МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждения образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ

ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет информационных технологий

Кафедра программной инженерии

Специальность 1-40 01 01 Программное обеспечение информационных технологий

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

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**КУРСОВОГО ПРОЕКТА:**

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

Тема” Программное средство для предоставления доступа к файлам”

Исполнитель

студент 2 курса группы 5 Ворович Кирилл Сергеевич

(Ф.И.О.)

Руководитель работы ассистент Панченко О.Л.

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

Курсовой проект защищен с оценкой

Председатель Смелов В.В.

(подпись)

Минск 2024

# Содержание

[Содержание 2](#_Toc166834699)

[Введение 4](#_Toc166834700)

[1. Постановка задачи и обзор литературы 5](#_Toc166834701)

[1.1 Анализ аналогов и прототипов 5](#_Toc166834702)

[1.2 Постановка задачи 7](#_Toc166834703)

[1.3 Вывод по разделу 7](#_Toc166834704)

[2. Анализ требований к программному средству и разработке функциональных требований 9](#_Toc166834705)

[2.1 Описание средств разработки 9](#_Toc166834706)

[2.1.1 Microsoft Visual Studio 2019 9](#_Toc166834707)

[2.1.2 Программная платформа .NET Framework 9](#_Toc166834708)

[2.1.3 Язык программирования C# 10](#_Toc166834709)

[2.1.4 Технология WPF 10](#_Toc166834710)

[2.1.5 Расширяемый язык разметки XAML 10](#_Toc166834711)

[2.1.6 Технология Entity Framework 10](#_Toc166834712)

[2.1.7 MS SQL Server 11](#_Toc166834713)

[2.1.8 Паттерн MVVM 11](#_Toc166834714)

[2.2 Спецификация функциональных требований к программному средству 12](#_Toc166834715)

[2.3 Вывод по разделу 12](#_Toc166834716)

[3. Проектирование программного средства 13](#_Toc166834717)

[3.1 Архитектура системы 13](#_Toc166834718)

[3.2 Взаимоотношения между классами 14](#_Toc166834719)

[3.3 Модель базы данных 14](#_Toc166834720)

[3.4 Проектирование последовательности взаимодействия 17](#_Toc166834721)

[3.5 Вывод по разделу 17](#_Toc166834722)

[4. Реализация программного средства 19](#_Toc166834723)

[4.1 Основные классы программного средства 19](#_Toc166834724)

[4.2 Авторизация 19](#_Toc166834725)

[4.3 Регистрация 19](#_Toc166834726)

[4.4 Добавление, удаление, редактирование постов. 20](#_Toc166834727)

[4.5 Отправка запроса подписку. 20](#_Toc166834728)

[4.6 Добавление, редактирование подписки. 20](#_Toc166834729)

[4.7 Принятие/отклонение запросов на подписку. 21](#_Toc166834730)

[4.8 Добавление комментариев. 21](#_Toc166834731)

[5. Тестирование, проверка работоспособности и анализ полученных результатов 22](#_Toc166834732)

[5.2 Тестирование регистрации 23](#_Toc166834733)

[5.3 Тестирование возможности добавления страницы автора 24](#_Toc166834734)

[5.4 Тестирование добавления уровней подписки 24](#_Toc166834735)

[5.5 Тестирование возможности добавления постов 25](#_Toc166834736)

[5.6 Тестирование возможности отправки комментариев 26](#_Toc166834737)

[5.7 Тестирование возможности отправки запроса на подписку 26](#_Toc166834738)

[5.8 Тестирование скрытия не доступных пользователю постов 27](#_Toc166834739)

[5.9 Тестирование возможности принятия запроса на подписку 27](#_Toc166834740)

[5.10 Выводы по разделу 28](#_Toc166834741)

[ЗАКЛЮЧЕНИЕ 29](#_Toc166834742)

[Список использованных источников 30](#_Toc166834743)

[ПРИЛОЖЕНИЕ А 31](#_Toc166834744)

[ПРИЛОЖЕНИЕ Б 32](#_Toc166834745)

[ПРИЛОЖЕНИЕ В 33](#_Toc166834746)

[ПРИЛОЖЕНИЕ Г 34](#_Toc166834747)

# Введение

В эпоху цифровизации и беспрерывного роста объемов информации, возникает необходимость в разработке инновационных инструментов для эффективного доступа и управления данными. Особенно это касается организаций и индивидуальных пользователей, для которых важен быстрый и безопасный доступ к файлам и контенту. В этом контексте, разработка программного средства, позволяющего предоставлять доступ к файлам, становится актуальной и востребованной задачей. Такой инструмент не только обеспечивает пользователей необходимым контентом, но и вносит значительный вклад в оптимизацию рабочих процессов, повышая общую производительность и удобство в использовании.

Целью данного курсового проекта является создание программного средства, которое будет предоставлять пользователям доступ к файлам и контенту, удовлетворяя при этом современные требования безопасности, масштабируемости и удобства использования. Разрабатываемое программное средство должно стать надежным инструментом в руках пользователей, позволяя им эффективно управлять доступом к нужным файлам и контенту, а также способствовать оптимизации рабочих и творческих процессов.

Для достижения поставленной цели, проект предусматривает решение следующих задач:

* Разработка удобного и интуитивно понятного интерфейса для пользователей, обеспечивающего легкий доступ к файлам и контенту.
* Обеспечение высокого уровня безопасности доступа к файлам, включая аутентификацию пользователей и шифрование данных.
* Реализация функционала для управления доступом к файлам, позволяя пользователю настраивать права доступа для различных категорий контента.
* Интеграция с современными облачными хранилищами данных для обеспечения масштабируемости и надежности хранения информации.

Основной целью курсового проекта является создание программного средства, которое удовлетворит все поставленные требования и задачи. Для этого будут использованы принципы объектно-ориентированного программирования, база данных MS SQL Server для хранения данных и технология Windows Presentation Foundation (WPF) для создания графического интерфейса.

# 1. Постановка задачи и обзор литературы

## 1.1 Анализ аналогов и прототипов

При разработке приложения для музыкантов, важно рассмотреть существующие аналоги и альтернативные решения, которые также предназначены для решения подобных задач. Этот обзор позволяет выявить преимущества и недостатки конкурирующих продуктов и определить, каким образом программное средство может выделиться на фоне аналогов.

patreon.com: Одна из крупнейших зарубежных веб-сайтов, на котором авторы творческих произведений могут распространять свои работы по платной подписке или предоставлять дополнительный контент для своих подписчиков. Они конкурируют напрямую с сервисом подписки на контент, предлагая аналогичные услуги. Рисунок 1.1.

|  |
| --- |
|  |

Рисунок 1.1 – Страница автора на сайте “patreon.com”

К недостаткам сайта можно отнести сильную перегруженность кнопками.

У сайта есть большое количество преимуществ: возможность установить картинки и название для подписок, минималистичный дизайн.

boosty.to: Еще один сервис подписки на контент, распространен среди российских авторов. Они также предстовляют аналогичные услуги. Рисунок 1.2.

|  |
| --- |
|  |

Рисунок 1.2 – Страница автора на сайте “boosty.to”

Вид страницы добавления постов представлен на рисунке 1.3.

На странице сразу видно описание автора, шапка страницы, иконка страницы, возможные подписки. На сайте отсутствует поиск авторов, поэтому автора можно найти только через ссылки со сторонних сайтов.

|  |
| --- |
|  |

Рисунок 1.3 – Страница добавления постов на сайте “boosty.to”

sponsr.ru: Менее известный сервис подписки на контент.  Платформа для размещения всех видов медиа-контента и заработка на подписках. Они также предлагают аналогичные услуги. Рисунок 1.4.

|  |
| --- |
|  |

Рисунок 1.4 – Страница поиска на сайте “ sponsr.ru ”

Стоит отметить, что данный сайт имеет возможность разделения авторов по категориям.

## 1.2 Постановка задачи

На основе анализа аналогичных решений и с учетом темы проекта был составлен список ролей пользователей и список функций, которые должно выполнять разрабатываемое приложение:

Основные задачи:

* создать каталог страниц авторов с возможностью перехода на страницу;
* разработать функционал для поиска страниц автора;
* разработать профиль пользователя, где будет располагаться созданных пользователем страниц, имя, почта, иконка пользователя;
* разработать добавления постов на страницу;
* разработать управления постами странице;
* провести тестирование программного средства;
* составить руководство по установке и использованию приложения.

## 1.3 Вывод по разделу

После изучения аналогичных продуктов были выявлены их основные преимущества и недостатки, что послужило основой для составления списка функций. Поскольку большинство аналогов обладают схожим функционалом, основное внимание было уделено красивому дизайну и удобству использования. Цель заключалась в том, чтобы обеспечить такой интерфейс, который был бы интуитивно понятен для любого пользователя, даже если он имел опыт работы лишь с одним из подобных приложений.

# Анализ требований к программному средству и разработке функциональных требований

Анализ требований в проекте программного обеспечения включает сбор, систематизацию и документирование требований от заказчика. Цель этого процесса - получить полное понимание задачи заказчика, определить границы проекта, выявить возможные проблемы и риски.

На этом этапе определяются основные методологические и технические требования, формулируются цели и задачи проекта, а также выявляются ключевые факторы успеха. Определение и описание требований являются критическими шагами, которые влияют на успешное завершение всего проекта, поскольку они определяют основу для всех последующих этапов. Этот процесс начинается с описания используемых технических средств, затем переходит к архитектуре системы и созданию диаграмм использования.

2.1 Описание средств разработки

Для разработки программного средства будут использоваться следующие инструменты:

* Среда разработки Microsoft Visual Studio 2022;
* Программная платформа .NET;
* Язык программирования C#;
* Технология WPF;
* Microsoft SQL Server MS 19;
* Entity Framework (для создания и взаимодействия с базой данных).

### **2.1.1 Microsoft Visual Studio 2019**

Microsoft Visual Studio 2019 - это интегрированная среда разработки (IDE) от Microsoft, которая обеспечивает разработчиков инструментами и возможностями для создания приложений с использованием Windows Presentation Foundation (WPF)[1].

### **2.1.2 Программная платформа .NET Framework**

.NET Framework, разработанный Microsoft в 2002 году, представляет собой технологию, которая обеспечивает поддержку разработки и запуска веб-служб и приложений для операционной системы Windows. Он предоставляет согласованную среду объектно-ориентированного программирования для сохранения и выполнения объектного кода как локально, так и удаленно через Интернет. Основой платформы является Common Language Runtime, общая среда выполнения, которая способна интерпретировать код на различных поддерживаемых языках программирования. [2].

### **2.1.3 Язык программирования C#**

В .NET Framework основным языком программирования является C#. Этот объектно-ориентированный язык обладает строгой статической типизацией и поддерживает такие возможности, как перегрузка операторов, указатели на функции-члены классов, атрибуты, события, свойства и исключения. Он также широко используется в технологии WPF в качестве основного языка программирования. [3].

### **2.1.4 Технология WPF**

Для создания пользовательского интерфейса и разделения дизайна от бизнес-логики в .NET Framework используется технология Microsoft WPF (Windows Presentation Foundation). WPF является альтернативой технологии WinForms и представляет собой среду для разработки клиентских приложений Windows с возможностями взаимодействия с пользователем. Она также включает графическую подсистему и использует язык разметки XAML (Extensible Application Markup Language).

### **2.1.5 Расширяемый язык разметки XAML**

WPF (Windows Presentation Foundation) предоставляет различные инструменты для создания визуального интерфейса, включая язык XAML (eXtensible Application Markup Language). XAML позволяет описывать элементы управления, привязку данных, макеты, двухмерную и трехмерную графику, анимацию, стили, шаблоны, документы, текст, мультимедиа и оформление в декларативной форме. XAML основан на XML и используется в WPF для определения различных аспектов пользовательского интерфейса.

**2.1.6 Технология Entity Framework**

Для взаимодействия с базой данных в приложении WPF используется технология Entity Framework Core — кроссплатформерная и расширяемая версия технологии Entity Framework с открытым исходным кодом, которая в текущий момент развивается и дополняется.

Если традиционные средства ADO.NET позволяют создавать подключения, команды и прочие объекты для взаимодействия с базами данных, то Entity Framework представляет собой более высокий уровень абстракции, который позволяет абстрагироваться от самой базы данных и работать с данными независимо от типа хранилища[2]. Entity Framework Core – это последняя версия Entity Framework 7, которая совместима с .NET Core и на данный момент продолжает развиваться. Entity Framework Core предполагает три возможных способа взаимодействия с базой данных:

* Database First: Entity Framework Core создает набор классов, которые отражают модель конкретной базы данных;
* Model First: сначала разработчик создает модель базы данных, по которой затем Entity Framework Core создает реальную базу данных на сервере;
* Code First: разработчик создает класс модели данных, которые будут храниться в базе данных, а затем Entity Framework Core по этой модели генерирует базу данных и ее таблицы [4].

### **2.1.7 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 [5].

### **2.1.8 Паттерн MVVM**

MVVM (Model-View-ViewModel) - это паттерн архитектуры, используемый в разработке программного обеспечения для разделения пользовательского интерфейса (View) от бизнес-логики (Model) и связывания их через промежуточный слой ViewModel.

В MVVM модель представляет данные и бизнес-логику приложения, которая может включать в себя операции чтения/записи данных из и в источники данных, такие как база данных или веб-сервисы.

Представление (View) отвечает за отображение данных и взаимодействие с пользователем. Оно не содержит бизнес-логику и должно быть максимально независимым от модели.

ViewModel представляет промежуточный слой между View и Model. Он содержит логику, необходимую для обработки пользовательских действий, обновления данных в модели и уведомления View об изменениях. ViewModel предоставляет свойства и команды, которые привязываются к элементам пользовательского интерфейса во View, позволяя им отображать данные и реагировать на действия пользователя.

MVVM позволяет достичь разделения ответственности между компонентами приложения, облегчает тестирование и повышает переиспользуемость кода. Он также способствует лучшей поддержке параллельной разработки пользовательского интерфейса и бизнес-логики, так как разработчики могут работать независимо над своими частями приложения.[6]

2.2 Спецификация функциональных требований к программному средству

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

Функции гостя:

* + Выполнение регистрации, авторизации;
  + Просмотр списка авторов;
  + Возможность поиска автора по имени;
  + Просмотр постов, не требующих подписки.

Функции Пользователь:

* + Возможность отправить запрос на подписку на автора;
  + Возможность оставить комментарий;
  + Возможность настроить профиль;
  + Возможность создать свои страници автора.

Функции автора:

* + Публикация постов;
  + Настройка страницы автора;
  + Удаление постов;
  + Удаление подписчиков;
  + Настройка уровней подписки;
  + Возможность настроить профиль;
  + Возможность принять запрос на подписку.

Функции администратора:

* + Добавление, удаление, редактирование постов пользователей;
  + Управления подписками пользователей;
  + Удаление страниц автора;
  + Удаление пользователей.

Причём, функции гостя наследует авторизованный пользователь, а функции авторизованного пользователя наследует администратор. Автор наследует функции пользователя.

В приложении А представлена диаграмма использования программного средства разными группами пользователей.

В данной главе были определены роли и группы пользователей, а также представлена диаграмма использования программного средства разными группами пользователей. В целом, спецификация функциональных требований к программному средству позволяет определить функциональность и ожидания пользователей, а также задает рамки для разработки и тестирования программного продукта.

2.3 Вывод по разделу

В данном разделе были определены требования к проекту, составлена User Case диаграмма, а также средства, которые будут использоваться в процессе разработки и функционирования программного средства.

3. Проектирование программного средства

3.1 Архитектура системы

В данном курсовом проекте используется архитектурный паттерн MVVM (Model-View-ViewModel). Он позволяет отделить логику приложения от визуальной части. MVVM состоит из трех частей:

* модель описывает используемые в приложении данные;
* представление определяет визуальный интерфейс, через который пользователь взаимодействует с приложением;
* модель представления связывает модель и представление через механизм привязки данных. ViewModel также содержит логику по получению данных из модели, которые потом передаются в представление.

Программное средство для предоставления доступа к файлам имеет следующую структуру, представленную на рисунке 3.1.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| |  |  |  | | --- | --- | --- | |  |  |  | |  |  |
|  |  |  |

Рисунок 3.1 – Структура проекта

Описание структуры основных папок проекта представлено в таблице 3.1.

Таблица 3.1 – Папки верхнего уровня

|  |  |
| --- | --- |
| Название папки/файла | Содержание |
| Resources | Папка, содержащая ресурсы для проекта |
| Utilities | Содержит разные утилиты, которые могут использовать как в View, так и в ViewModel |
| MVVM | Папка, содержащая компоненты, составляющие архитектуру приложения |

Более подробно можно рассмотреть содержимое папок “MVVM” и “Utilities”. Описание этих папок представлены в таблицах 3.2 и 3.3.

Таблица 3.2 – Папки в директории “MVVM”

|  |  |
| --- | --- |
| Название | Описание |
| Model | Хранит модели данных |
| View | Хранит представления окон |
| ViewModel | Хранит реализацию ViewModel |

Таблица 3.3 – Папки в директории “Utilities”

|  |  |
| --- | --- |
| Название | Описание |
| BaseModel | Класс для манипуляции базой данных |
| NavigationService | Содержит классы для реализации навигации между окнами |
| Store | Содержит классы, реализующие обмен данных между ViewModel |
| Service | Содержит сервисы, которые используются в классах ViewModel |

Таким образом, сформированная таблица помогает понять общую структуру проектируемого программного средства.

3.2 Взаимоотношения 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. Диаграмма классов представлена в приложении Б.

3.3 Модель базы данных

Для реализации поставленной задачи была создана база данных SUBSCRIPTIONBD. Для ее создания использовалась система управления реляционными базами данных MS SQL Server. В дальнейшем база данных будет создаваться с помощью Entity Framework и подхода Database-First. Database-First это один из подходов к разработке программного обеспечения, особенно к разработке приложений, которые используют базы данных. В этом подходе сначала проектируется и создается база данных, а затем на основе её структуры автоматически генерируется модель приложения (например, объектно-реляционная модель данных). Таким образом, база данных служит источником исходных данных для создания приложения. База данных состоит из 9 таблиц: Users, Logins, Files, Posts, AuthorPages, Comments, Requests, Subscriptions, Subscription\_types. Диаграмма базы данных представлена на рисунке 3.2.

|  |
| --- |
|  |

Рисунок 3.2 – Диаграмма базы данных

На рисунке 3.3 изображена структура таблицы Users, которая содержит информацию о пользователях.

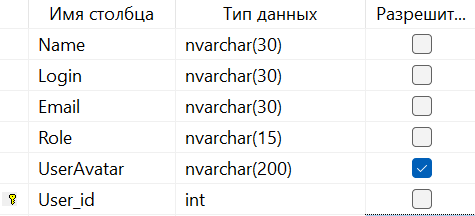


Рисунок 3.3 – Структура таблицы Users

На рисунке 3.4 приведена структура таблицы AuthorPages, содержащая информацию о страницах автора.

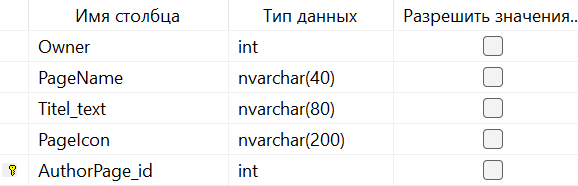


Рисунок 3.4 – Структура таблицы AuthorPages

На рисунке 3.5 изображена структура таблицы Logins, которая хранит информацию о логине и пароле пользователя.

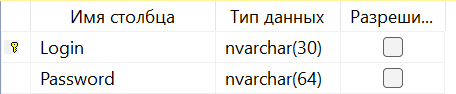


Рисунок 3.5 – Структура таблицы Logins

На рисунке 3.3 изображена структура таблицы Files, которая содержит информацию о файлах для постов.

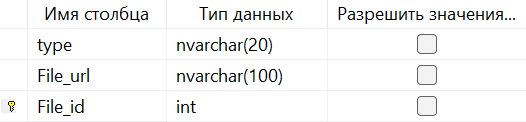


Рисунок 3.3 – Структура таблицы Files

На рисунке 3.3 изображена структура таблицы Posts, которая содержит информацию о постах.

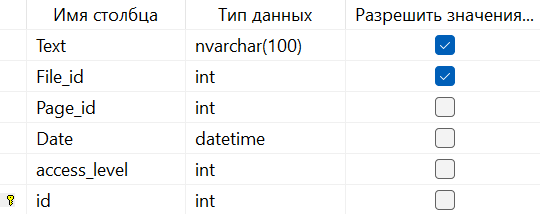


Рисунок 3.3 – Структура таблицы Posts

На рисунке 3.3 изображена структура таблицы Comments, которая содержит информацию о комментариях пользователей.

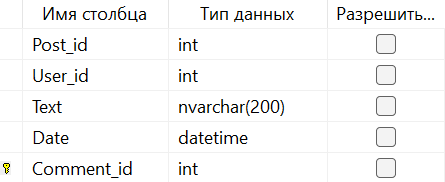


Рисунок 3.3 – Структура таблицы Сomments

На рисунке 3.3 изображена структура таблицы Requsts, которая содержит запросы на подписку.

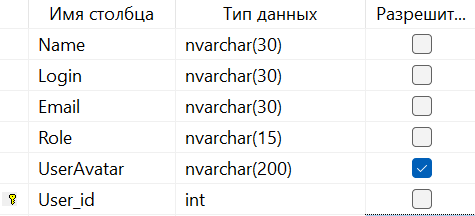


Рисунок 3.3 – Структура таблицы Requsts

На рисунке 3.3 изображена структура таблицы Subscriptions, которая содержит информацию о подписках.

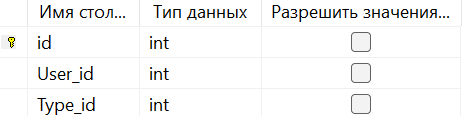


Рисунок 3.3 – Структура таблицы Subscriptions

На рисунке 3.3 изображена структура таблицы Subscription\_types, которая содержит информацию о типах подписок.

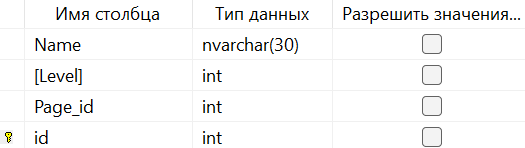


Рисунок 3.3 – Структура таблицы Subscription\_types

3.4 Проектирование последовательности взаимодействия

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

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

Диаграмма последовательности взаимодействия пользователя с приложением представлена в Приложении В.

3.5 Вывод по разделу

В данном разделе представлена структура программного средства, включающая основные элементы с подробными пояснениями к каждому из них. Описана диаграмма классов, которая иллюстрирует взаимосвязи между различными сущностями в программном средстве, а также модификаторы доступа, необходимые для их функционирования. Также разработана диаграмма последовательности, которая используется для описания процессов бизнес-логики, взаимодействия между компонентами системы, а также процессов ввода-вывода и взаимодействия между пользователем и системой. Эти диаграммы помогают в понимании архитектуры и функционирования программного средства, а также облегчают коммуникацию между разработчиками и другими участниками проекта.

# 4. Реализация программного средства

## 4.1 Основные классы программного средства

Для выполнения технических задач программное средство для предоставления доступа к файлам должны быть реализованы следующие функции и соответствующие им классы и методы:

* авторизация;
* регистрация;
* добавление/удаление/редактирование постов;
* добавление/редактирование уровней подписки;
* отправка запросов на подписку;
* принятие/отклонение запросов на подписку;
* добавление комментариев.

## 4.2 Авторизация

Класс Store содержит текущего пользователя, необходимый функционал для обеспечения возможности входа пользователя в свой.

Данный класс содержит метод Login, активируемый при клике на кнопку "Войти". Данный метода использует HashPassword, который преобразует пароль пользователя в хэш для проверки его соответствия с паролем в базе данных. При успешной авторизации, когда логин и пароль найдены в базе, пользователь направляется на домашнюю страницу. В случае неудачи выводится уведомление об ошибке.

Листинг реализации класса Store приведен в приложении Г.

## 4.3 Регистрация

Для регистрации нового аккаунта пользователем, в классе RegisterVM реализован метод RegisterUser. При нажатии пользователем кнопки "Регистрация" вызывается основной метод класса Validator - Validate(). В этом методе все введенные пользователем данные проходят проверку на корректность с использованием регулярных выражений, а также проверяется, что все поля заполнены.

После успешного прохождения указанных проверок происходит попытка добавить пользователя в базу данных, и пользователь перенаправляется на страницу входа. Если логин уже зарегистрированы в базе данных, то пользователь получает соответствующее уведомление об ошибке.

Листинг реализации метода RegisterUser и класса Validator приведен в приложении Г.

## 4.4 Добавление, удаление, редактирование постов.

Для добавления поста необходима нажать на кнопку “Создать пост” на страницу автора. После открывается форма для создания поста. Для создания поста нужно заполнить поле “Текст”, выбрать файл и выбрать уровень подписки. Для сохранения поста необходимо нажать кнопку “Сохранить”. После нажатия на кнопку выполниться метод Save() из класс CreatePostVM(). Данный метод сохрани пост в таблицу базы данных Posts.

Для удаления поста необходимо нажать на иконку по постом с изображением мусорного ведра. После нажатия на иконку выполниться метод DeletePost() из класса Posts. Данный метод удаляет пост из базы данных.

Для изменения поста необходимо нажать на иконку по постом с изображением карандаша. После нажатия на иконку выполниться метод EditPost() из класса Posts. Данный метод открывает страницу изменения поста. подписки. Для сохранения поста необходимо нажать кнопку “Сохранить”. После нажатия на кнопку выполниться метод Save() из класс CreatePostVM(). Данный метод сохрани пост в таблицу базы данных Posts и выполнит метод CopyFileToFolder() из класс FileService. Выбранный файл сохраниться в папку приложение/Resources/Data/Files/idфайла/

Листинг методов Save() и DeletePost () и класс FileService представлен в приложении Г.

## 4.5 Отправка запроса подписку.

Для отправки запроса на подписку необходима нажать на кнопку “Подписаться” на странице автора. После открывается форма для выбора подписки. После нажатия на кнопку “Отправить запрос” и выбора подписки выполниться метод CreateRequest() из класса SubscriptionService. Данный метод сохранит запрос в таблицу базы данных Requests.

Листинг класс SubscriptionService представлен в приложении Г.

## 4.6 Добавление, редактирование подписки.

Для добавления подписки необходима нажать на кнопку “Добавить” на странице редактирования страницы автора. После открывается форма для создания подписки. Для создания подписки нужно заполнить поле “Название”,”Уровень подписки”. Для сохранения тип подписки необходимо нажать кнопку “Сохранить”. После нажатия на кнопку выполниться метод Save() из класс CreateSubscriptionTypesVM (). Данный метод сохранит тип подписки в таблицу базы данных Subscription\_Types.

Для изменения подписки необходимо нажать на тип подписки на странице редактирования страницы автора. После нажатия открывается страница изменения подписки. Для сохранения типа подписки необходимо нажать кнопку “Сохранить”. После нажатия на кнопку выполниться метод Save() из класс CreateSubscriptionTypesVM (). Данный метод сохрани пост в таблицу базы данных Posts.

Листинг метода Save() из CreateSubscriptionTypesVM представлен в приложении Г.

## 4.7 Принятие/отклонение запросов на подписку.

Для принятия запроса необходима нажать на иконку “галочка”. После выполниться Accept() из класс Requests. Данный метод удаляет запрос из базы данных и создает подписку в таблице Subscriptions.

Для отклонения запроса необходима нажать на иконку “крестик”. После выполниться Reject() из класс Requests. Данный метод удаляет запрос из базы данных.

Листинг методов Accept(), Reject() из класс Requests представлен в приложении Г.

## 4.8 Добавление комментариев.

Чтобы оставить комментарий необходимо нажать на иконку комментария под постом. После откроется страница комментариев. Пользователь может просмотреть оставленные другими пользователями комментарии.

Пользователь может оставить свой комментарий нажав на кнопку отправить. После нажатия на кнопку выполниться метод Send(). Данный метод получает данные текстового поля, соответствующего полю Text, из пользовательского интерфейса. Он создает объект контекста базы данных с указанием идентификатора пользователя. После этого поле ввода комментария очищается, и пользователь может повторить процедуру добавления комментария.

Листинг метода Send() представлен в приложении Г.

5. Тестирование, проверка работоспособности и анализ полученных результатов

**5.1 Тестирование авторизации**

В процессе авторизации пользователь может ввести пароль, не соответствующий действительному или несуществующий логин, в таком случае приложение выдаст пользователю ошибку.

Результат данного теста представлен на рисунке 5.1.

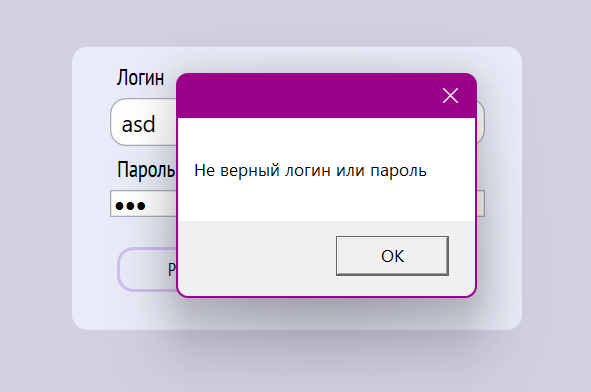


Рисунок 5.1 — Результат первого тестирования

После успешного входа, имя пользователя и иконка отобразится в шапке приложения. Рисунок 5.2.

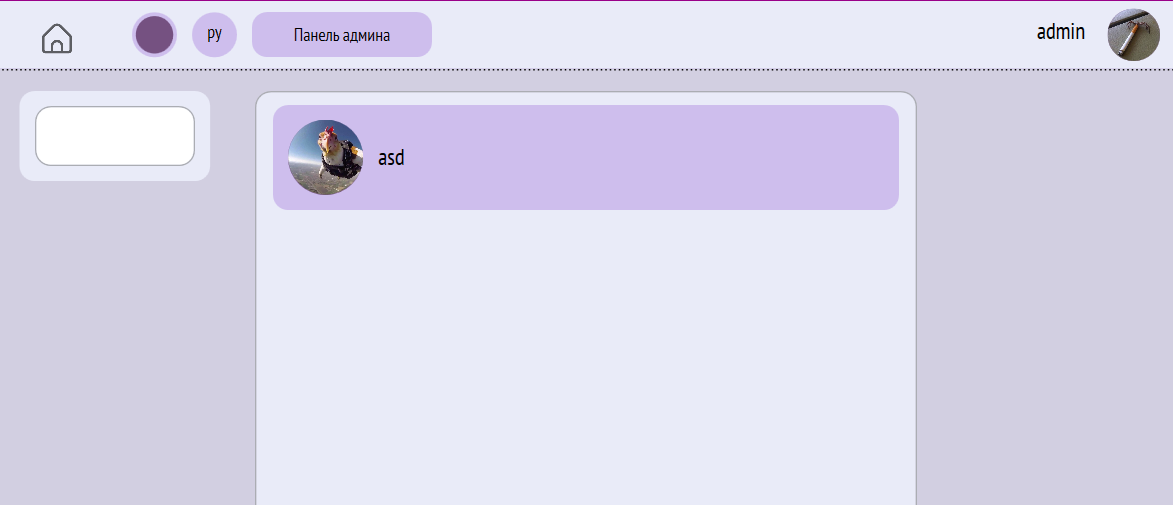


Рисунок 5.2 — Главная страница

**5.2 Тестирование регистрации**

При регистрации пользователь может совершить попытку использования уже существующего логина. В таком случае приложение оповестит пользователя об ошибке. Результат данного теста представлен на рисунке 5.3.

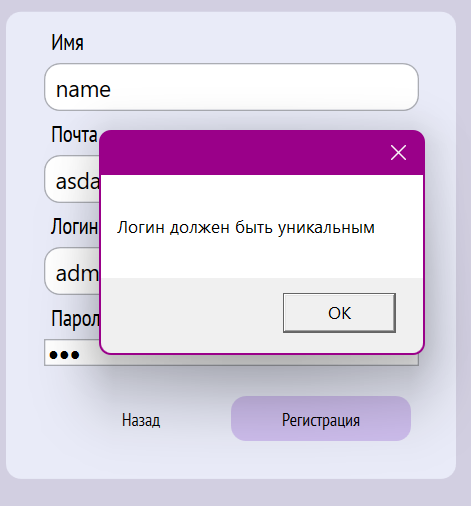


Рисунок 5.3 — Результат первого тестирования регистрации

Также во время регистрации пользователь может ввести некорректные значения в поля или оставить их пустыми. Проводится тестирование данной ситуации – пользователь вводит некорректные значения. Результат данного тестирования представлен на рисунке 5.4.

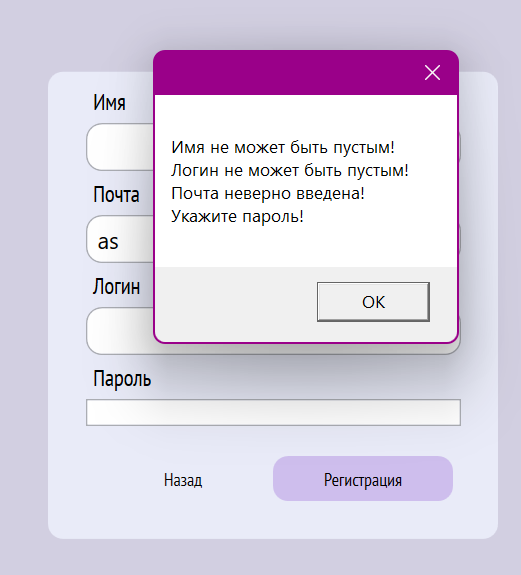


Рисунок 5.4 — Результат второго тестирования регистрации

## 5.3 Тестирование возможности добавления страницы автора

Если авторизованный пользователь хочет свою станицу, то он переходит в окна пользователя, нажимает на кнопку создать. Заполняет поля и выбирает иконку. Все поля должны быть заполнены. В ином случае будет выведено оповещение, как это показано на рисунке 5.5.

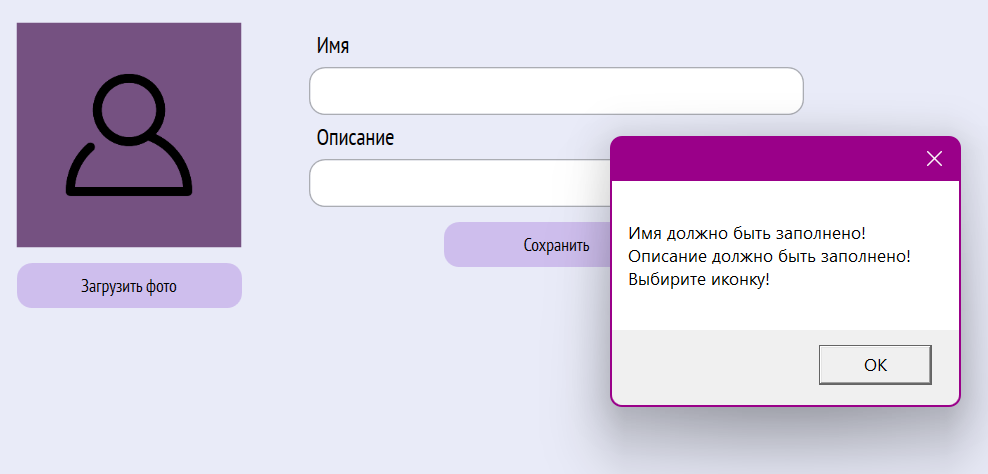


Рисунок 5.5 – Окно добавления страницы автора

## 5.4 Тестирование добавления уровней подписки

Если автор хочет добавить на свою станицу тип подписки, то он переходит в на страницу изменения страницы, нажимает на кнопку добавить. Заполняет поля и нажимает сохранить. Все поля должны быть заполнены. В ином случае будет выведено оповещение, как это показано на рисунке на рисунок 5.6.

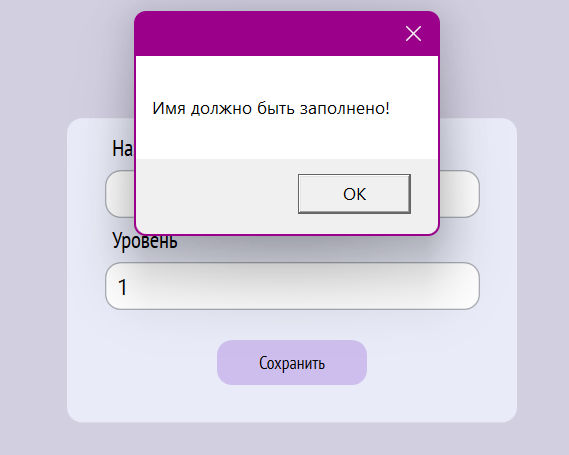


Рисунок 5.6 – Окно добавления страницы

## 5.5 Тестирование возможности добавления постов

Автор может добавить пост на свою страницу. Для добавления необходима нажать на кнопку “создать пост” на странице автора. После заполнения палея, выбора подписки, выбора файла и нажатия кнопки сохранить пост будет добавлен на страницу. Поле текст и файл должны быть выбраны, подписка может быть не выбрана, тогда пост будет доступен все пользователям. В ином случае система оповестить пользователя, что видно на рисунке 5.7.

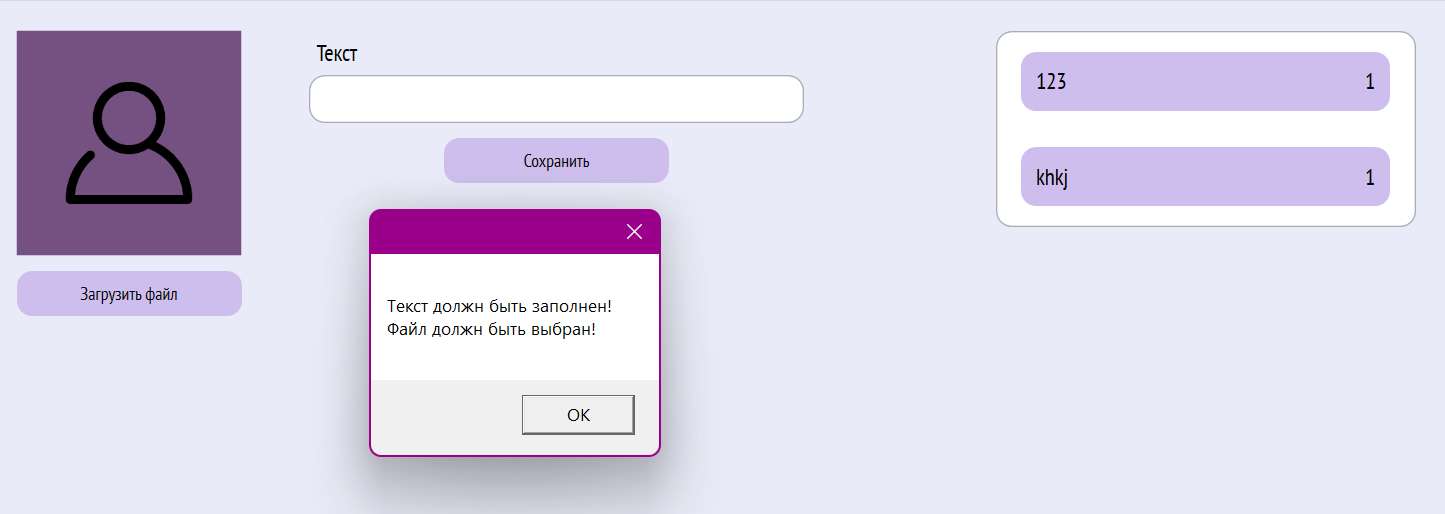


Рисунок 5.7 – Окно добавления постов

После нажатия на кнопку сохранить пост будет добавлен в база данных, и пост будет виден на странице. Рисунок 5.8.

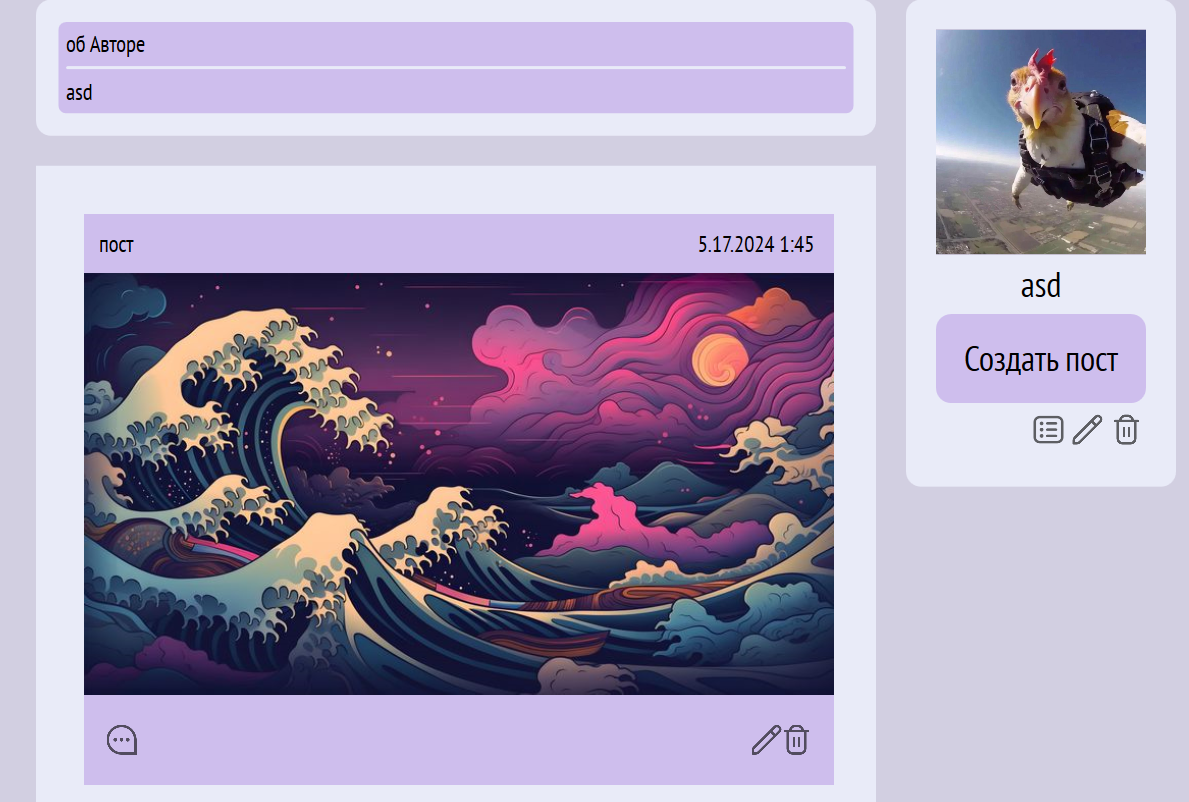


Рисунок 5.8 – Страница автора

## 5.6 Тестирование возможности отправки комментариев

У авторизованного пользователя есть возможность оставить комментарий. Для это необходимо ввести текст комментария и нажать кнопку отправить, после чего комментарий будет добавлен. Если поле пустое, то комментарий не будет добавлен. Рисунок 5.9.

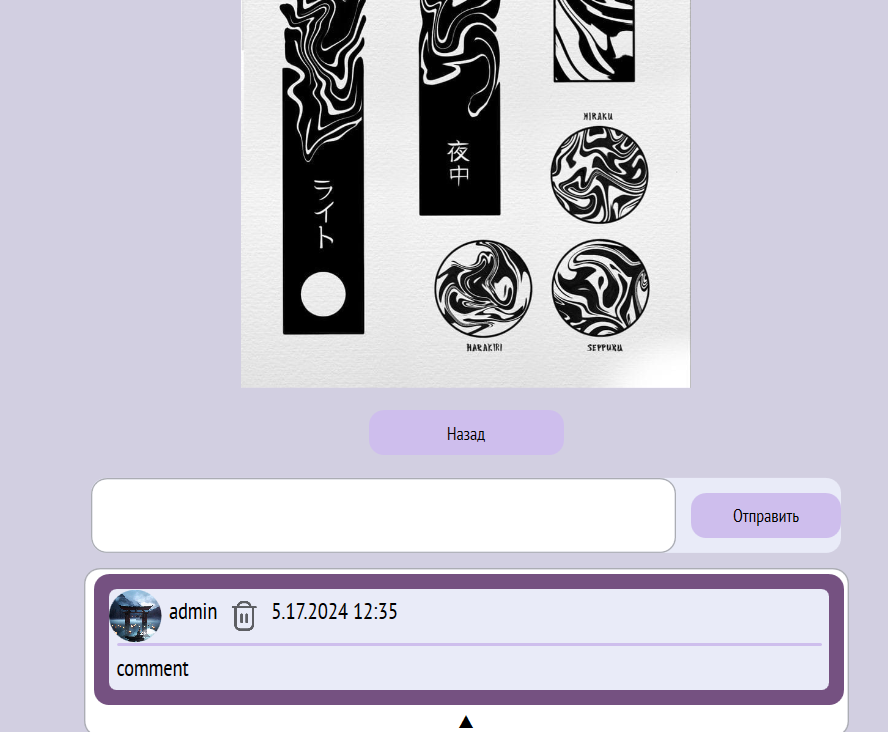


Рисунок 5.9 – Страница комментариев

## 5.7 Тестирование возможности отправки запроса на подписку

У авторизованного пользователя есть возможность отправки запроса на подписку. Необходимо выбрать уровень подписки и нажать кнопку “отправить запрос”, на которую будет отправлен запрос. После успешной отправки будет выведено сообщение. Рисунок 5.9.

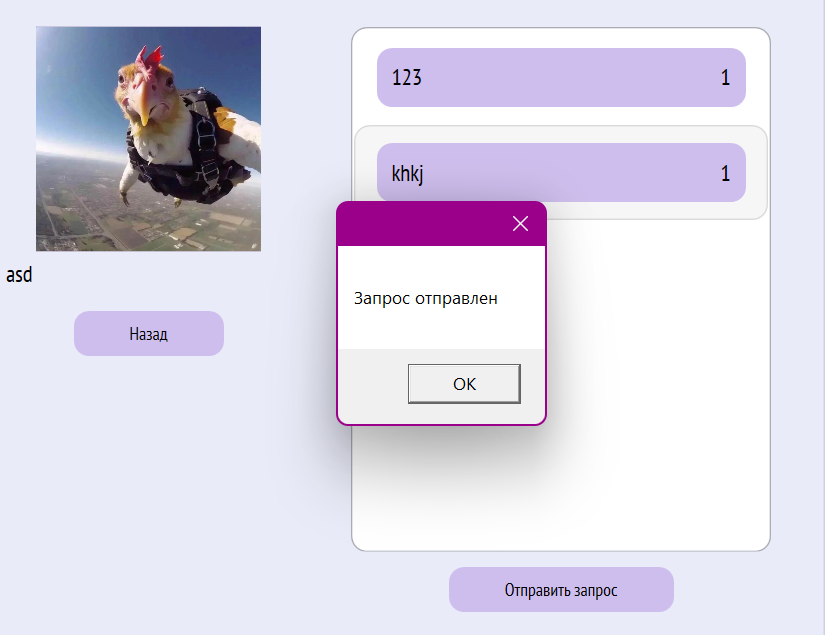


Рисунок 5.10 – Страница отправки запроса на подписку

## 5.8 Тестирование скрытия не доступных пользователю постов

Если у пользователя нету подписку, которая будет выше, или ровна уровню подписки необходимому посту, то пост будет не виден для пользователя. Рисунок 5.11.

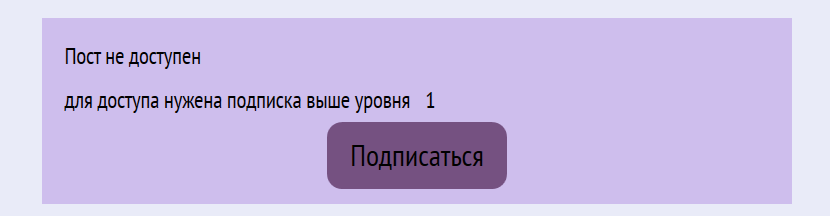


Рисунок 5.11 – Пост недоступный для пользователя

## 5.9 Тестирование возможности принятия запроса на подписку

Автор может принять, или отклонить запрос пользователя на подписку. Для принятия необходимо нажать на иконку галочки. Для отклонения необходимо нажать иконку с крестиком.

После принятия запроса пользователю будет добавлена подписка на страницу и удален запрос на подписку. После отклонения запрос будет удален. Рисунок 5.12.

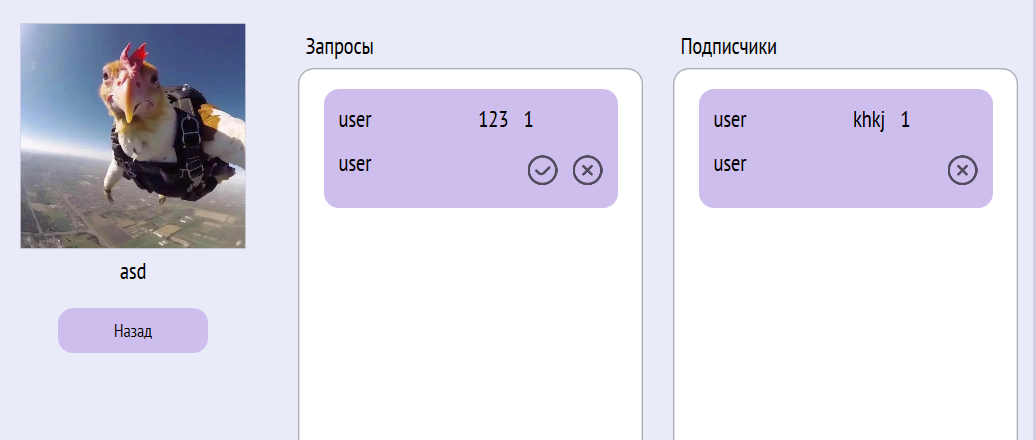


Рисунок 5.12 – Страница запросов

## 5.10 Выводы по разделу

Разными ролями были протестированы основные функции приложения, а именно возможности: добавления страницы авторы, добавления постов, регистрации и входа в аккаунт, отправка запросов на подписку, скрытие недоступных постов, отправку комментариев, принятие запросов.

По результатам тестирования можно сказать, что приложение работает правильно.

# ЗАКЛЮЧЕНИЕ

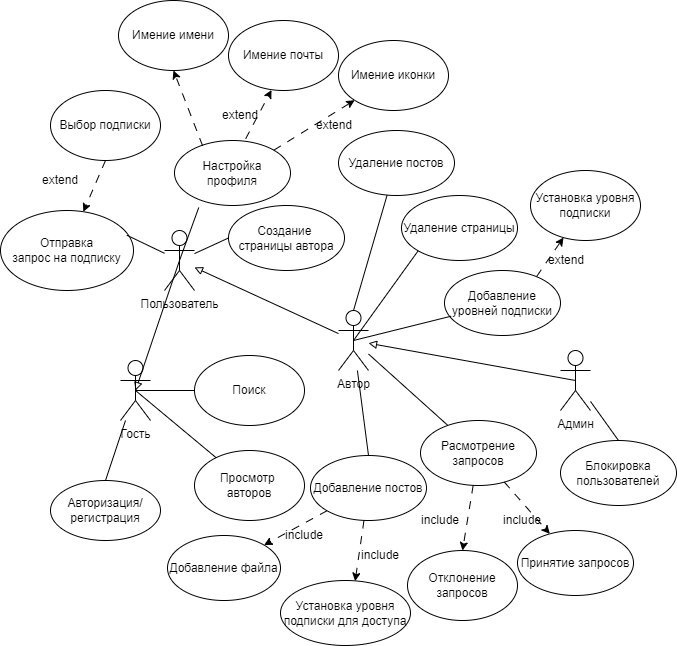
В итоге выполнения данного курсового проекта было разработано приложение для музыкантов. При разработке были выполнены все пункты из указанного списка предполагаемого основного функционала приложения. Также были реализованы дополнительные функции, которые лаконично вписываются и дополняют приложение.

Спроектирована и реализована эффективная архитектура проекта и её модулей, налажена не только логика взаимодействия с базой данных, а ещё и логики взаимодействия между разными компонентами приложения, создан алгоритм смены тональности песни, все основные функции приложения протестированы, а также, написано подробное руководство использования приложения согласно роли пользователя.

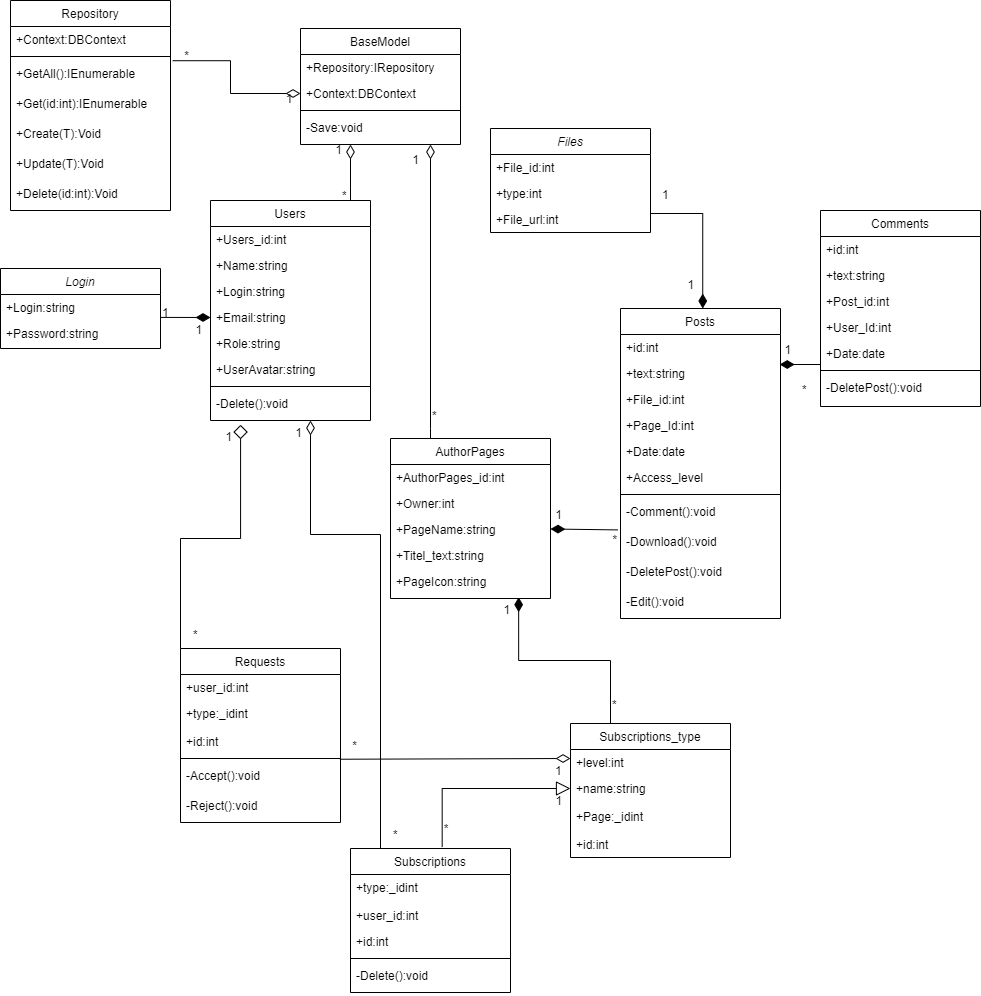
**Список использованных источников**

1. Microsoft Visual Studio [Электронный ресурс] – https://ru.wikipedia.org/wiki/Microsoft\_Visual\_Studio – Дата доступа 23.04.2023
2. Полное руководство по языку программирования C# 7.0 и платформе .NET 4.7. Режим доступа: https://metanit.com/sharp/tutorial/ – Дата доступа: 23.04.2023
3. Пацей, Н.В. Курс лекций по языку программирования C# / Н. В. Пацей. – Минск: БГТУ, 2018. – 175 с.
4. Руководство по WPF // [Электронный ресурс]. – Режим доступа: https://metanit.com/sharp/wpf/ – Дата доступа: 25.04.2023
5. Руководство по XAML // [Электронный ресурс]. – Режим доступа: https://www.tutorialspoint.com/xaml/index.htm– Дата доступа: 25.04.2023
6. Блинова, Е.А. Курс лекций по Базам данным / Е.А. Блинова. – Минск: БГТУ, 2019. – 175 с.

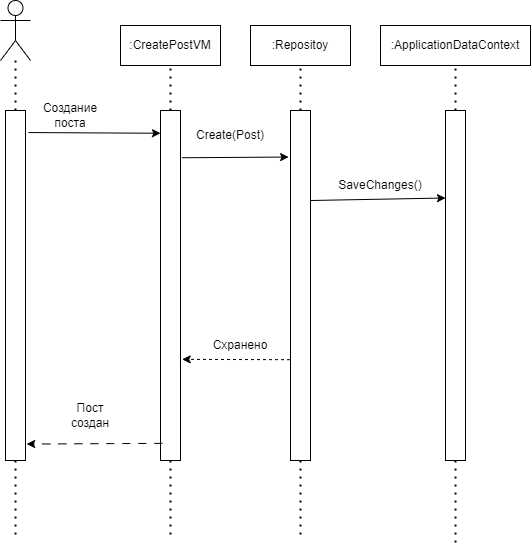
# ПРИЛОЖЕНИЕ А



**ПРИЛОЖЕНИЕ Б**



**ПРИЛОЖЕНИЕ В**



**ПРИЛОЖЕНИЕ Г**

Листинг класса Store

|  |
| --- |
| internal class Store  {  private static UserDataService \_currentUser;  static Store()  {  \_currentUser = new UserDataService();  }  public static UserDataService UserDataService  {  get  {  return \_currentUser;  }  }  public static Users User  {  get  {  return \_currentUser.CurrentUser;  }  }    public static bool Login(Logins \_login)  {  var user = BaseModel.Context.Logins.FirstOrDefault(l => l.Login == \_login.Login && l.Password == \_login.Password);  if (user != null)  {  \_currentUser.LoadUser(FindUserByLogin(\_login));  return true;  }  else  {  return false;  }  }  public static void SignOut()  {  \_currentUser.LoadUser(null);  }  private static Users FindUserByLogin(Logins login)  {  var user = BaseModel.Users.GetAll().FirstOrDefault(u => u.Login == login.Login);  if (user != null)  {  user.Logins.Password = login.Password;  return user;  }  else  return null;  }  } |

Листинг класса Validater

|  |
| --- |
| class Validater  {  public static bool IsValidImageFormat(string fileUrl)  {  var validImageExtensions = new HashSet<string>(StringComparer.OrdinalIgnoreCase)  {  ".jpg", ".jpeg", ".png", ".bmp", ".gif", ".tiff", ".jfif"  };  string fileExtension = Path.GetExtension(fileUrl);  return validImageExtensions.Contains(fileExtension);  }  public static string HashPassword(string password)  {  using (SHA256 sha256 = SHA256.Create())  {  byte[] bytes = sha256.ComputeHash(Encoding.UTF8.GetBytes(password));  StringBuilder builder = new StringBuilder();  for (int i = 0; i < bytes.Length; i++)  {  builder.Append(bytes[i].ToString("x2"));  }  return builder.ToString();  }  }  public static Regex eMailRegex = new Regex(@"^[\w-\.]+@([\w-]+\.)+[\w-]{2,4}$");  public static StringBuilder Validate(Users user)  {  StringBuilder error = new StringBuilder();  if (user.Name.IsNullOrEmpty())  error.Append("Имя не может быть пустым!\n");  else if (user.Name.Length >= 30)  error.Append("Длина имени не может быть больше 30!\n");  if (user.Logins.Login.IsNullOrEmpty())  error.Append("Логин не может быть пустым!\n");  else if (user.Logins.Login.Length >= 30)  error.Append("Длина логина не может быть больше 30!\n");  if (user.Email.IsNullOrEmpty())  error.Append("Почта не может быть пуста!\n");  else if (user.Email.Length >= 30)  error.Append("Длина почты не может быть больше 30!\n");  else if (!eMailRegex.IsMatch(user.Email))  error.Append("Почта неверно введена!\n");  return error;  }  public static StringBuilder Validate(Subscription\_type type)  {  StringBuilder error = new StringBuilder();  if (type.Name.IsNullOrEmpty())  error.Append("Имя должно быть заполнено!\n");  else if (type.Name.Length >= 30)  error.Append("Длина имени не может быть больше 30!\n");  return error;  }  public static StringBuilder Validate(AuthorPages page)  {  StringBuilder error = new StringBuilder();  if (page.PageName.IsNullOrEmpty())  error.Append("Имя должно быть заполнено!\n");  else if (page.PageName.Length >= 40)  error.Append("Длина имени не может быть больше 30!\n");  if (page.Titel\_text.IsNullOrEmpty())  error.Append("Описание должно быть заполнено!\n");  else if (page.Titel\_text.Length >= 40)  error.Append("Длина имени не может быть больше 30!\n");  if (page.PageIcon.IsNullOrEmpty())  error.Append("Выбирите иконку!\n");  return error;  }  public static StringBuilder Validate(Posts post)  {  StringBuilder error = new StringBuilder();  if (post.Text.IsNullOrEmpty())  error.Append("Текст должн быть заполнен!\n");  else if (post.Text.Length >= 100)  error.Append("Текст не может быть больше 100 символов\n");  if (post.Files.File\_url.IsNullOrEmpty())  error.Append("Файл должн быть выбран!\n");  return error;  }  } |

Листинг метода RegisterUser

|  |
| --- |
| private void RegisterUser()  {  NewUser.Role = "user";  StringBuilder errors = Validater.Validate(NewUser);  if (string.IsNullOrWhiteSpace(NewUser.Logins.Password))  errors.AppendLine("Укажите пароль!");  else if (NewUser.Logins.Password.Length > 30)  errors.AppendLine("Размер пороля должен быть не больше 30");  if (BaseModel.Context.Logins.Any((login => login.Login == NewUser.Logins.Login)))  {  errors.Append("Логин должен быть уникальным");  }  if (errors.Length > 0)  {  MessageBox.Show(errors.ToString());  return;  }  NewUser.Logins.Password = Validater.HashPassword(NewUser.Logins.Password);  if (NewUser.User\_id == 0)  BaseModel.Users.Create(NewUser);  try  {  BaseModel.Save();  MessageBox.Show("успешно");  GoBack();  }  catch (Exception ex)  {  MessageBox.Show(ex.ToString());  }  } |

Листинг класса FileService

|  |
| --- |
| internal class FileService  {  public static void DeleteFolder(string folderName)  {  string relativePath = Path.Combine("Resources/Data/Files", folderName);  string fullPath = Path.Combine(AppDomain.CurrentDomain.BaseDirectory, relativePath);  if (Directory.Exists(fullPath))  {  Directory.Delete(fullPath, true);  }  }  public static void DownloadFileToFolder(string folderName, string fileName, string path)  {  try  {  string sourcePath = Path.Combine(AppDomain.CurrentDomain.BaseDirectory, "Resources", "Data", "Files", folderName, fileName);  if (!File.Exists(sourcePath))  {  MessageBox.Show("Исходный файл не найден.");  return;  }  string destinationPath = Path.Combine(path, fileName);  File.Copy(sourcePath, destinationPath, true);  MessageBox.Show($"Файл успешно скачан");  }  catch (Exception ex)  {  MessageBox.Show($"Произошла ошибка при копировании файла: {ex.Message}");  }  }  public static void CopyFileToFolder(string sourceFilePath, string folderName)  {  try  {  if (!File.Exists(sourceFilePath))  {  return;  }  string destinationPath = Path.Combine(AppDomain.CurrentDomain.BaseDirectory, "Resources", "Data", "Files", folderName);  if (!Directory.Exists(destinationPath))  {  Directory.CreateDirectory(destinationPath);  }  string fileName = Path.GetFileName(sourceFilePath);  string destinationFilePath = Path.Combine(destinationPath, fileName);  File.Copy(sourceFilePath, destinationFilePath, true);  }  catch (Exception ex)  {  MessageBox.Show($"Произошла ошибка при копировании файла: {ex.Message}");  }  }  } |

Листинг метода Save()

|  |
| --- |
| private void Save()  {  StringBuilder errors = Validater.Validate(newPost);  \_post.Date = DateTime.Now;  string filePath = \_post.Files.File\_url;  string fileName = Path.GetFileName(Icon);  \_post.Files.File\_url = fileName;  if (SelectedSubscriptionTypes != null)  \_post.access\_level = SelectedSubscriptionTypes.Level;  if (Validater.IsValidImageFormat(\_post.Files.File\_url))  \_post.Files.type = "img";  else  \_post.Files.type = "file";  if (errors.Length > 0)  {  MessageBox.Show(errors.ToString());  return;  }  if (newPost.id == 0)  BaseModel.Posts.Create(newPost);  try  {  BaseModel.Save();  FileService.CopyFileToFolder(filePath, \_post.File\_id.ToString());  App.NavigationService.NavigateWithoutHistory(FrameNames.MainFrame, new AuthorPage(\_page));  }  catch (DbEntityValidationException ex)  {  StringBuilder error = new StringBuilder();  foreach (var eve in ex.EntityValidationErrors)  {  error.AppendLine($"Entity of type \"{eve.Entry.Entity.GetType().Name}\" in state \"{eve.Entry.State}\" has the following validation errors:");  foreach (var ve in eve.ValidationErrors)  {  error.AppendLine($"- Property: \"{ve.PropertyName}\", Error: \"{ve.ErrorMessage}\"");  }  }  MessageBox.Show(error.ToString());  }  catch (Exception ex)  {  MessageBox.Show(ex.ToString());  }  } |

Листинг метода DeletePost()

|  |
| --- |
| private void DeletePost()  {  BaseModel.Posts.Delete(id);  try  {  BaseModel.Save();  App.NavigationService.NavigateWithoutHistory(FrameNames.MainFrame, new AuthorPage(BaseModel.AuthorPages.Get(Page\_id)));  }  catch (DbEntityValidationException ex)  {  StringBuilder error = new StringBuilder();  foreach (var eve in ex.EntityValidationErrors)  {  error.AppendLine($"Entity of type \"{eve.Entry.Entity.GetType().Name}\" in state \"{eve.Entry.State}\" has the following validation errors:");  foreach (var ve in eve.ValidationErrors)  {  error.AppendLine($"- Property: \"{ve.PropertyName}\", Error: \"{ve.ErrorMessage}\"");  }  }  MessageBox.Show(error.ToString());  }  catch (Exception ex)  {  MessageBox.Show(ex.ToString());  }  } |

Листинг класса SubscriptionService

|  |
| --- |
| internal class SubscriptionService  {  public static void AcceptRequest(Requests request)  {  if (GetSubscriptionByType(request.User\_id, request.Type\_id) != null)  return;  Subscriptions subscriptions = new Subscriptions();  subscriptions.Type\_id = request.Type\_id;  subscriptions.User\_id = request.User\_id;  BaseModel.Subscriptions.Create(subscriptions);  BaseModel.Request.Delete(request.id);  try  {  BaseModel.Save();  }  catch (DbEntityValidationException ex)  {  StringBuilder error = new StringBuilder();  foreach (var eve in ex.EntityValidationErrors)  {  error.AppendLine($"Entity of type \"{eve.Entry.Entity.GetType().Name}\" in state \"{eve.Entry.State}\" has the following validation errors:");  foreach (var ve in eve.ValidationErrors)  {  error.AppendLine($"- Property: \"{ve.PropertyName}\", Error: \"{ve.ErrorMessage}\"");  }  }  MessageBox.Show(error.ToString());  }  catch (Exception ex)  {  MessageBox.Show(ex.ToString());  }  }  public static Subscriptions GetMatchingSubscription(int userId, int pageId)  {  // Получаем тип подписки, связанный со страницей  var subscriptionType = BaseModel.Context.AuthorPages  .Where(ap => ap.AuthorPage\_id == pageId)  .SelectMany(ap => ap.Subscription\_type)  .Select(st => st.id)  .Distinct()  .ToList();  if (subscriptionType == null)  {  return null;  }  Subscriptions subscription = BaseModel.Context.Subscriptions  .Where(s => s.User\_id == userId && subscriptionType.Contains(s.Type\_id))  .OrderByDescending(s => s.Subscription\_type.Level)  .FirstOrDefault();  return subscription;  }  public static Requests GetRequestByType(int userId, int typeId)  {  var request = BaseModel.Context.Requests  .FirstOrDefault(r => r.User\_id == userId && r.Type\_id == typeId);  return request;  }  public static Subscriptions GetSubscriptionByType(int userId, int typeId)  {  var request = BaseModel.Context.Subscriptions  .FirstOrDefault(r => r.User\_id == userId && r.Type\_id == typeId);  return request;  }  public static List<Subscriptions> GetSubscriptionsByPage(int pageId)  {  var subscriptionTypeIds = BaseModel.Context.AuthorPages  .Where(ap => ap.AuthorPage\_id == pageId)  .SelectMany(ap => ap.Subscription\_type)  .Select(st => st.id)  .Distinct()  .ToList();  if (!subscriptionTypeIds.Any())  {  return new List<Subscriptions>();  }  // Получаем все подписки, которые соответствуют одному из ID типов подписок  var subscriptions = BaseModel.Context.Subscriptions  .Where(s => subscriptionTypeIds.Contains(s.Type\_id))  .ToList();  return subscriptions;  }  public static List<Requests> GetRequestsByPage(int pageId)  {  var subscriptionTypeIds = BaseModel.Context.AuthorPages  .Where(ap => ap.AuthorPage\_id == pageId)  .SelectMany(ap => ap.Subscription\_type)  .Select(st => st.id)  .Distinct()  .ToList();  if (!subscriptionTypeIds.Any())  {  return new List<Requests>();  }  var subscriptions = BaseModel.Context.Requests  .Where(s => subscriptionTypeIds.Contains(s.Type\_id))  .ToList();  return subscriptions;  }  public static StringBuilder CreateRequst(int userId, int typeId)  {  StringBuilder error = new StringBuilder();  if (GetRequestByType(userId, typeId) != null)  return error.Append("Запрос уже был отправлен!");  if (GetSubscriptionByType(userId, typeId) != null)  return error.Append("Вы уже подписаны!");  Requests requests = new Requests();  requests.User\_id = userId;  requests.Type\_id = typeId;  BaseModel.Request.Create(requests);  try  {  BaseModel.Save();  }  catch (DbEntityValidationException ex)  {  foreach (var eve in ex.EntityValidationErrors)  {  error.AppendLine($"Entity of type \"{eve.Entry.Entity.GetType().Name}\" in state \"{eve.Entry.State}\" has the following validation errors:");  foreach (var ve in eve.ValidationErrors)  {  error.AppendLine($"- Property: \"{ve.PropertyName}\", Error: \"{ve.ErrorMessage}\"");  }  }  return error;  }  catch (Exception ex)  {  error.AppendLine(ex.ToString());  return error;  }  return error;  }  } |

Листинг метода Save() из CreateSubscriptionTypesVM

|  |
| --- |
| private void Save()  {  StringBuilder errors = Validater.Validate(\_subscription\_Type);    if (errors.Length > 0)  {  MessageBox.Show(errors.ToString());  return;  }  if (\_subscription\_Type.id == 0)  BaseModel.SubscriptionType.Create(\_subscription\_Type);  try  {  BaseModel.Save();  App.NavigationService.NavigateWithoutHistory(FrameNames.MainFrame, new AuthorPageSetting(\_page));  }  catch (Exception ex)  {  MessageBox.Show(ex.ToString());  }  } |

Листинг метода Reject ()

|  |
| --- |
| private void Reject()  {  BaseModel.Request.Delete(id);  try  {  BaseModel.Save();  }  catch (DbEntityValidationException ex)  {  StringBuilder error = new StringBuilder();  foreach (var eve in ex.EntityValidationErrors)  {  error.AppendLine($"Entity of type \"{eve.Entry.Entity.GetType().Name}\" in state \"{eve.Entry.State}\" has the following validation errors:");  foreach (var ve in eve.ValidationErrors)  {  error.AppendLine($"- Property: \"{ve.PropertyName}\", Error: \"{ve.ErrorMessage}\"");  }  }  MessageBox.Show(error.ToString());  }  catch (Exception ex)  {  MessageBox.Show(ex.ToString());  }  RequestRejected?.Invoke(this, EventArgs.Empty);  } |

Листинг метода Accept ()

|  |
| --- |
| private void Accept()  {  SubscriptionService.AcceptRequest(this);  RequestRejected?.Invoke(this, EventArgs.Empty);  RequesAccepted?.Invoke(this, EventArgs.Empty);  } |

Листинг метода Send()

|  |
| --- |
| private void Send()  {  Comments comments = new Comments();  if (Text.IsNullOrEmpty())  return;  comments.Post\_id = \_post.id;  comments.Text = Text;  comments.User\_id = Store.User.User\_id;  comments.Date = DateTime.Now;  BaseModel.Comments.Create(comments);  try  {  BaseModel.Save();  Comments.Add(comments);  comments.CommentDeleted += CommentDeleted;  Text = "";  }  catch (DbEntityValidationException ex)  {  StringBuilder error = new StringBuilder();  foreach (var eve in ex.EntityValidationErrors)  {  error.AppendLine($"Entity of type \"{eve.Entry.Entity.GetType().Name}\" in state \"{eve.Entry.State}\" has the following validation errors:");  foreach (var ve in eve.ValidationErrors)  {  error.AppendLine($"- Property: \"{ve.PropertyName}\", Error: \"{ve.ErrorMessage}\"");  }  }  MessageBox.Show(error.ToString());  }  catch (Exception ex)  {  MessageBox.Show(ex.ToString());  }  } |