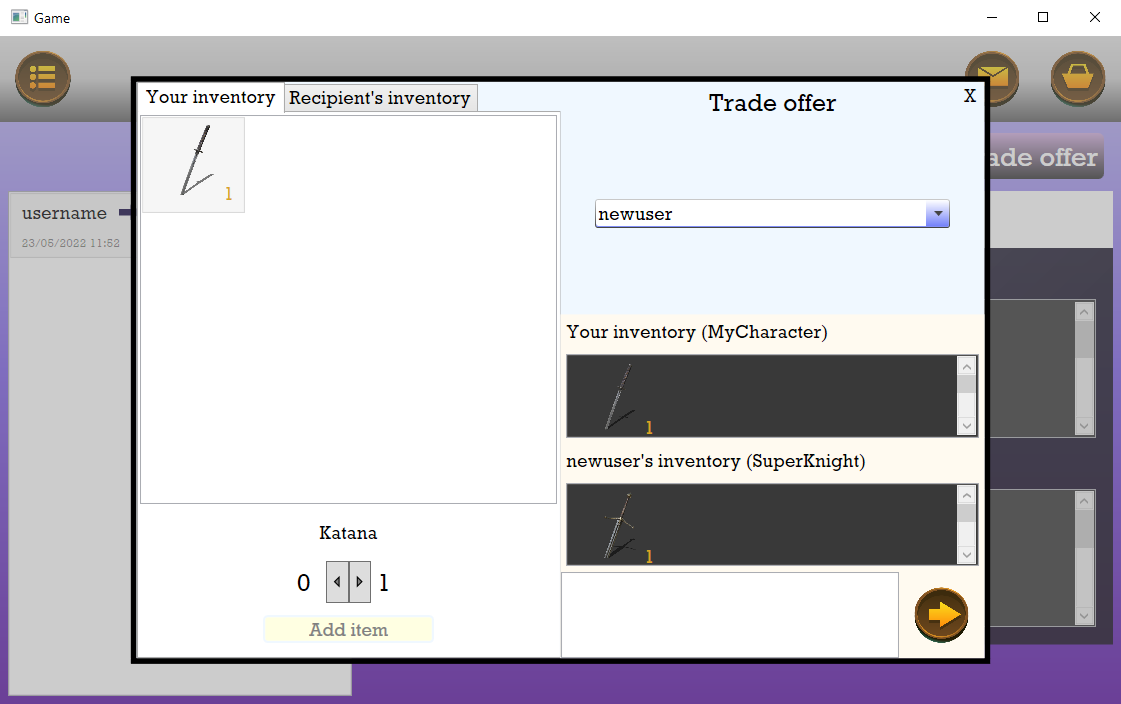


Рисунок 6.10 – Окно создания предложения обмена

Страница настроек предоставляет пользователю интерфейс, с помощью которого он может сменить язык локализации приложения, изменить режим отображения окна, выбрать, необходим ли ему автоматический вход в приложение и сохранить эти настройки. Также здесь возможен выход из аккаунта с последующим перенаправлением на страницу входа. Данная страница представлена на рисунке 6.11.

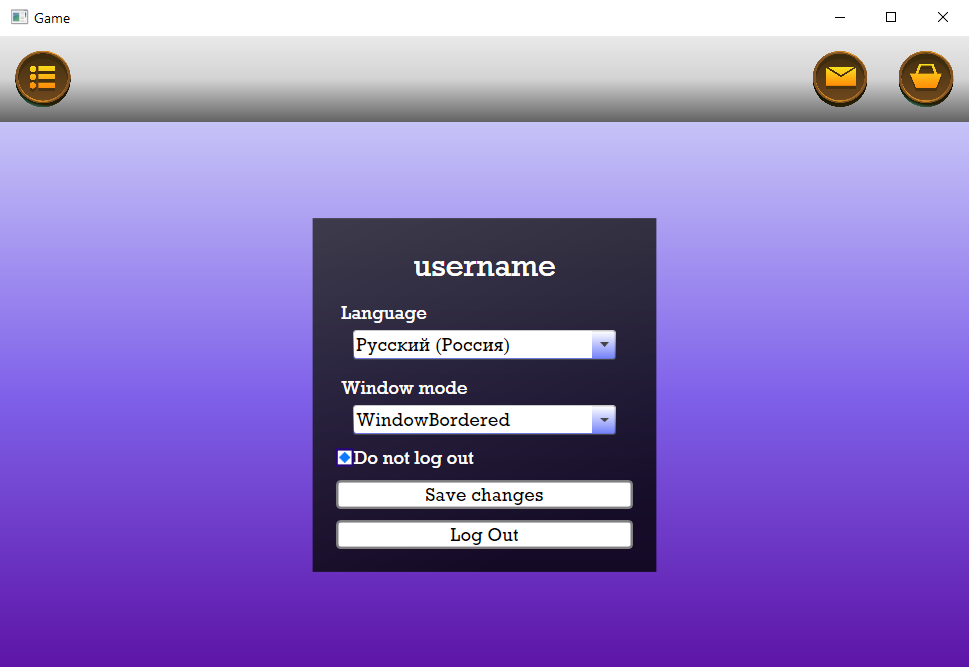


Рисунок 6.11 – Страница настроек

При входе в приложение под аккаунтом администратора присутствует возможность просмотра списка всех пользователей, с возможностью изменения роли или статуса выбранного пользователя. Также на дополнительной панели можно просмотреть информацию об отправленных и принятых сообщениях, предложениях обмена, а также о персонажах этого пользователя. Страница администрирования представлена на рисунке 6.12.

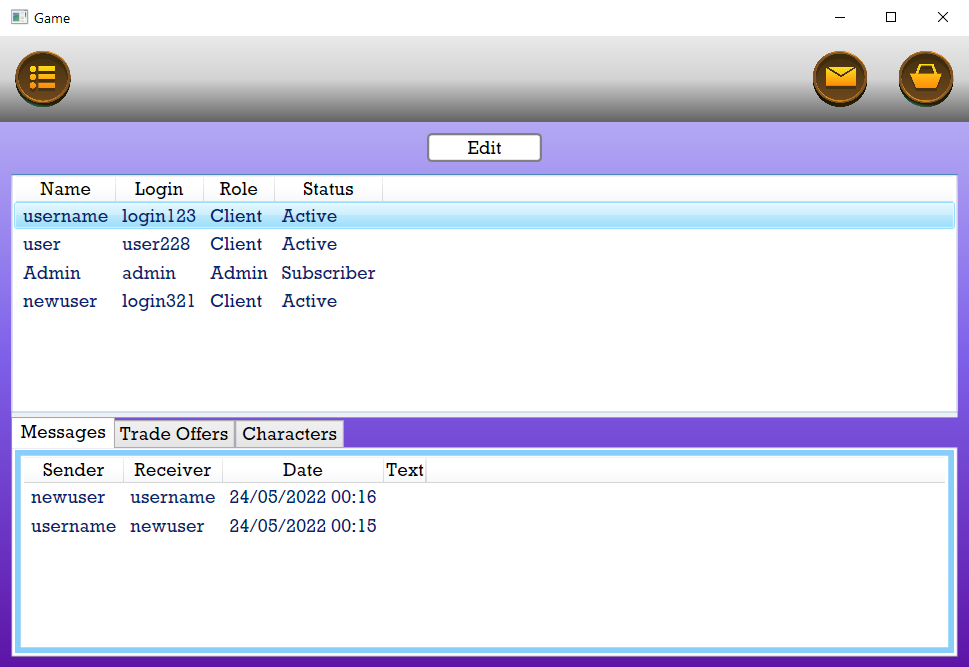
****

Рисунок 6.12 – Страница администрирования приложения

# Заключение

Основной задачей данного проекта было усвоение основных технологий разработки десктопных приложений. Разработанное программное средство послужило практической реализацией знаний, полученных в результате изучения теоретических знаний работы с базами данных, технологиями WPF и Entity Framework Core.

В процессе решения поставленной задачи была достигнута поставленная цель по созданию приложения «Компьютерная ролевая игра». Данное приложение может служить не только примером практического применения полученных знаний, но также представляет из себя готовый продукт, который может иметь практическую пользу при использовании его по назначению.

В программном средстве были реализованы следующие функции:

* регистрация и авторизация;
* создание персонажей пользователя;
* развитие персонажа и накопление игровой валюты;
* приобретение игровых предметов;
* отправка и чтение сообщений пользователей;
* создание и отправка предложений обмена предметов пользователей;
* подтверждение и получение предметов предложений обмена других пользователей;
* изменение внешнего вида приложения и настройка приложения под предпочтения пользователя;
* администрирование приложения.

Разработанная программа имеет завершенный вид, работает верно и готова к использованию.

# СПИСОК ЛИТЕРАТУРЫ

[1] Миха Мартин, Роберт Мартин «Agile Principles, Patterns, and Practices in C# » – 2006 г.

[2] Статья “WPF – система авторизации и регистрации” [Электронный ресурс]. – Электронные данные. – https://you-hands.ru/2018/08/31/wpf-sistema-avtorizacii-i-registracii/

[3] Статья “WPF” [Электронный ресурс]. – Электронные данные. – https://metanit.com/sharp/wpf/1.php

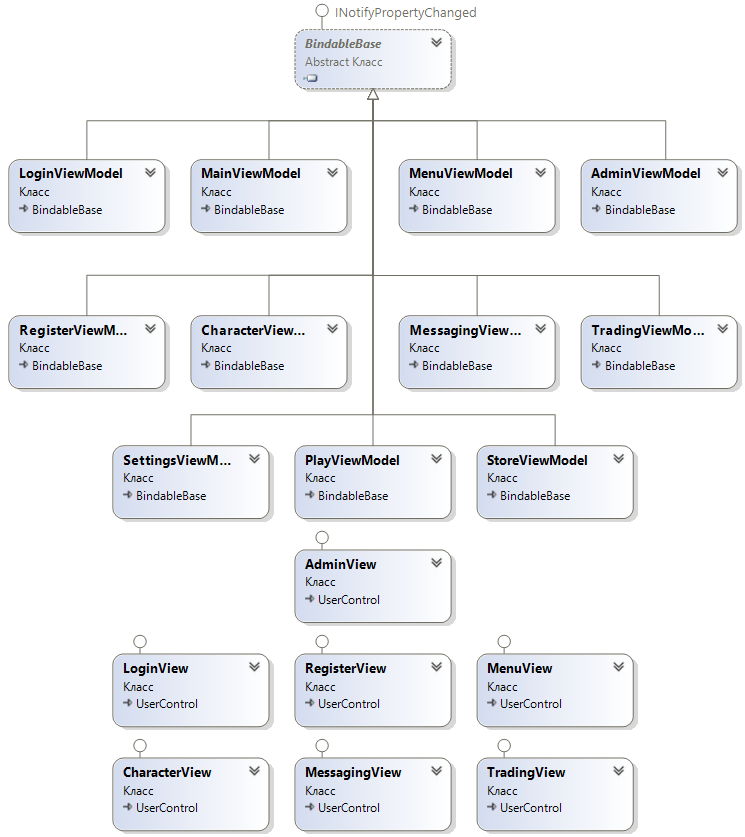
[4] Статья “Generating model from an existing database” [Электронный ресурс]. –Электронные данные. – <https://www.learnentityframeworkcore.com/walkthroughs/existing-database>

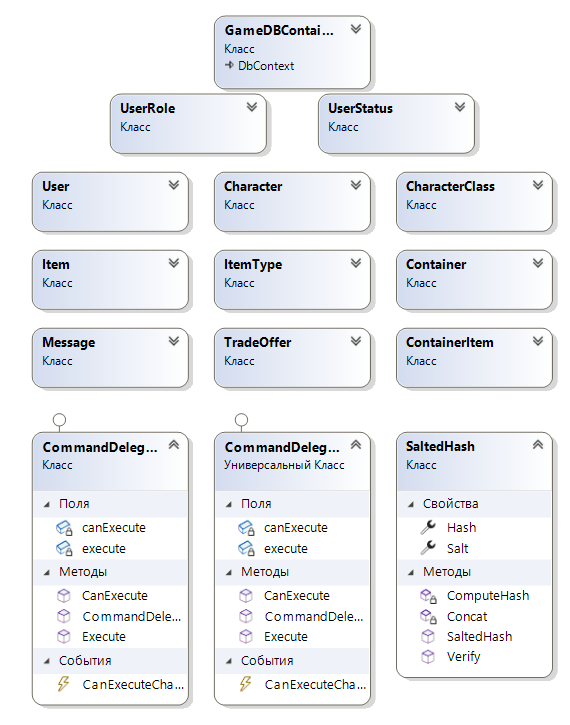
[5] Статья “Паттерн MVVM” [Электронный ресурс]. – Электронные данные. – https://metanit.com/sharp/wpf/22.1.php

# Приложения

## Приложение А

**Диаграмма классов**

****

****

## Приложение Б

**Скрипт для создания таблиц базы данных приложения**

CREATE DATABASE [GameDB];

USE [GameDB];

CREATE TABLE [Users] (

[Id] uniqueidentifier NOT NULL,

[Name] nvarchar(100) NOT NULL,

[Login] nvarchar(100) NOT NULL,

[PswHash] varbinary(max) NOT NULL,

[PswSalt] varbinary(max) NOT NULL,

[Role\_Id] int NOT NULL,

[Status\_Id] int NOT NULL,

[ActiveCharacter\_Id] uniqueidentifier,

CONSTRAINT [PK\_Users] PRIMARY KEY CLUSTERED ([Id] ASC)

);

CREATE TABLE [UserRoles] (

[Id] int NOT NULL,

[Name] nvarchar(50) NOT NULL,

CONSTRAINT [PK\_UserRoles] PRIMARY KEY CLUSTERED ([Id] ASC)

);

CREATE TABLE [UserStatuses] (

[Id] int NOT NULL,

[Name] nvarchar(50) NOT NULL,

CONSTRAINT [PK\_UserStatuses] PRIMARY KEY CLUSTERED ([Id] ASC)

);

CREATE TABLE [Characters] (

[Id] uniqueidentifier NOT NULL,

[Name] nvarchar(100) NOT NULL,

[Level] int NOT NULL,

[Experience] int NOT NULL,

[Gold] int NOT NULL,

[Health] int NOT NULL,

[Endurance] int NOT NULL,

[Damage] int NOT NULL,

[Class\_Id] int NOT NULL,

[User\_Id] uniqueidentifier NOT NULL,

[Inventory\_Id] uniqueidentifier NOT NULL,

[Weapon\_Id] uniqueidentifier,

[Util\_Id] uniqueidentifier,

[Head\_Id] uniqueidentifier,

[Armor\_Id] uniqueidentifier,

CONSTRAINT [PK\_Characters] PRIMARY KEY CLUSTERED ([Id] ASC)

);

CREATE TABLE [CharacterClasses] (

[Id] int NOT NULL,

[Name] nvarchar(50) NOT NULL,

[Description] nvarchar(1000) NOT NULL,

[BaseHealth] int NOT NULL,

[BaseEndurance] int NOT NULL,

[BaseDamage] int NOT NULL,

[Image] nvarchar(max) NOT NULL,

CONSTRAINT [PK\_CharacterClasses] PRIMARY KEY CLUSTERED ([Id] ASC)

);

CREATE TABLE [Items] (

[Id] uniqueidentifier NOT NULL,

[Name] nvarchar(100) NOT NULL,

[Description] nvarchar(1000) NOT NULL,

[Level] int NOT NULL,

[Price] int NOT NULL,

[Stat] int NOT NULL,

[Image] nvarchar(max) NOT NULL,

[ItemType\_Id] int NOT NULL,

[ItemClass\_Id] int NOT NULL,

CONSTRAINT [PK\_Items] PRIMARY KEY CLUSTERED ([Id] ASC)

);

CREATE TABLE [ItemTypes] (

[Id] int NOT NULL,

[Name] nvarchar(50) NOT NULL,

[Image] nvarchar(max) NOT NULL,

CONSTRAINT [PK\_ItemTypes] PRIMARY KEY CLUSTERED ([Id] ASC)

);

CREATE TABLE [Containers] (

[Id] uniqueidentifier NOT NULL,

CONSTRAINT [PK\_Containers] PRIMARY KEY CLUSTERED ([Id] ASC)

);

CREATE TABLE [ContainerItems] (

[Id] uniqueidentifier NOT NULL,

[Quantity] int NOT NULL,

[Item\_Id] uniqueidentifier NOT NULL,

[Container\_Id] uniqueidentifier NOT NULL,

CONSTRAINT [PK\_InventoryItems] PRIMARY KEY CLUSTERED ([Id] ASC)

);

CREATE TABLE [Messages] (

[Id] uniqueidentifier NOT NULL,

[Text] nvarchar(1000) NOT NULL,

[IsNew] bit NOT NULL,

[Date] datetime NOT NULL,

[Sender\_Id] uniqueidentifier NOT NULL,

[Receiver\_Id] uniqueidentifier NOT NULL,

CONSTRAINT [PK\_Messages] PRIMARY KEY CLUSTERED ([Id] ASC)

);

CREATE TABLE [TradeOffers] (

[Id] uniqueidentifier NOT NULL,

[Comment] nvarchar(max) NOT NULL,

[IsActive] bit NOT NULL,

[Date] datetime NOT NULL,

[Sender\_Id] uniqueidentifier NOT NULL,

[Receiver\_Id] uniqueidentifier NOT NULL,

[SenderContainer\_Id] uniqueidentifier NOT NULL,

[ReceiverContainer\_Id] uniqueidentifier NOT NULL,

CONSTRAINT [PK\_TradeOffers] PRIMARY KEY CLUSTERED ([Id] ASC)

);

ALTER TABLE [Users]

ADD CONSTRAINT [FK\_UserUserRole]

FOREIGN KEY ([Role\_Id])

REFERENCES [UserRoles]

([Id])

ON DELETE NO ACTION ON UPDATE NO ACTION;

ALTER TABLE [Users]

ADD CONSTRAINT [FK\_UserUserStatus]

FOREIGN KEY ([Status\_Id])

REFERENCES [UserStatuses]

([Id])

ON DELETE NO ACTION ON UPDATE NO ACTION;

ALTER TABLE [Users]

ADD CONSTRAINT [FK\_UserActiveCharacter]

FOREIGN KEY ([ActiveCharacter\_Id])

REFERENCES [Characters]

([Id])

ON DELETE NO ACTION ON UPDATE NO ACTION;

ALTER TABLE [Characters]

ADD CONSTRAINT [FK\_CharacterCharacterClass]

FOREIGN KEY ([Class\_Id])

REFERENCES [CharacterClasses]

([Id])

ON DELETE NO ACTION ON UPDATE NO ACTION;

ALTER TABLE [Characters]

ADD CONSTRAINT [FK\_UserCharacters]

FOREIGN KEY ([User\_Id])

REFERENCES [Users]

([Id])

ON DELETE NO ACTION ON UPDATE NO ACTION;

ALTER TABLE [Characters]

ADD CONSTRAINT [FK\_CharacterInventory]

FOREIGN KEY ([Inventory\_Id])

REFERENCES [Containers]

([Id])

ON DELETE NO ACTION ON UPDATE NO ACTION;

ALTER TABLE [Characters]

ADD CONSTRAINT [FK\_CharacterWeapon]

FOREIGN KEY ([Weapon\_Id])

REFERENCES [ContainerItems]

([Id])

ON DELETE NO ACTION ON UPDATE NO ACTION;

ALTER TABLE [Characters]

ADD CONSTRAINT [FK\_CharacterUtil]

FOREIGN KEY ([Util\_Id])

REFERENCES [ContainerItems]

([Id])

ON DELETE NO ACTION ON UPDATE NO ACTION;

ALTER TABLE [Characters]

ADD CONSTRAINT [FK\_CharacterHead]

FOREIGN KEY ([Head\_Id])

REFERENCES [ContainerItems]

([Id])

ON DELETE NO ACTION ON UPDATE NO ACTION;

ALTER TABLE [Characters]

ADD CONSTRAINT [FK\_CharacterArmor]

FOREIGN KEY ([Armor\_Id])

REFERENCES [ContainerItems]

([Id])

ON DELETE NO ACTION ON UPDATE NO ACTION;

ALTER TABLE [Items]

ADD CONSTRAINT [FK\_ItemItemType]

FOREIGN KEY ([ItemType\_Id])

REFERENCES [ItemTypes]

([Id])

ON DELETE NO ACTION ON UPDATE NO ACTION;

ALTER TABLE [Items]

ADD CONSTRAINT [FK\_ItemItemClass]

FOREIGN KEY ([ItemClass\_Id])

REFERENCES [CharacterClasses]

([Id])

ON DELETE NO ACTION ON UPDATE NO ACTION;

ALTER TABLE [ContainerItems]

ADD CONSTRAINT [FK\_ContainerItemItem]

FOREIGN KEY ([Item\_Id])

REFERENCES [Items]

([Id])

ON DELETE NO ACTION ON UPDATE NO ACTION;

ALTER TABLE [ContainerItems]

ADD CONSTRAINT [FK\_ContainerItemContainer]

FOREIGN KEY ([Container\_Id])

REFERENCES [Containers]

([Id])

ON DELETE NO ACTION ON UPDATE NO ACTION;

ALTER TABLE [Messages]

ADD CONSTRAINT [FK\_MessageSender]

FOREIGN KEY ([Sender\_Id])

REFERENCES [Users]

([Id])

ON DELETE NO ACTION ON UPDATE NO ACTION;

ALTER TABLE [Messages]

ADD CONSTRAINT [FK\_MessageReceiver]

FOREIGN KEY ([Receiver\_Id])

REFERENCES [Users]

([Id])

ON DELETE NO ACTION ON UPDATE NO ACTION;

ALTER TABLE [TradeOffers]

ADD CONSTRAINT [FK\_TradeOfferSender]

FOREIGN KEY ([Sender\_Id])

REFERENCES [Users]

([Id])

ON DELETE NO ACTION ON UPDATE NO ACTION;

ALTER TABLE [TradeOffers]

ADD CONSTRAINT [FK\_TradeOfferReceiver]

FOREIGN KEY ([Receiver\_Id])

REFERENCES [Users]

([Id])

ON DELETE NO ACTION ON UPDATE NO ACTION;

ALTER TABLE [TradeOffers]

ADD CONSTRAINT [FK\_TradeOfferSenderContainer]

FOREIGN KEY ([SenderContainer\_Id])

REFERENCES [Containers]

([Id])

ON DELETE NO ACTION ON UPDATE NO ACTION;

ALTER TABLE [TradeOffers]

ADD CONSTRAINT [FK\_TradeOfferReceiverContainer]

FOREIGN KEY ([ReceiverContainer\_Id])

REFERENCES [Containers]

([Id])

ON DELETE NO ACTION ON UPDATE NO ACTION;

## Приложение В

**Класс MainViewModel**

using System;

using System.Linq;

using System.Windows;

using System.Windows.Input;

using Prism.Mvvm;

using GameWPF.Model;

using GameWPF.Model.Enum;

using GameWPF.View;

namespace GameWPF.ViewModel

{

public class MainViewModel : BindableBase

{

private GameDBContainer dbContext;

public GameDBContainer DbContext

{

get => dbContext ?? (dbContext = new GameDBContainer());

}

public User User { get; set; }

public bool IsUserAuthorizing { get; set; }

private WindowState windowState;

public WindowState WindowState

{

get => windowState;

set => SetProperty(ref windowState, value);

}

private WindowStyle windowStyle;

public WindowStyle WindowStyle

{

get => windowStyle;

set => SetProperty(ref windowStyle, value);

}

private BindableBase currentViewModel;

public BindableBase CurrentViewModel

{

get => currentViewModel;

set => SetProperty(ref currentViewModel, value);

}

private RegisterViewModel RegisterVM { get; set; }

private LoginViewModel LoginVM { get; set; }

private MenuViewModel MenuVM { get; set; }

private SettingsViewModel SettingsVM { get; set; }

private AdminViewModel AdminVM { get; set; }

private MessagingViewModel MessagingVM { get; set; }

private TradingViewModel TradingVM { get; set; }

private StoreViewModel StoreVM { get; set; }

private CharacterViewModel CharacterVM { get; set; }

private PlayViewModel PlayVM { get; set; }

private CommandDelegate<ViewModelEnum> setVMCommand;

public ICommand SetVMCommand

{

get => setVMCommand ?? (setVMCommand = new CommandDelegate<ViewModelEnum>(SetViewModel));

}

public MainViewModel()

{

InitializeDatabase();

SetWindowMode(WindowMode.GetWindowMode(Properties.Settings.Default.DefaultWindowMode));

User = DbContext.Users.Find(Properties.Settings.Default.UserId);

IsUserAuthorizing = true;

SetViewModel(User != null ? ViewModelEnum.Menu : ViewModelEnum.Login);

}

public void SetWindowMode(WindowMode windowMode)

{

WindowState = windowMode.WindowState;

WindowStyle = windowMode.WindowStyle;

}

public void SetViewModel(ViewModelEnum vm)

{

if (User != null && IsUserAuthorizing)

{

IsUserAuthorizing = false;

MenuVM = new MenuViewModel(this);

SettingsVM = new SettingsViewModel(this);

AdminVM = new AdminViewModel(this);

MessagingVM = new MessagingViewModel(this);

TradingVM = new TradingViewModel(this);

StoreVM = new StoreViewModel(this);

CharacterVM = new CharacterViewModel(this);

PlayVM = new PlayViewModel(this);

}

switch (vm)

{

case ViewModelEnum.Register:

{

RegisterVM = new RegisterViewModel(this);

CurrentViewModel = RegisterVM;

}

break;

case ViewModelEnum.Login:

{

LoginVM = new LoginViewModel(this);

CurrentViewModel = LoginVM;

}

break;

case ViewModelEnum.Menu: CurrentViewModel = MenuVM; break;

case ViewModelEnum.Settings: CurrentViewModel = SettingsVM; break;

case ViewModelEnum.Admin: CurrentViewModel = AdminVM; break;

case ViewModelEnum.Messaging: CurrentViewModel = MessagingVM; break;

case ViewModelEnum.Trading: CurrentViewModel = TradingVM; break;

case ViewModelEnum.Store: CurrentViewModel = StoreVM; break;

case ViewModelEnum.Character: CurrentViewModel = CharacterVM; break;

case ViewModelEnum.Play: CurrentViewModel = PlayVM; break;

case ViewModelEnum.Exit: Application.Current.Shutdown(); break;

}

}

private void InitializeDatabase()

{

var transaction = DbContext.Database.BeginTransaction();

try

{

if (DbContext.UserRoles.Count() == 0) DbContext.UserRoles.AddRange(Enum.GetValues(typeof(UserRoleEnum)).Cast<UserRoleEnum>().Select(@enum => (UserRole)@enum));

if (DbContext.UserStatuses.Count() == 0)

DbContext.UserStatuses.AddRange(Enum.GetValues(typeof(UserStatusEnum)).Cast<UserStatusEnum>().Select(@enum => (UserStatus)@enum));

if (DbContext.CharacterClasses.Count() == 0)

DbContext.CharacterClasses.AddRange(Enum.GetValues(typeof(CharacterClassEnum)).Cast<CharacterClassEnum>().Select(@enum => (CharacterClass)@enum));

if (DbContext.ItemTypes.Count() == 0)

DbContext.ItemTypes.AddRange(Enum.GetValues(typeof(ItemTypeEnum)).Cast<ItemTypeEnum>().Select(@enum => (ItemType)@enum));

DbContext.SaveChanges();

transaction.Commit();

}

catch (Exception exception)

{

transaction.Rollback();

throw new Exception("Primary database tables were not initialized", exception);

}

transaction = DbContext.Database.BeginTransaction();

try

{

if (DbContext.Users.Count() == 0)

{

User admin = new User();

admin.Id = Guid.NewGuid();

admin.Name = "Admin";

admin.Login = "admin";

Util.Hash.SaltedHash psw = new Util.Hash.SaltedHash("admin123");

admin.PswHash = psw.Hash;

admin.PswSalt = psw.Salt;

admin.Role = UserRoleEnum.Admin.GetModel(DbContext.UserRoles);

admin.Status = UserStatusEnum.Subscriber.GetModel(DbContext.UserStatuses);

DbContext.Users.Add(admin);

}

DbContext.SaveChanges();

transaction.Commit();

}

catch (Exception exception)

{

transaction.Rollback();

throw new Exception("Secondary database tables were not initialized", exception);

}}}}

## Приложение Г

**Класс CommandDelegate**

using System;

using System.Windows.Input;

namespace GameWPF.ViewModel

{

public class CommandDelegate : ICommand

{

private Action<object> execute;

private Func<object, bool> canExecute;

public event EventHandler CanExecuteChanged

{

add { CommandManager.RequerySuggested += value; }

remove { CommandManager.RequerySuggested += value; }

}

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

{

this.execute = execute;

this.canExecute = canExecute;

}

public void Execute(object parameter)

{

if (this.execute != null)

this.execute(parameter);

}

public bool CanExecute(object parameter)

{

return this.canExecute != null ? this.canExecute(parameter) : true;

}

}

public class CommandDelegate<T> : ICommand

{

private Action<T> execute;

private Func<T, bool> canExecute;

public event EventHandler CanExecuteChanged

{

add { CommandManager.RequerySuggested += value; }

remove { CommandManager.RequerySuggested += value; }

}

public CommandDelegate(Action<T> execute, Func<T, bool> canExecute = null)

{

this.execute = execute;

this.canExecute = canExecute;

}

public void Execute(object parameter)

{

if (this.execute != null)

this.execute((T)parameter);

}

public bool CanExecute(object parameter)

{

return this.canExecute != null ? this.canExecute((T)parameter) : true;

}}}

## Приложение Д

**Класс SaltedHash**

|  |
| --- |
| using System;  using System.Linq;  using System.Text;  using System.Security.Cryptography;  namespace GameWPF.Util.Hash  {  public class SaltedHash  {  public byte[] Hash { get; set; }  public byte[] Salt { get; set; }  public SaltedHash(string psw)  {  byte[] saltBytes = new byte[32];  new Random().NextBytes(saltBytes);  Salt = Encoding.UTF8.GetBytes(Convert.ToBase64String(saltBytes));  byte[] passwordAndSaltBytes = Concat(psw, saltBytes);  Hash = ComputeHash(passwordAndSaltBytes);  }  public static bool Verify(byte[] hash, byte[] salt, string psw)  {  byte[] saltBytes = Convert.FromBase64String(Encoding.UTF8.GetString(salt));  var passwordAndSaltBytes = Concat(psw, saltBytes);  var hashAttempt = ComputeHash(passwordAndSaltBytes);  return Enumerable.SequenceEqual(hash, hashAttempt);  }  private static byte[] ComputeHash(byte[] passwordAndSaltBytes)  {  using (SHA256 sha256 = SHA256.Create())  {  return Encoding.UTF8.GetBytes(Convert.ToBase64String(sha256.ComputeHash(passwordAndSaltBytes)));  }  }  private static byte[] Concat(string password, byte[] saltBytes)  => Encoding.UTF8.GetBytes(password).Concat(saltBytes).ToArray();  }  } |