МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«МАГНИТОГОРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ

УНИВЕРСИТЕТ ИМ. Г. И. НОСОВА»

КАФЕДРА ИНФОРМАТИКИ И ИНФОРМАЦИОННОЙ БЕЗОПАСНОСТИ

**КУРСОВАЯ РАБОТА**

по дисциплине: Языки программирования

на тему: Разработка Windows-приложения мессенджера

Исполнитель: Осяев Д. В. студент 2 курса, группа АИБ-23-2  
 (Ф.И.О.)

Руководитель: Кузьмина У. В., к.т.н., доцент   
 (Ф.И.О. должность, уч. степень, уч. звание)

Работа допущена к защите "\_\_\_" \_\_\_\_\_\_\_\_\_ 2025г. \_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись)

Работа защищена "\_\_\_" \_\_\_\_\_\_\_\_\_ 2025г. с оценкой \_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_

(оценка) (подпись)

Магнитогорск, 2025

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«МАГНИТОГОРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ

УНИВЕРСИТЕТ ИМ. Г. И. НОСОВА»

КАФЕДРА ИНФОРМАТИКИ И ИНФОРМАЦИОННОЙ БЕЗОПАСНОСТИ

**ЗАДАНИЕ НА КУРСОВУЮ РАБОТУ**

Тема: Разработка Windows-приложения мессенджера

Студенту: Осяеву Данилу Викторовичу

(фамилия имя отчество)

Исходные данные: Лекционные и справочные материалы, электронные справочные материалы по Visual Studio 2022, интернет-ресурсы

Срок сдачи: «\_\_\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2025 г

Руководитель: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ / Кузьмина У.В.   
 (подпись) (расшифровка подписи)

Задание получил: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ / Осяев Д.В.   
 (подпись) (расшифровка подписи)

Магнитогорск, 2025

**Содержание**

[Введение 4](#_Toc188703856)

[1. Теоретическая часть 6](#_Toc188703857)

[1.1 Описание объектно-ориентированного программирования (ООП) 6](#_Toc188703858)

[1.2 Описание клиент-серверной архитектуры 7](#_Toc188703859)

[1.3 Описание Windows Communication Foundation 10](#_Toc188703860)

[1.4 Описание Windows Presentation Foundation 12](#_Toc188703861)

[1.5 Двухфакторная аутентификация 14](#_Toc188703862)

[1.6 Сокеты в клиент-серверной архитектуре 15](#_Toc188703863)

[2. Практическая часть 17](#_Toc188703864)

[2.1 Разработка приложения 17](#_Toc188703865)

[2.2 Структура серверной части приложения 18](#_Toc188703866)

[2.3 Структура приложения-хостинга 25](#_Toc188703867)

[2.4 Структура клиентской части приложения 27](#_Toc188703868)

[2.5 Интерфейс клиентской части приложения 35](#_Toc188703869)

[2.6 Серверная часть приложения 43](#_Toc188703870)

[Заключение 45](#_Toc188703871)

[Список литературы 46](#_Toc188703872)

[Приложение А IServiceChat.cs 47](#_Toc188703873)

[Приложение Б Листинг ServiceChat.cs 50](#_Toc188703874)

[Приложение В Листинг ContactList 60](#_Toc188703875)

[Приложение Г Листинг Contacts.xaml.cs 73](#_Toc188703876)

# Введение

В современном мире коммуникация занимает важное место в повседневной жизни людей и организаций. С распространением мобильных устройств и высокоскоростного интернета мессенджеры стали неотъемлемой частью общения. Они позволяют мгновенно обмениваться текстовыми сообщениями, мультимедийными файлами и проводить видеозвонки. По данным статистики, число пользователей мессенджеров стремительно растет, и эта тенденция продолжает усиливаться

Актуальность темы данной работы обусловлена необходимостью создания эффективных и безопасных средств общения, которые удовлетворяли бы потребности пользователей в удобстве, скорости передачи данных и защите личной информации. Мессенджеры, как одни из самых популярных форм электронной коммуникации, вызывают интерес с технической, социальной и экономической точек зрения.

Цель данной курсовой работы заключается в разработке мессенджера, который будет сочетать в себе простоту использования, высокую скорость работы и надежные механизмы защиты данных. Приложение разрабатывается с применением технологий Windows Presentation Foundation (WPF) и Windows Communication Foundation (WCF). WPF представляет собой мощную платформу для создания богатых пользовательских интерфейсов, обеспечивая высокую степень гибкости и адаптивности дизайна. WCF, в свою очередь, является фреймворком для построения сервисов, который позволяет легко создавать надежные и безопасные коммуникации между различными приложениями.

Приложение разрабатывается на клиент-серверной архитектуре, где WCF-сервер будет выполнять функции центрального сервиса, обрабатывающего запросы от клиентских приложений. К сервису WCF-клиента могут подключиться множество пользователей, не имея при этом прямой связи друг с другом, но имея возможность обмениваться сообщениями по средствам работы с общей базой данных. Этот подход позволяет разделить логику обработки данных и пользовательский интерфейс, повышая масштабируемость и удобство поддержки.

В разрабатываемом мессенджере все действия производятся с использованием методов классов, что обеспечивает четкую структуру приложения и легкость в поддержке кода. Основной функционал при разработке клиентской части, реализован в виде вызовов методов сервера, логика которых полностью описана в серверной части приложения, и защищена от пользователя. Такой подход позволяет обезопасить серверное приложение, жестко ограничив возможности приложения клиента. Клиент может взаимодействовать с серверной частью приложения для исполнения различных задач, таких как отправка сообщений, работа с персональными закладками, работа с личным профилем пользователя, работа с контактами пользователя.

Основные задачи, поставленные в рамках работы, включают анализ современных мессенджеров, проектирование архитектуры приложения, реализацию основного функционала и оценку производительности разработанного продукта.

Объектом исследования в данной курсовой работе является программное обеспечение мессенджера, а предметом — его функциональные возможности и архитектурные решения. При выполнении работы применяются методы анализа, проектирования и разработки программного обеспечения, а также тестирования созданного приложения.

# Теоретическая часть

## 1.1 Описание объектно-ориентированного программирования (ООП)

Объектно-ориентированное программирование (ООП) является одним из самых популярных подходов к разработке программного обеспечения. Этот метод программирования основывается на использовании объектов, которые представляют собой объединение данных и методов, действующих на эти данные. Главная цель ООП — сделать процесс разработки программ более структурированным и удобным, а также улучшить поддержку, расширяемость и повторное использование кода.

Объект в ООП — это класс, который является своего рода шаблоном, определяющим свойства (атрибуты) и поведение (методы), присущие объектам данного типа. Каждое свойство может иметь свое значение, которое задается для конкретного объекта. Методы представляют собой функции, которые могут выполнять действия или вычисления, оперируя данными объекта.

Классы могут наследоваться друг от друга, что позволяет создавать более специфичные классы на основе общего. Это значительно упрощает процесс разработки, так как дает возможность повторно использовать уже существующий код и расширять его.

В разрабатываемом мессенджере все действия производятся при использовании методов тех или иных классов, что способствует структурированию кода и улучшает его читаемость. Основной функционал приложения представлен клиенту в виде методов сервера, что обеспечивает взаимодействие между клиентскими приложениями и серверной частью. Такой подход позволяет разделить логику обработки данных и пользовательский интерфейс, обеспечивая гибкость и простоту в дальнейшем обслуживании и разработке.

Существует четыре основных принципа ООП:

1. Инкапсуляция: Этот принцип подразумевает скрытие внутреннего состояния объекта и предоставление доступа к нему только через публичные методы. Инкапсуляция позволяет защитить данные объекта от непреднамеренных изменений и упрощает поддержку кода.

2. Наследование: Наследование позволяет создать новый класс, основанный на существующем. Это обеспечивает повторное использование кода и позволяет создать иерархию классов. Например, если существует класс «Животное», можно создать класс «Собака», который будет наследовать характеристики «Животного», добавляя при этом особенности, специфичные для собак.

3. Полиморфизм: Полиморфизм позволяет методам иметь одно имя, но разное поведение в зависимости от контекста, в котором они используются. Например, разные классы могут реализовывать метод с одинаковым именем по-разному, что позволяет использовать единый интерфейс для работы с различными объектами.

4. Абстракция: Этот принцип подразумевает выделение ключевых характеристик объекта и создание упрощенной модели, которая акцентирует внимание на необходимых аспектах. Абстракция позволяет сосредоточиться на том, как использовать объект, не вдаваясь в детали его реализации.

Одним из главных преимуществ объектно-ориентированного подхода является возможность создания масштабируемых и легко управляемых приложений. Благодаря инкапсуляции, наличие четких границ между различными частями программы способствует минимизации ошибок. Наследование позволяет быстро адаптироваться к изменениям требований, а полиморфизм делает систему более гибкой, позволяя работать с различными типами объектов через единый интерфейс.

Объектно-ориентированное программирование стало основой для многих современных языков программирования, таких как Java, C#, Python и других. Эти языки предоставляют богатый набор средств для реализации концепций ООП, что делает разработку программ более эффективной и организованной.

В рамках разработки мессенджера, применение принципов ООП позволяет создать масштабируемую архитектуру, упростить процесс добавления новых функций и улучшить поддержку кода, что, в свою очередь, позитивно сказывается на качестве и надежности конечного продукта.

1.2 Описание клиент-серверной архитектуры

Клиент-серверная архитектура является одним из наиболее популярных подходов к разработке современных приложений, особенно в контексте реализации многофункциональных систем, таких как мессенджеры. Этот подход предполагает разделение приложения на две основные части: клиентскую и серверную. Клиентская часть отвечает за взаимодействие с пользователем, тогда как серверная обрабатывает запросы, управляет данными и выполняет логику приложения.

В клиент-серверной архитектуре клиенты отправляют запросы на сервер, который в свою очередь обрабатывает эти запросы и возвращает соответствующие ответы. Это позволяет распределять нагрузку на систему и обеспечивает возможность централизованного управления данными. Такое разделение также улучшает безопасность, так как сервер может контролировать доступ к данным и логике приложения, а клиенты остаются легковесными и ориентированными на интерфейс пользователя.

Система клиент-серверного приложения разделена на два уровня:

Уровень клиента: Этот уровень отвечает за пользовательский интерфейс и взаимодействие с пользователем. Он реализован с использованием Windows Presentation Foundation (WPF), которая позволяет создавать богатые и адаптивные пользовательские интерфейсы с поддержкой современных графических технологий.

Уровень сервера: Этот уровень обрабатывает логику и обеспечивает доступ к данным. Он реализован с использованием Windows Communication Foundation (WCF), которая предоставляет возможности для построения распределенных приложений и реализации различных протоколов передачи данных.

В разрабатываемом мессенджере клиентская часть реализована с использованием WPF, что позволит создать современный и удобный интерфейс для пользователей. Серверная часть будет основана на WCF, что обеспечит надежное и масштабируемое взаимодействие между клиентами и центром обработки данных.

Дополнительно, использование клиент-серверной архитектуры в сочетании с WCF и WPF позволяет:

- Легко настраивать и изменять способы обмена данными между клиентами и сервером.

- Обеспечить высокую производительность и надежность приложения.

- Создать платформу, которая может адаптироваться под будущие требования пользователей и технологии.

Таким образом, применение клиент-серверной архитектуры на базе WCF и WPF служит основой для создания эффективного и функционального мессенджера, способного удовлетворить потребности современных пользователей в быстром и безопасном обмене информацией.

Существует несколько архитектурных подходов, которые можно рассмотреть как альтернативы клиент-серверной архитектуре. Основные из них включают:

1. Одноуровневая архитектура (Монолитная архитектура): В этой архитектуре все компоненты приложения (пользовательский интерфейс, бизнес-логика, доступ к данным) находятся в одном исполняемом пакете. Такой подход проще в реализации и деплое, однако он не масштабируем и сложно поддерживаем. Изменения в одном из компонентов могут повлиять на всю систему. Данная архитектура не подходит под тему проекта, так как мессенджер должен быть связан с общей базой данных для обработки данных.

2. N-Tier архитектура: Этот подход предполагает разделение приложения на несколько логических уровней (например, уровень представления, уровень бизнес-логики, уровень доступа к данным). Хотя такая архитектура предлагает большую гибкость и возможность распределения нагрузки, она может быть сложнее в реализации и требует значительных усилий для координации между различными уровнями. Данная архитектура подходит под тему проекта, но она не была выбрана из-за сложности реализации и недостатка времени на разработку. Выделение отдельного сервера для работы с базой данных является перспективным вариантом развития проекта мессенджера.

3. Микро сервисная архитектура: В этой модели приложение разбивается на множество мелких, независимых сервисов, каждый из которых выполняет определенные функции и может взаимодействовать с другими сервисами через API. Хотя микросервисы обеспечивают высокую степень гибкости и масштабируемости, их внедрение требует сложной инфраструктуры и контроля. Данная архитектура подходит под тему проекта, но она не была выбрана из-за сложности реализации и отсутствии необходимости разбиения приложения на микросервисы.

## 1.3 Описание Windows Communication Foundation

При разработке серверной части приложения, использовалась технология WCF.

Windows Communication Foundation (WCF) — это платформа разработки и выполнения распределенных приложений от компании Microsoft, являющаяся частью .NET Framework. WCF предоставляет разработчикам инструментарий для создания сервис-ориентированных приложений, обеспечивающих взаимодействие между компонентами и системами независимо от их платформ и технологий.

Основные особенности WCF включают поддержку различных протоколов связи (HTTP, TCP, Named Pipes, MSMQ и др.), возможность использования различных форматов данных (например, XML, JSON), а также механизм безопасности и управления транзакциями.

Основные компоненты WCF:

**1. Сервис (Service)** — это компонент, предоставляющий функциональность для других приложений. Он состоит из контрактов, которые описывают интерфейсы и структуры данных, используемые для взаимодействия.

**2. Клиент (Client)** — это компонент, который обращается к сервису для выполнения операций. Клиент может быть любым приложением или устройством, способным отправлять запросы на сервер WCF.

**3. Адрес (Address)** — определяет местоположение сервиса. Адрес задается с использованием универсального идентификатора ресурса (URI) и протокола, например, http: или net.tcp:.

**4. Привязка (Binding)** — описывает способ связи между клиентом и сервисом. Привязки определяют транспортный протокол, формат сообщений, а также настройки безопасности.

**5. Контракт (Contract)** — описывает, что именно сервис может выполнять. Преимущества использования WCF.

Преимущества WCF:

**1. Поддержка различных протоколов связи**. WCF позволяет работать с несколькими протоколами в рамках одного приложения.

**2. Унификация разработки**. Все виды взаимодействия (HTTP, TCP, IPC и др.) реализуются с использованием единого подхода.

**3. Безопасность**. WCF включает встроенные механизмы аутентификации, авторизации и шифрования данных.

**4. Расширяемость**. WCF позволяет добавлять пользовательские поведения и расширения для настройки работы сервиса.

**5. Интеграция с другими технологиями**. WCF поддерживает взаимодействие с другими стандартами и платформами, такими как SOAP, REST и WS-\* стандарты.

Альтернативы WCF в .NET:

**1. ASP.NET Core gRPC**: gRPC — это современный фреймворк для высокопроизводительных RPC (удалённых вызовов процедур), который поддерживается в ASP.NET Core. Он отличается высокой скоростью работы благодаря использованию протокола HTTP/2 и бинарному формату данных Protocol Buffers.

**2. ASP.NET Web API**:Используется для создания RESTful API, работающих поверх протокола HTTP. Это более лёгкий и простой подход по сравнению с WCF, идеально подходящий для интеграции с веб-клиентами и мобильными приложениями.

**3. SignalR**: Фреймворк для реализации двусторонней связи между клиентом и сервером в реальном времени. Подходит для приложений, таких как чаты, уведомления и онлайн-игры.

**4. MassTransit и NServiceBus**: Это библиотеки для реализации обмена сообщениями между компонентами распределённых систем. Они хорошо подходят для создания систем на основе сообщений и предоставляют дополнительные функции, такие как ретрансляция сообщений, управление очередями и маршрутизацией.

**5. Raw Sockets**: Если требуется максимальная производительность и полный контроль над соединением, можно использовать System.Net.Sockets для реализации низкоуровневого сетевого взаимодействия.

Для разработки серверной части мессенджера была выбрана технология WCF из-за относительной простоты использования и наличия большого количества учебных материалов по WCF.

## 1.4 Описание Windows Presentation Foundation

Windows Presentation Foundation (WPF) — это платформа для создания пользовательских интерфейсов (UI) в приложениях Windows. Она была разработана Microsoft и является частью .NET Framework. WPF предоставляет мощные инструменты для разработки гибких и визуально богатых приложений, используя современные подходы к созданию графики и взаимодействия.

Основные особенности WPF:

**1. Поддержка декларативного подхода**. Интерфейс в WPF описывается с помощью языка XAML (eXtensible Application Markup Language), что позволяет отделить логику интерфейса от логики приложения.

**2. Гибкость визуализации**. WPF предоставляет возможности для создания как простых форм, так и сложных визуальных эффектов, включая 2D- и 3D-графику, анимацию и стилизацию.

**3. Привязка данных (Data Binding)**. WPF обеспечивает мощные механизмы для привязки данных, упрощая разработку приложений с динамически обновляемыми интерфейсами.

**4. Шаблоны и стили**. WPF позволяет легко изменять внешний вид элементов управления с использованием шаблонов и стилей, что обеспечивает высокую степень кастомизации.

**5. Поддержка мультимедиа**. Платформа поддерживает воспроизведение видео, аудио и других мультимедийных элементов прямо в приложении.

**6. Реакция на изменения размеров**. Интерфейсы, созданные с помощью WPF, автоматически адаптируются к изменениям размеров окон и экранов.

Преимущества использования WPF

**1. Мощные графические возможности**. Поддержка 2D, 3D, анимации и мультимедиа делает WPF подходящим для создания визуально насыщенных приложений.

**2. Единообразие разработки**. Отделение логики интерфейса и бизнес-логики упрощает поддержку и развитие приложений.

**3. Поддержка современных технологий**. Возможности интеграции с другими компонентами .NET, такими как LINQ, Entity Framework и др.

**4. Кросс-платформенность внутри Windows**. Приложения WPF работают на всех поддерживаемых версиях Windows.

Аналоги WPF в .NET:

1. **WinForms (Windows Forms):** Более старая технология для создания пользовательских интерфейсов, основанная на GDI+. Она легче в освоении и использовании по сравнению с WPF, но имеет ограниченные возможности для создания сложных графических интерфейсов и анимаций.

2. **MAUI (Multi-platform App UI)**: Современная кросс-платформенная альтернатива WPF, которая позволяет создавать приложения для Windows, macOS, Android и iOS с использованием общего кода и интерфейсов на XAML.

3. **Uno Platform**: Фреймворк для создания кросс-платформенных приложений, основанный на Universal Windows Platform (UWP). Поддерживает создание приложений для Windows, macOS, Android, iOS и WebAssembly.

4. **Blazor**: Технология для создания веб-интерфейсов с использованием C# и .NET. Подходит для создания интерактивных пользовательских интерфейсов, которые могут работать в браузере с помощью WebAssembly.

5. **Avalonia UI**: Кросс-платформенный фреймворк для создания приложений, схожий по функциональности и синтаксису с WPF. Поддерживает Windows, macOS, Linux и другие платформы.

6. **UWP (Universal Windows Platform)**: Платформа для создания универсальных приложений Windows, которые могут работать на устройствах с Windows 10 и выше, включая ПК, планшеты, Xbox и HoloLens.

7. **Qt for .NET**: Кросс-платформенный инструмент для разработки интерфейсов, который поддерживает C# через библиотеки-обёртки, такие как Qml.Net. Подходит для создания приложений с богатым UI на разных платформах.

WPF использовался при разработке учебных программ. Выбор WCF обусловлен удобством технологии и современным дизайном интерфейса.

## Двухфакторная аутентификация

Одно из требований при разработке приложения-мессенджера для данной курсовой работы – это реализация двухфакторной аутентификации при регистрации и авторизации. Двухфакторная аутентификация должна быть реализована через отправку сообщения с кодом на почту и проверку введенного пользователем кода на подлинность.

Двухфакторная аутентификация – это метод усиленной безопасности, который требует от пользователя предоставления двух различных факторов для подтверждения своей личности перед получением доступа к учетной записи или системе.

При использовании двухфакторной аутентификации, помимо ввода пароля, пользователю необходимо предоставить дополнительную информацию, как правило, код, отправленный на его мобильное устройство или электронную почту.

Двухфакторная аутентификация существенно повышает уровень безопасности учетных записей и систем. Основные причины ее важности включают:

- Увеличение защиты от взломов: Даже если злоумышленник получает доступ к вашему паролю, он не сможет войти без второго фактора аутентификации.

- Минимизация рисков утечек данных: Защита важных данных и информации в финансовых или медицинских системах становится более надежной.

- Защита от фишинга: Двухфакторная аутентификация затрудняет доступ к учетной записи, даже если пароли были скомпрометированы.

Также существует несколько схожих терминов, имеющих непосредственное отношение к данному проекту, а именно: авторизация и идентификация.

- Идентификация — это процесс подтверждения того, кто вы есть. Например, ввод вашего имени пользователя или адреса электронной почты.

- Авторизация — это процесс проверки прав и разрешений пользователя после его идентификации. Например, доступ к определенным ресурсам в зависимости от ваших прав в системе.

- Двухфакторная аутентификация сочетает оба эти аспекта, добавляя дополнительный уровень проверки после идентификации, что делает доступ к данным и системам более защищенным.

Двухфакторная аутентификация через почту считается менее надежной по сравнению с другими методами (такими как SMS или приложения для аутентификации), по причинам:

1. Уязвимость почты: Если злоумышленник сможет получить доступ к вашей электронной почте (например, через фишинговую атаку), он может перехватить коды, отправленные на почту.

2. Задержки и проблемы с доставкой: Коды могут приходить с задержкой, или электронное письмо может не прийти вовсе, что может привести к неудобствам.

3. Отсутствие дополнительного шифрования: Почта может не быть защищена так же надежно, как SMS сообщения или специальные приложения для аутентификации, которые используют временные коды и обеспечивают более высокий уровень защиты.

Тем не менее, уровень безопасности, предоставляемый двухфакторной аутентификацией, подходит под требования разработки приложения, так же выделяясь большим удобством реализации.

## 1.6 Сокеты в клиент-серверной архитектуре

WebSocket - это протокол двунаправленной связи между веб-приложением и сервером в режиме реального времени.

Сокеты – это один из ключевых компонентов в клиент-серверной архитектуре, который позволяет обмениваться данными между клиентом и сервером через сети, такие как Интернет, или локальные сети. Сокеты предоставляют программный интерфейс для реализации сетевых соединений и позволяют передавать данные в виде сообщений или потоков. Сокет представляет конечную точку соединения в сети. Он используется для передачи и получения данных между приложениями, работающими на разных устройствах.

- Сокеты могут быть ориентированы на соединение (TCP) или без соединения (UDP).

- В клиент-серверной архитектуре клиент инициирует соединение и отправляет запросы, в то время как сервер ожидает входящих соединений, обрабатывает запросы и отправляет ответы.

- Клиент обычно работает на одном хосте, а сервер — на другом, но это не обязательно.

Типы сокетов:

 - TCP-сокеты: обрабатывают соединения, основанные на протоколе TCP (Transmission Control Protocol). Обеспечивают надежную, ориентированную на соединение передачу данных. Гарантируют доставку пакетов, порядок и целостность данных.

- UDP-сокеты: используют протокол UDP (User Datagram Protocol). Не обеспечивают надежность и порядок доставки, что делает их менее грузными и более быстрыми, но также и более рискованными в плане потери данных.

# Практическая часть

## 2.1 Разработка приложения

Приложение мессенджер реализовано на клиент-серверной архитектуре. Клиентская часть представляет собой WPF приложение, серверная часть WCF библиотека классов. Отдельно запускается приложение хост, которое реализует логику работы сервера, используя библиотеку классов WPC. Для удобства, приложение клиент и приложение хост с библиотекой классов WCF, собраны в одном решении Visual Studio.

Взаимодействие клиента и сервера осуществляется по средствам обращения клиентской части к серверу, через класс ServiceChat, листинг которого представлен в приложении Б и листинге 2. Сервер предоставляет клиенту свои методы – интерфейсы, через которые пользователь взаимодействует с базой данных мессенджера – добавляя контакты, или отправляя сообщения.

Параметры сетевого подключения сервера и клиента, описываются в файлах App.config клиента и сервера, там указывается сетевой путь через интернет, или через локальную сеть.

Структура проекта представлена на рисунке 1.

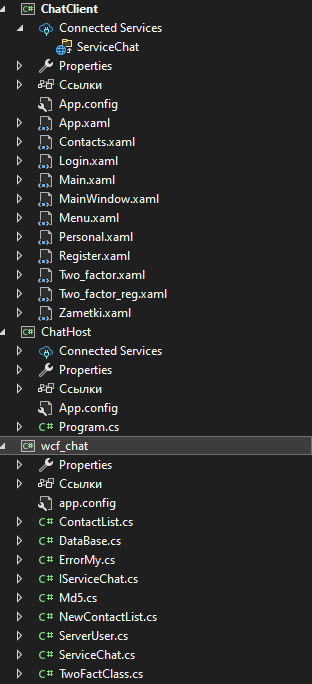


Рисунок 1 – Структура приложения

## 2.2 Структура серверной части приложения

Структура серверной части приложения представлена на рисунке 2. Основные классы, необходимые для работы сервера представлены ниже.

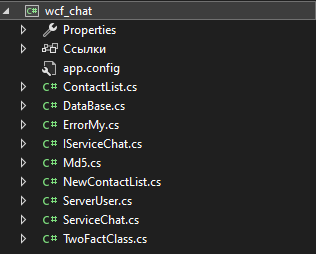


Рисунок 2 – Структура серверной части приложения

Класс IServiceChat является родительским классом для класса ServiceChat. Данный класс определяет методы взаимодействия внутри сервера и интерфейсы, доступные пользователю. Методы, являющиеся интерфейсами, обозначаются атрибутом [OperationContract]. Часть кода класса IServiceChat.cs представлена в листинге 1. Полный код представлен в приложении А.

Клиент создает экземпляр класса сервера, через этот экземпляр класс, клиент обращается к его методам, именуемым интерфейсами. К примеру, в листинге 1 класса IServiceChat.cs, есть интерфейс GetUserData, к нему клиент может обратиться и получить доступ к функционалу сервера, а конкретно, получить актуальные данные для профиля пользователя.

Листинг 1 – Код класса IServiceChat.cs

1. [ServiceContract(CallbackContract = typeof(IServerChatCallback))] // базовая настройка

2. public interface IServiceChat { // Метод для создания новой строки с данными о

3. //пользователе в таблице с данными о пользователях.

Продолжение листинга 1.

4. [OperationContract] string NewUserData(string name, string last\_name, string status);

5. // Метод для получения данных о пользователе

6. [OperationContract] (string, string, string) GetUserData();

7. // Метод для добавления нового сообщения в таблицу чата

8. [OperationContract] string NewMessChat(string log, string message);

9. // Метод для получения списка контактов

10. [OperationContract] (List<string>, List<string>) GetContList();

11. // Метод для добавления нового контакта

12. [OperationContract] string NewCont(string log);

Класс ServiceChat наследует методы класса IServiceChat и описывает их функционал. Данный класс является основным на сервере, и в нем написана основная логика работы приложения. Часть кода класса ServiceChat с описанием одного метода, представлена в листинге 2. Полный код класса представлен в приложении Б.

Клиент создает переменную для работы с сервером с типом данных ServiceChat. Создание подобной переменной и работа с ней представлена на приложении В и листинге 11. Конкретное использование методов сервера – интерфейсов, подробно описывается в теме 2.4, где конкретно рассматривается код клиентской части.

В данной теме описывается непосредственное создание интерфейсов – методов класса работы с сервером.

В классе IServiceChat, интерфейсы только были объявлены, в классе же ServiceChat описан их функционал. Так, на примере листинга 2, где представлен метод для создания новой строки с информацией о пользователе, в общей таблице с списком всех активных пользователей. Таким образом, с помощью интерфейсов (методов) класса ServiceChat, пользователь и получает весь доступный ему функционал взаимодействия с сервером.

Тут мы используем заранее объявленный интерфейс NewUserData, и описываем его логику. Так, к примеру, в нем создается экземпляр класса работы с базой данных DataBase, выполняются запросы к базе данных, цель которых, в данном случае, добавить новую строку в базу.

Листинг 2 – Код класса ServiceChat.cs

1. DataBase database = new DataBase();// Создание переменной для работы с базой данных
2. private string active\_user\_login = null; // Логин активного пользователя
3. // Метод для создания новой строки с данными о пользователе в таблице
4. public string NewUserData(string name, string last\_name, string status) {
5. try {
6. // Классы, необходимые для работы с базой данных
7. SqlDataAdapter adapter = new SqlDataAdapter();
8. DataTable table = new DataTable();
9. // Строка запроса
10. string queryString = $"use kurs; update usersData set user\_first\_name = '{name}',
11. user\_last\_name = '{last\_name}', user\_status = '{status}' where user\_login =
12. '{active\_user\_login}';";
13. // Объект класса sqlCommand, хранящий созданную строку запроса
14. SqlCommand command1 = new SqlCommand(queryString,
15. database.getConnection());
16. // Работа с адаптером, выполнение запроса к базе данных
17. adapter.SelectCommand = command1;
18. adapter.Fill(table);
19. return "OK"; }// Возврат статуса выполнения метода

Класс ContactList предоставляет функционал для работы со списком контактов пользователя и чатами пользователя. Часть кода класса ContactList представлена в листинге 3. Полный код класса ContactList представлен в приложении В.

Класс ContactList используется в интерфейсах класса ServiceChat.cs, предоставляя функционал для работы со списком контактов пользователя. Клиент не имеет прямого доступа к классу ContactList, доступ к функционалу данного класса предоставляется через интерфейсы класса ServiceChat.cs.

К примеру, в классе ContactList, представлен метод создания нового сообщения в чате NewMessage, он добавляет в таблицу чата новую строку сообщения.

Методы данного класса вызываются в ServiceChat.cs через: listCont = contLists.GetContactLists().Item1; - данная строка получает список контактов пользователя и записывает его в переменную список. Обращение через .Item2, в данном случае вернуло бы черный список пользователя.

Листинг 3 – Код класса ContactList.cs

1. // Класс для работы с контактами и чатами
2. internal class ContactList { // Метод для создания нового сообщения в чате
3. public string NewMessage(string log, string actLog, string message) {
4. try { // Блок перехвата ошибок
5. List<string> listMess = new List<string>(); // Список сообщений
6. SqlDataAdapter adapter = new SqlDataAdapter(); // Адаптер для работы с бд
7. DataTable table = new DataTable(); // Переменная для хранения таблицы
8. string log1 = actLog; // Переменная для активного логина
9. string log2 = log; // Переменная для логина получателя
10. // Сравнение строк, чтобы создавалась таблица с корректным названием
11. int sravn = string.Compare(log1, log2); // Функция сравнения строк
12. if (sravn > 0) { string per = log1; log1 = log2; log2 = per; }
13. string queryString = $"use kurs; insert into {log1 + "CHAT" + log2} (mess) values
14. ('{message}');"; // Строка запроса для добавления нового сообщения в чат
15. SqlCommand commandBlack = new SqlCommand(queryString,
16. database.getConnection()); // Открытие контакта с бд
17. adapter.SelectCommand = commandBlack; // Выполнение запроса к бд
18. adapter.Fill(table); // Заполнение переменной-таблицы значениями
19. database.closeConnection(); // Закрытие контакта с бд
20. return "OK"; // Возврат статуса выполнения }

Класс NewContactList.cs предоставляет функционал для создания таблицы со списком контактов пользователя в базе данных. Код класса NewContactList.cs представлен в листинге 4.

Логика обращения к данному классу не отличается от логики обращения к ContactList.cs.

Листинг 4 – Код класса NewContactList.cs

1. // Класс для создания новых таблиц списка контактов пользователя и черного списка

Продолжение листинга 4.

1. internal class NewContactList {
2. DataBase database = new DataBase(); // Переменная для работы с бд
3. public NewContactList(string newLogin) // метод создания нового контакта {
4. // Классы, необходимые для работы с базой данных
5. SqlDataAdapter adapter = new SqlDataAdapter();
6. DataTable table = new DataTable();
7. // Строка запроса создания таблицы контактов
8. string queryString1 = $"use kurs; create table {newLogin + "Contacts"} (contact\_id int
9. identity(1,1) not null, contact\_login varchar(20) not null primary key (contact\_id));";
10. // Строка запроса создания таблицы черного списка
11. string queryString2 = $"use kurs; create table {newLogin + "BlackContacts"} (contact\_idi
12. nt identity(1,1) not null, contact\_login varchar(20) not null primary key (contact\_id));";

14. // Объекты класса sqlCommand, хранящие строки запроса

15. SqlCommand command1 = new SqlCommand(queryString1, database.getConnection());

16. SqlCommand command2 = new SqlCommand(queryString2, database.getConnection());

17. adapter.SelectCommand = command1; // Применение запроса к бд

18. adapter.Fill(table); // Заполнение переменной таблицы данными

19. adapter.SelectCommand = command2; // Применение запроса к бд

20. adapter.Fill(table); // Заполнение переменной таблицы данными

21. database.closeConnection(); // Закрытие соединения с бд }}

Класс DataBase.cs предоставляет функционал для работы с базой данных MSSQL. В нем хранится информация о подключении к базе данных. Код класса DataBase.cs представлен в листинге 5.

В данном классе, производится подключение к базе данных, по заранее указанным параметрам подключения. Так, строкой sqlConnection указываются параметры сервера базы данных, к примеру как адрес источника данных Data Source=DESKTOP R0C7M0\SQLEXPRESS;

Обращение к классу DataBase происходит практически в каждом классе сервера, так, к примеру, в ServerChat.cs, обращение происходит через создание экземпляра класса: SqlDataAdapter adapter = new SqlDataAdapter();, и обращения к его методу открытия соединения: SqlCommand command1 = new SqlCommand( queryString , database.getConnection());

Листинг 5 – Код класса DataBase.cs

1. internal class DataBase // Класс работы с базой данных {
2. // Строка подключения
3. SqlConnection sqlConnection = new SqlConnection(@"Data Source=DESKTOP
4. R0C7M0\SQLEXPRESS; Initial Catalog=sevenFirst; Integrated Security=True;
5. ncrypt=False");
6. public void openConnection() // Метод открывающий связь с базой данных {
7. // Если статус подключения закрыт, открыть
8. if (sqlConnection.State == System.Data.ConnectionState.Closed)
9. { sqlConnection.Open(); } }
10. public void closeConnection() // Метод закрывающий связь с базой данных {
11. // Если статус подключения закрыт, открыть
12. if (sqlConnection.State == System.Data.ConnectionState.Open)
13. { sqlConnection.Close(); } }
14. // Метод возвращающий строку о состоянии открытия
15. public SqlConnection getConnection()
16. { return sqlConnection; } }

Класс TwoFactClass.cs предоставляет функционал, необходимы для реализации двухфакторной аутентификации по почте при регистрации и авторизации пользователя. В классе формируется уникальный код и отправляется на указанную почту. Основная часть кода класса представлена в листинге 6.

Листинг 6 – Код класса TwoFactClass.cs

1. internal class TwoFactClass // Класс для реализации двухфакторной аутентификации {
2. // Метод для отправки сообщения на почту
3. private void SendMessage(string text, string toUser) {
4. // Установка адреса и отображаемого в письме имени
5. MailAddress from = new MailAddress("testpost7788@gmail.com", "TEST");
6. MailAddress to = new MailAddress(toUser); // Установка адреса получателя

Продолжение листинга 6.

7. MailMessage m = new MailMessage(from, to); // Создание объекта сообщения

8. m.Subject = $"Код безопасности"; // Установка темы письма

9. m.Body = $"{text}"; // Установка текста письма

10. m.IsBodyHtml = true; // Параметр, xnj письмо представляет код html

11. // Адрес smtp-сервера и порт, с которого будеn отправляться письмо

12. SmtpClient smtp = new SmtpClient("smtp.gmail.com", 587);

13. smtp.Credentials = new NetworkCredential("testpost7788@gmail.com", "agno sbkk crup

14. iei"); // Логин и пароль почты отправителя

15. smtp.EnableSsl = true; // Установка параметра безопасности

16. smtp.Send(m); // Отправка письма }}

Класс ErrorMy.cs предоставляет функционал, необходимы для вывода сообщений о ошибках, в зависимости от ситуации. Код класса представлен в листинге 7.

Листинг 7 – Код класса ErrorMy.cs

1. internal class ErrorMy { // Класс обработки ошибок
2. public string registerAlreadyExistingAccount() { return "Аккаунт с таким логином уже
3. существует";}
4. public string anotherError() { return "Неопознанная ошибка"; }
5. public string wrongPassword() { return "Неправильный пароль";}
6. public string wrongTwoCode() { return "Неверный код двухфакторной
7. аутентификации"; } } }

## 2.3 Структура приложения-хостинга

Приложение-хостинг ChatHost представляет собой консольное приложение, целью которого является запуск библиотеки классов wcf\_chat и создания приложения сервера с возможностью подключения к нему приложений-клиентов. Приложение-хостинг работает в локальной сети, но при желании, его можно захостить в глобальную сеть. Структура приложения-хостинга представлена на рисунке 3.

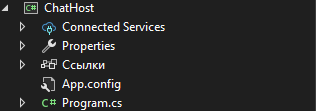


Рисунок 3 – Структура приложения-хостинга

Весь функционал приложения-хостинга – это запуск на своей основе библиотеки классов wcf\_chat, это реализуется в классе Program.cs. Код класса Program.cs представлен в листинге 8. В данном классе задаются стандартные параметры открытия соединения через var host = new ServiceHost( typeof( wcf\_chat.ServiceChat))), другого функционала тут нет.

Листинг 8 – Код класса Program.cs

1. namespace ChatHost { // Консольное приложение хост
2. internal class Program { // Хост для сервиса wcf\_chat
3. static void Main(string[] args) { // Главный метод класса
4. // использование библиотеки классов WCF
5. using (var host = new ServiceHost(typeof(wcf\_chat.ServiceChat))) {
6. host.Open(); // Запуск хоста
7. Console.WriteLine("Хост стартовал"); // Вывод сообщения о запуске хоста
8. Console.ReadLine(); // Ожидание ввода в консоль } } } }

Параметры подключения приложения-клиента описаны в файле App.Config. Основная часть кода файла App.config представлена в листинге 9.

Так к примеру, необходимо указать адрес хоста, в данном случае он указывается строкой <add baseAddress="net.tcp://localhost:8302" />.

Также необходимо указать адрес клиента: <add baseAddress="http://localhost:8301" />.

Листинг 9 – Код файла App.config

1. // Установка адреса в локальной сети
2. <client> <endpoint address="net.tcp://localhost:8302/" binding="netTcpBinding"
3. // Настройка параметров безопасности
4. bindingConfiguration="NetTcpBinding\_IServiceChat" contract="ServiceChat.IServiceChat"
5. name="NetTcpBinding\_IServiceChat">
6. // Установка параметров компьютера-клиента
7. <identity><userPrincipalName value="DESKTOP-HR0C7M0\deaan" /> </identity>
8. </endpoint> </client>
9. // Установка параметров сервера
10. <services><service name="wcf\_chat.ServiceChat" behaviorConfiguration="mexBeh">
11. // Установка библиотеки классов WCF
12. <endpoint address="" binding="netTcpBinding" contract="wcf\_chat.IServiceChat"/>
13. // Установка сетевого протокола
14. <endpoint address="mex" binding="mexHttpBinding" contract="IMetadataExchange" />
15. <host><baseAddresses> // Сетевые адреса
16. <add baseAddress="http://localhost:8301" /> // Адрес клиента
17. <add baseAddress="net.tcp://localhost:8302" /> // Адрес сервера
18. </baseAddresses></host></service></services>

## 2.4 Структура клиентской части приложения

Структура клиентской части приложения представлена на рисунке 4. Все объекты с расширением .xaml, кроме App.xaml и MainWindow.xaml, являются окнами, на которых расположен клиентский функционал. Визуальная часть приложения и руководство пользователя находятся в приложении 1.

Клиент создает переменную client, являющуюся экземпляром класса ServiceChatClient. Далее в программе, происходит обращения к методам этого класса – интерфейсам, которые создаются в листинге 1 сервера и функционал которых описывается в листинге 2 сервера. Так, обращение к экземпляру класса сервера в классе Login.xaml.cs, происходит через вызов интерфейса Login, описывающего логику авторизации пользователя: client.Login(log, pas);

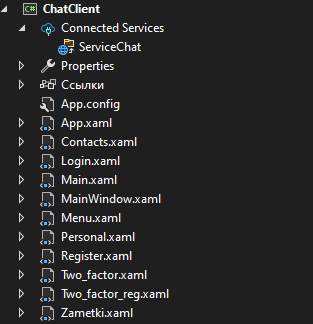


Рисунок 4 – Структура клиентской части приложения

WPF страница Main.xaml предоставляет пользователю доступ к страницам авторизации и регистрации. В коде файла Main.xsml.cs реализуется два события. Код Main.xaml.cs представлен в листинге 10. На странице Main.xaml не происходит взаимодействия с сервером. На данной странице происходит только навигация по страницам клиентской части приложения. Основной функционал – это переход на другую страницу, это осуществляется строкой NavigationService.Navigate( MainWindow.listPage[1]); с указанием индекса нужной строки.

Листинг 10 – Код класса Main.xaml.cs

1. namespace ChatClient { // Пространство имен
2. public partial class Main : Page {
3. public Main() { InitializeComponent(); } // Конструктор класса
4. // Событие нажатия кнопкаи для перехода на страницу авторизации
5. private void Button\_Click(object sender, RoutedEventArgs e)
6. { NavigationService.Navigate(MainWindow.listPage[1]); }
7. // Событие нажатия кнопки для перехода на страницу регистрации
8. private void Button\_Click\_1(object sender, RoutedEventArgs e)
9. { NavigationService.Navigate(MainWindow.listPage[2]); } } }

WPF страница Login.xaml предоставляет пользователю доступ к странице авторизации. В коде файла Login.xaml.cs реализуется два события. Код Login.xaml.cs представлен в листинге 11.

Как уже описано выше, клиент работает с функционалом сервера через экземпляр класса сервер ServiceChatClient client и его методы, именуемые интерфейсами. Мы обращаемся к данным интерфейсам, сервер получает эту команду и выполняет заложенную в них логику. Так, вызов интерфейса client.Login(log, pas) производит на сервере авторизацию пользователя по передаваемым параметрам log (логин пользователя), и pas (пароль пользователя).

Листинг 11 – Код Login.xaml.cs

1. public partial class Login : Page { // Класс логики страницы авторизации
2. // Объект типа хоста в клиенте, для взаимодействия с его методами
3. ServiceChatClient client;
4. public Login().{ InitializeComponent(); } // Конструктор класса
5. // Событие нажатия на кнопку отправки логина и пароля
6. private void enterButton\_Click(object sender, RoutedEventArgs e) {
7. client = MainWindow.client; // Инициализация клиента

8. string log = loginTBox.Text; // Получение логина с страницы

9. string pas = registerTBox.Text; // Получение почты с страницы

10. (string, string, string) gg = client.Login(log, pas); // Использование метода сервера

11. if (gg.Item1 == "Ошибка1") { MessageBox.Show($"Авторизация не прошла."); }

12. else if (gg.Item2 != "123")

13. { MessageBox.Show($"Авторизация прошла успешно. Пользователь {gg.Item1}.

14. почта {gg.Item2}"); // Вывод статуса авторизации

15. if (gg.Item3 == "OK") // Если необходима двухфакторная аутентификация

16. { MessageBox.Show("Письмо отправлено на почту"); // Статус отправки письма

17. // Переход на страницу проверки

18. NavigationService.Navigate(MainWindow.listPage[3]); }

Продолжение листинга 11.

19. else // Если двух. аутентификация не нужна

20. { MessageBox.Show("Письмо не отправлено на почту"); }

21. else // Если проверки пройдены, переход к странице пользователя

22. { MainWindow.active\_user\_login = gg.Item1;

23. MessageBox.Show($"Авторизация прошла успешно. Пользователь {gg.Item1}");

24. NavigationService.Navigate(MainWindow.listPage[8]); } } // Переход

25. private void backButton\_Click(object sender, RoutedEventArgs e) // Кнопка возврата

26. { NavigationService.Navigate(MainWindow.listPage[0]); } } }

WPF страница Two\_factor.xaml предоставляет пользователю доступ к странице двухфакторной аутентификации по почте. Переход на эту страницу происходит в случае наличия у пользователя почты. В коде файла Two\_factor.xaml.cs реализуется два события. Код Two\_factor.xaml.cs представлен в листинге 12.

Листинг 12 – Код Two\_factor.xaml.cs

1. ServiceChatClient client; // Объект типа хоста, для взаимодействия с его методами
2. public Two\_factor() // Констуктор класса
3. { InitializeComponent();
4. client = MainWindow.client; } // Инициализация клиента
5. // Событие нажатия кнопки проверки кода двухфакторной аутентификации
6. private void button1\_Click(object sender, RoutedEventArgs e) {
7. string cd = textBox1.Text; // Получение введенного кода
8. string status = client.TwoFact(cd); // Получение статуса проверки
9. if (status == "OK") // Если статус удовлетворяет
10. { MessageBox.Show("Проверка пройдена"); // Вывод сообщения
11. NavigationService.Navigate(MainWindow.listPage[8]); } // Переход к профилю
12. Else { MessageBox.Show("Проверка не пройдена"); } }

WPF страница Register.xaml предоставляет пользователю доступ к странице регистрации пользователя. В коде файла Register.xaml.cs реализуется два события. Основной код класса Register.xaml.cs представлен в листинге 13.

Листинг 13 – Код Register.xaml.cs

1. // Событие нажатия кнопки регистрации пользователя
2. private void enterButton\_Click(object sender, RoutedEventArgs e) {
3. loginReg = loginTBox.Text; // Получение логина
4. passwordReg = passwordTBox.Text; // Получение пароля
5. emailReg = emailTBox.Text; // Получение почты
6. client = MainWindow.client; // Инициализация клиента
7. otvetStatus = null; // Обнуление статуса ответа
8. if (emailReg == "") { // Если нет почты при регистрации
9. otvetReg=client.Register(loginReg,passwordReg,"").Item1; // Вызов метода
10. // регистрации
11. if (otvetReg != "Ошибка добавления пользователя. Возможно логин занят
12. // ERROR4")
13. { if (emailReg != "") // Если есть почта
14. { MessageBox.Show("Почта есть"); // Вывод сообщения
15. MessageBox.Show($"Статус регистрации: {otvetReg}"); }
16. Else // Если нет почты
17. { MessageBox.Show("Почты нет"); // Вывод сообщения
18. MessageBox.Show($"Статус регистрации: {otvetReg}"); }
19. MessageBox.Show("Пользователь успешно зарегистрирован");
20. NavigationService.Navigate(MainWindow.listPage[0]); } // Переход в начало
21. else { // Если регистрация не прошла
22. MessageBox.Show("Пользователь не зарегистрирован. Возможно логин уже
23. занят. 1"); } }
24. else if (client.two\_access() == "NO") // Если не нужна проверка почты
25. { (string, string) otvet = client.Register(loginReg, passwordReg, emailReg);
26. otvetReg = otvet.Item1; // установка статуса ответа
27. otvetCode = otvet.Item2; // Установка нового кода
28. otvetStatus = "OK"; // Установка статуса проверки
29. NavigationService.Navigate(MainWindow.listPage[5]); } } // Переход к профилю

WPF страница Two\_factor\_reg.xaml предоставляет пользователю доступ к странице проверки почты при регистрации пользователя. В коде файла Two\_factor\_reg.xaml.cs реализуется два события. Основной код класса Two\_factor\_reg.xaml.cs представлен в листинге 14.

Листинг 14 – Код Two\_factor\_reg.xaml.cs

1. // Событие нажатия кнопки проверки кода
2. private void button1\_Click(object sender, RoutedEventArgs e) {
3. string cd = textBox1.Text; // Получение кода
4. string status = Register.otvetStatus; // Получение статуса проверки
5. string codeReg = Register.otvetCode; // Получение кода для проверки
6. if (status == "OK") // Если статус удовлетворяет
7. { MessageBox.Show("Проверка пройдена");
8. client.two\_access\_izm("YES"); // Установка статуса
9. if (Register.otvetReg != "Ошибка добавления пользователя. Возможно логин
10. занят ERROR4")
11. { if (Register.emailReg != ""). // Если почты нет
12. { MessageBox.Show("Почта есть"); // Вывод сообщения
13. MessageBox.Show($"Статус регистрации: {Register.otvetReg}"); }
14. Else // Если к аккаунту привязана почта
15. { MessageBox.Show("Почты нет"); // Вывод сообщения
16. MessageBox.Show($"Статус регистрации: {Register.otvetReg}");}
17. MessageBox.Show("Пользователь успешно зарегистрирован");
18. NavigationService.Navigate(MainWindow.listPage[0]); }
19. else { MessageBox.Show("Пользователь не зарегистрирован. Возможно логин
20. уже занят."); } }
21. Else // Если проверка регистрации не прошла
22. { client.two\_access\_izm("NO"); // Изменение статуса проверки
23. MessageBox.Show("Проверка не пройдена"); } } // Вывод сообщения

WPF страница Personal.xaml предоставляет пользователю доступ к странице личного профиля. В коде файла Personal.xaml.cs реализуется четыре события. Основной код класса Personal.xaml.cs представлен в листинге 15. В листинге 15 представлено событие нажатия кнопки обновления данных.

Листинг 15 – Код Personal.xaml.cs

1. // Событие загрузки страницы - загрузка актуальных данных о пользователе

2. private void Grid\_Loaded(object sender, RoutedEventArgs e) {

3. // Обновление активного логина на странице

4. activeUserTextBlock.Text = $"Аккаунт: \_{client.get\_active\_login()}\_";

5. (string, string, string) userData = client.GetUserData();//Получение данных

6. activeUserNameTextBlock.Text = "Имя: " + userData.Item1;//Установка имени

7. activeUserLastNameTextBlock.Text="Фамилия:"+userData.Item2;//Установка фамилии

8. activeUserStatusNameTextBlock.Text="Статус: "+userData.Item3;} // Установка статуса

9. // Событие нажатия кнопки обновления данных пользователя

10. private void newDataButton\_Click(object sender, RoutedEventArgs e) { try {

11. string name = nameTB.Text; // Получение имени

12. string last\_name = last\_nameTB.Text; // Получение фамилии

13. string status = statusTB.Text; // Получение статуса

14. MessageBox.Show(client.NewUserData(name, last\_name, status));//Обновление данных

15. (string, string, string) userData = client.GetUserData(); // Получение новых данных

16. activeUserNameTextBlock.Text = "Имя: " + userData.Item1; // Установка имени

17. activeUserLastNameTextBlock.Text="Фамилия:"+userData.Item2;//Установка фамилии

18. activeUserStatusNameTextBlock.Text="Статус:"+userData.Item3;//Установка статуса

19. } catch { MessageBox.Show("Ошибка обновления данных"); } } } }

WPF страница Menu.xaml предоставляет пользователю доступ к странице меню. В коде файла Menu.xaml.cs реализуется пять событий. Код Menu.xaml.cs представлен в листинге 16.

Листинг 16 – Код Menu.xaml.cs

1. // Событие нажатия кнопки выхода из аккаунта
2. private void backButton\_Click(object sender, RoutedEventArgs e)
3. { NavigationService.Navigate(MainWindow.listPage[0]); // Переход к Main
4. MainWindow.active\_user\_email = null; // Очистка логина активного пользователя
5. MainWindow.active\_user\_login = null; } // Очистка почты активного пользователя
6. // Событие нажатия кнопки перехода к странице с заметками
7. private void zametkiButton\_Click(object sender, RoutedEventArgs e)

Продолжение листинга 16.

1. { NavigationService.Navigate(MainWindow.listPage[6]); }
2. // Событие загрузки страницы - обновление подписи логина активного пользователя
3. private void Grid\_Loaded(object sender, RoutedEventArgs e)
4. // Установка логина активного пользователя в меню
5. { try { activeUserTextBlock.Text = $"Аккаунт: \_{client.get\_active\_login()}\_"; }
6. catch { MessageBox.Show("??");} }
7. // Событие нажатия кнопки перехода к странице с контактами
8. private void contactsButton\_Click(object sender, RoutedEventArgs e)
9. { NavigationService.Navigate(MainWindow.listPage[7]); }
10. // Событие нажатия кнопки перехода к странице с профилем
11. private void personalButton\_Click(object sender, RoutedEventArgs e)
12. { NavigationService.Navigate(MainWindow.listPage[8]); } } }

WPF страница Zametki.xaml предоставляет пользователю доступ к личным заметкам. В коде файла Zametki.xaml.cs реализуется пять событий. Основной код класса Zametki.xaml.cs представлен в листинге 17.

По аналогии с описанием класса Login,xaml.cs, логика работы Zametki.xaml.cs не отличается. Также создается экземпляр класса сервера, идет обращение к его интерфейсам для использования функционала сервера.

Листинг 17 – Код Zametki.xaml.cs

1. // Событие кнопки добавления заметки
2. private void zametkiButton\_Click(object sender, RoutedEventArgs e) {
3. string note = noteTB.Text; // Получение текста заметки

4. client.AddUserNote(note); // Использование метода сервера. Добавление заметки

5. listBox.Items.Clear(); // Очистка списка заметок

6. List<string> list = new List<string>(); // Переменная список заметок

7. list = client.GetUserNotes().ToList(); // Получение заметок с сервера

8. foreach (string note1 in list) |// Цикл по списку заметок

9. { listBox.Items.Add(note1); } } // Запись заметок в список

WPF страница Contacts.xaml предоставляет пользователю доступ к списку контактов и чатам. В коде файла Contacts.xaml.cs реализуется десять событий. Часть кода Contacts.xaml.cs представлена в листинге 18. Полный код класса Contacts.xaml.cs представлен в приложении Г.

Листинг 18 – Код Contacts.xaml.cs

1. public void updateChat() { // Функция обновления чата

2. if (comboBoxCont.SelectedValue != null ) { // Проверка на то, не пуст ли выбор контакта

3. int j = 0; // Проверка на то, находится ли контакт в черном списке:

4. for (int i = 0; i < contBlackList.Count; i++)

5. { if (comboBoxCont.SelectedValue.ToString() == contBlackList[i]) j = 1; }

6. if (j == 0) { try { // Блок перехвата ошибок

7. List<string> messList = new List<string>(); // Список сообщений

8. messListBox.ItemsSource = messList; // Источник данных для списка сообщений

9. string comb = comboBoxCont.SelectedValue.ToString(); // Выбранный контакт

10. int k1 = 0; // Переменная индикатор

11. contBlackList.Clear(); // Очистка черного списка. Получение его значений:

12. contBlackList = client.GetContList().Item2.ToList();

13. for (int i = 0; i < contBlackList.Count; i++) // Цикл по черному списку

14. { if (contBlackList[i] == comb) { k1 = 1; } } // Проверка, есть ли такой контакт

15. if (k1 == 1) // Если контакт в черном списке

16. { MessageBox.Show("Контакт в черном списке. "); return; }

17. Else // Если контакт не в черном списке, открытие чата

18. { messList.Clear(); // Очистка чата

19. messList = client.GetChatMesList(comb).ToList(); } // Загрузка сообщений

20. messListBox.ItemsSource = messList; } // Установка списка сообщений

21. catch { } } } }

## 2.5 Интерфейс клиентской части приложения

Первоначальная страница

При запуске приложения открывается окно выбора авторизации и регистрации (Рисунок 5).

При нажатии на кнопку «Авторизация», пользователь будет перенесен на страницу авторизации.

При нажатии на кнопку «Регистрация» пользователь будет перенесен на страницу регистрации.

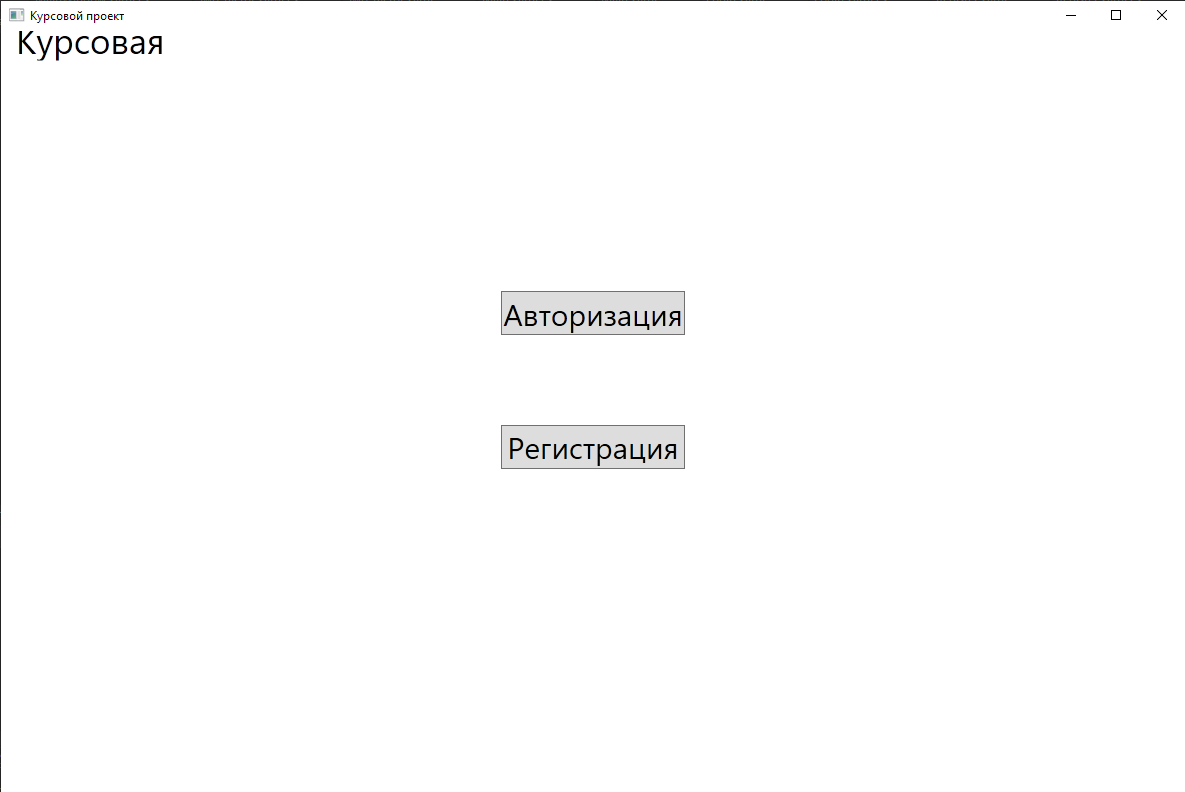


Рисунок 5 – Первоначальная страница

Страницы регистрации и проверки почты при регистрации

Для регистрации нового аккаунта необходимо вести логин и пароль. По желанию, можно ввести почту. После заполнения полей регистрации, необходимо нажать на кнопку «Ввести данные» (Рисунок 6). Если пользователем введена почта, то пользователь будет перенесен в окно проверки почты (Рисунок 7). Если почта не была введена, аккаунт зарегистрируется без нее. В случае наличия почты, на указанную пользователем почту, будет прислан код двухфакторной аутентификации код. Для проверки кода двухфакторной аутентификации, его необходимо ввести в поле ввода и для продолжения взаимодействия, необходимо нажать кнопку «Далее».

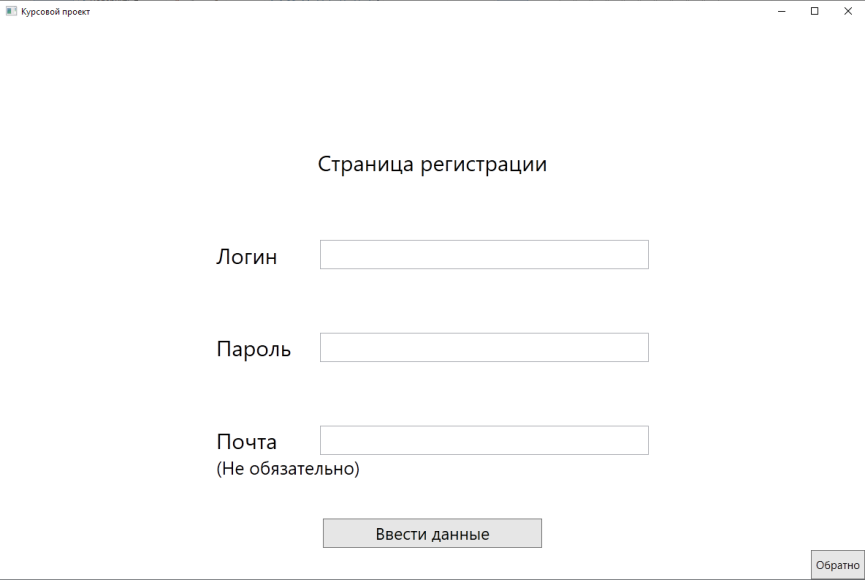


Рисунок 6 – Страница регистрации

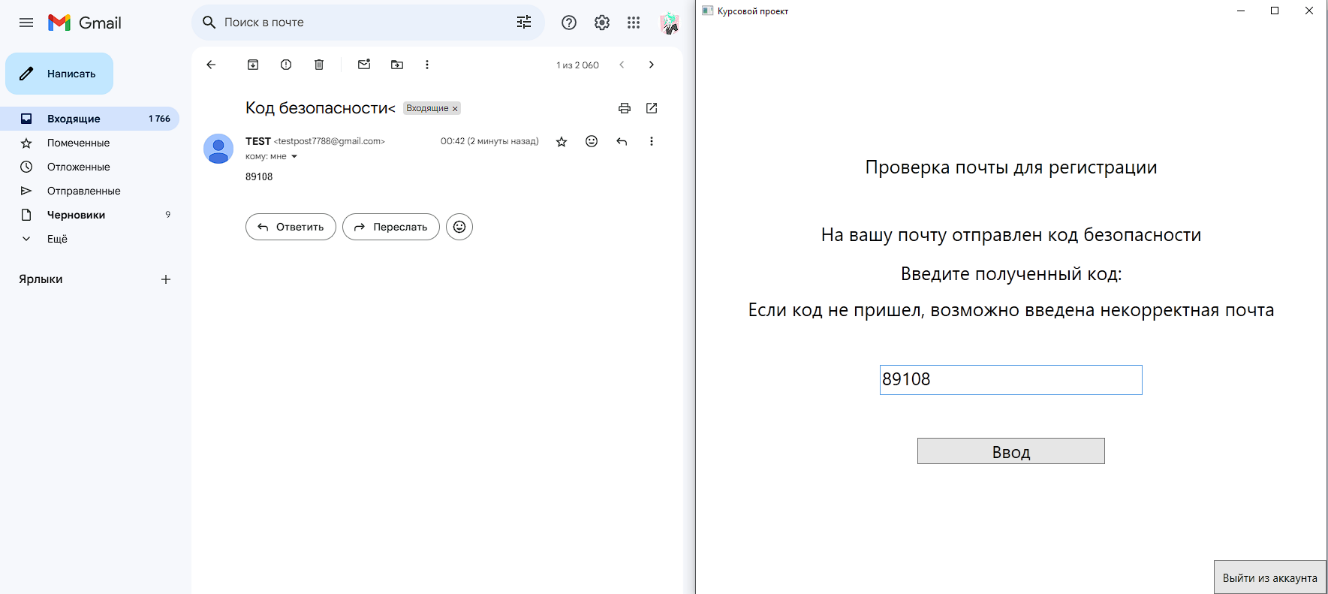


Рисунок 7 – Страница проверки почты при регистрации

Страницы авторизации и двухфакторной аутентификации

Для того чтобы приступить к работе с приложением, пользователю необходимо ввести свои данные в окне авторизации и нажать на кнопку «Ввести данные» (Рисунок 8). В случае наличия почты на зарегистрированном аккаунте, пользователь будет перенесен на страницу двухфакторной аутентификации, на ней необходимо ввести код, который должен быть отправлен на указанную при регистрации почту (Рисунок 9). В случае если при регистрации аккаунта, пользователь не указал почту, будет произведен переход непосредственно в профиль, минуя страницу двухфакторной аутентификации.

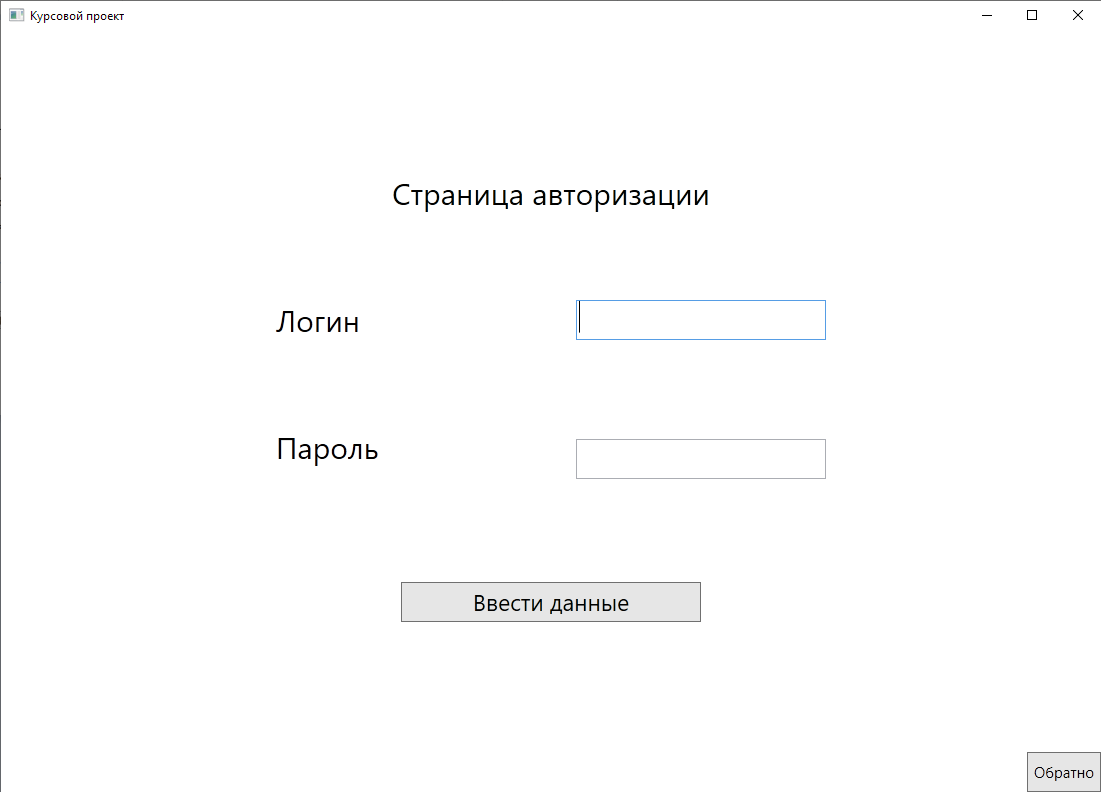


Рисунок 8 – Страница авторизации

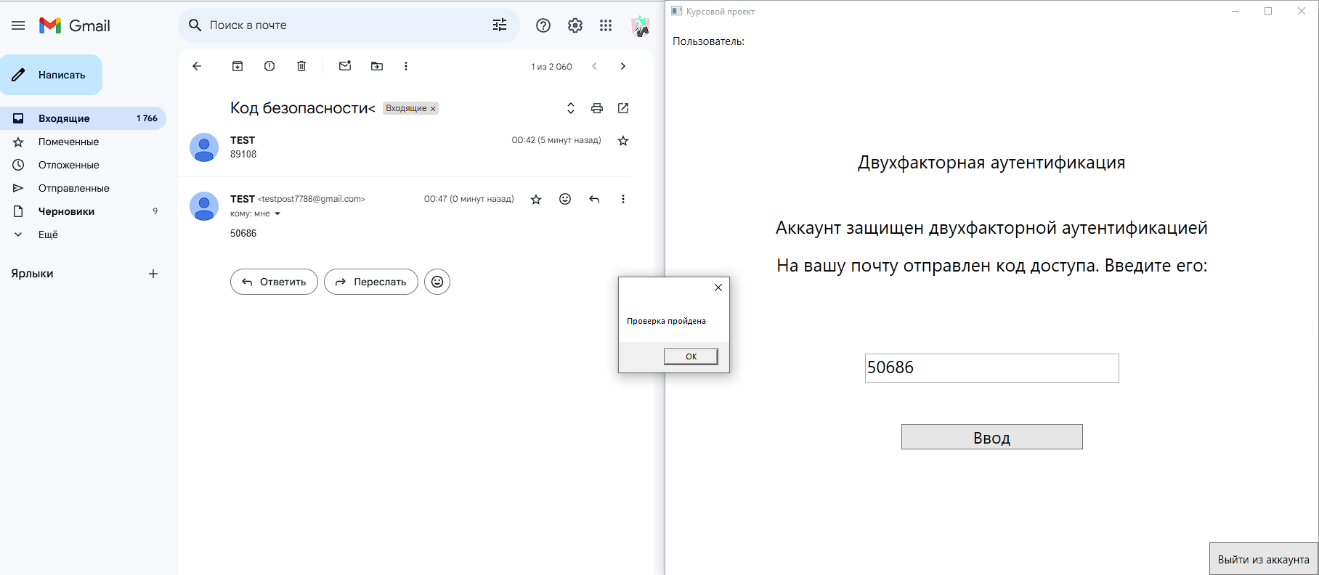


Рисунок 9 – Страница двухфакторной аутентификации

Страница профиля

После успешной авторизации, пользователь попадает в профиль (Рисунок 10). Здесь отображены актуальные данные пользователя, которые можно изменить, введя новые данные в блоки ввода и при нажатии кнопки обновления данных. Также на странице находятся кнопки выхода из аккаунта и перехода в меню. Все данные о пользователе хранятся в личной таблице в базе данных, и доступ к этим данным имеет только авторизированный, актуальный на данном клиенте пользователь. Другие пользователи не имеют доступа к чужим данным профиля, эти данные конфиденциальны. В поля могут быть введены данные любых типов, как строчные, так и числовые, это никак не влияет на работу приложения.

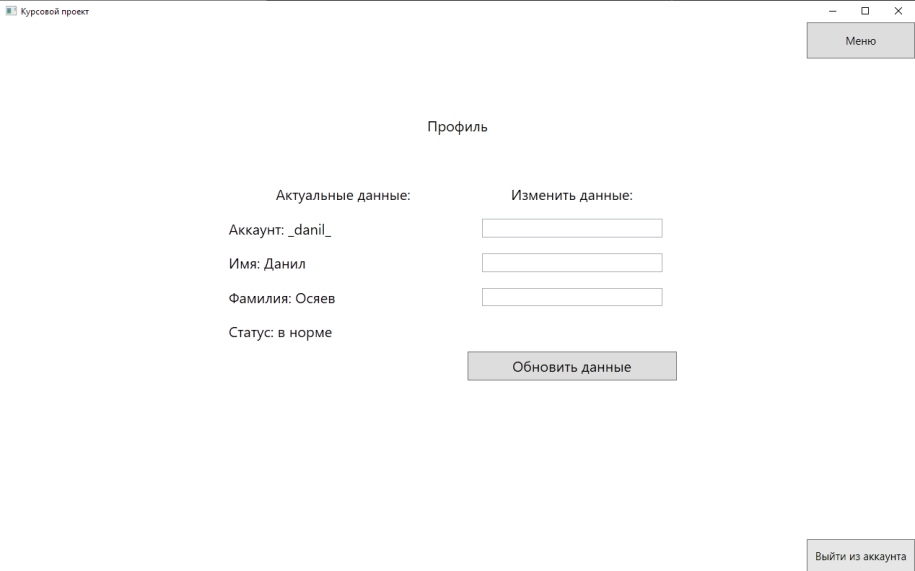


Рисунок 10 – Страница личного профиля

Страница меню

На странице меню находится четыре кнопки навигации. Кнопка «Заметки» направляет пользователя на страницу с личными заметками. Кнопка «Контакты» переводит пользователя на страницу с чатами и контактами. Кнопка «Профиль» переводит пользователя на страницу с его личным профилем. Кнопка «Выйти из аккаунта» предназначена для выхода из аккаунта (Рисунок 11). После нажатия кнопки «Выйти из аккаунта», пользователь будет перенесен на первоначальную страницу, где сможет выбрать авторизацию или регистрацию.



Рисунок 11 – Страница меню

Страница заметок

На странице заметок, пользователь может добавлять личные заметки и удалять их. На странице находится четыре кнопки. Кнопка «Добавить запись» добавляет текст из блока текста слева в список заметок. Кнопка «Очистить записи» удаляет все заметки. Кнопка «Выйти из аккаунта» предназначена для выхода из аккаунта. Кнопка «Меню» переводит пользователя в меню. (Рисунок 12). Заметки пользователя хранятся в таблице базы данных, и доступ к данной таблице имеет только авторизированный, активный на данном клиенте пользователь.

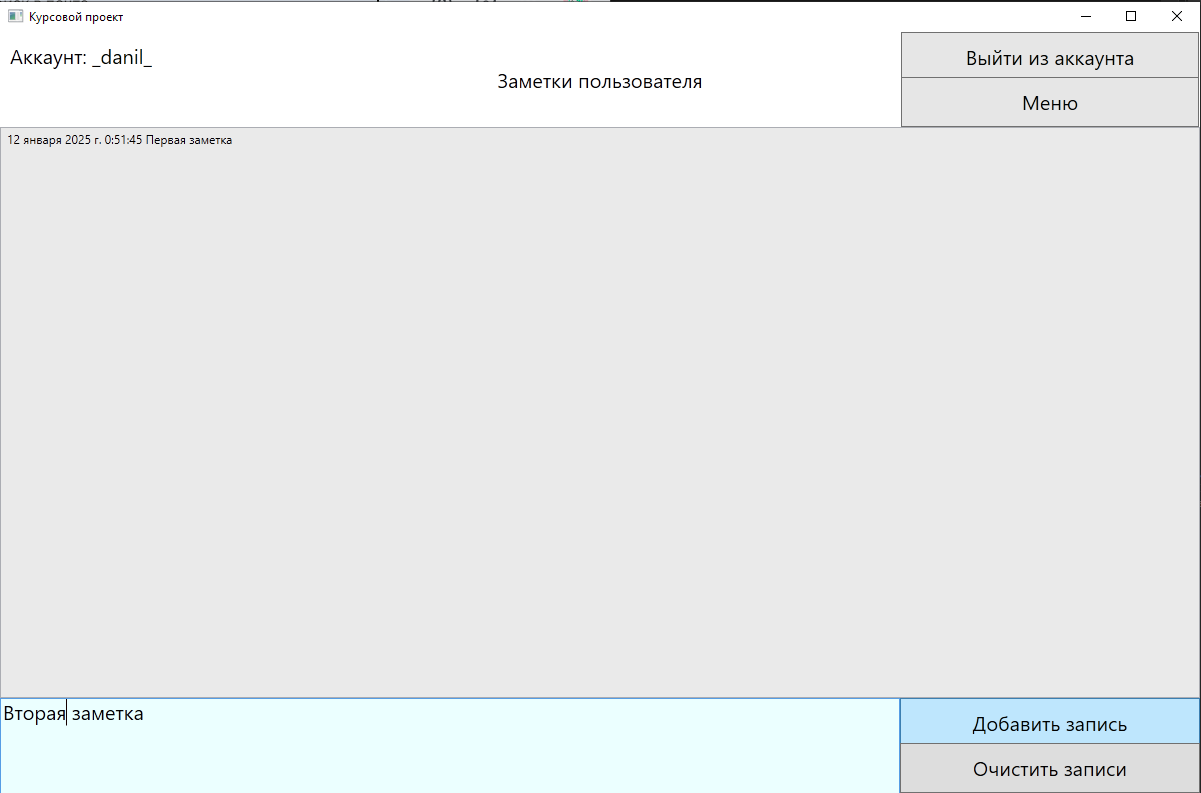


Рисунок 12 – Страница заметок

Страница контактов

На странице контактов, пользователь может взаимодействовать со своим списком контактов: добавлять контакты, удалять контакты, добавлять контакты в черный список, удалять контакты из черного списка. Также на странице контактов, пользователь может взаимодействовать с чатом. Пользователь может выбрать контакт, открыть с ним чат, и начать переписку в реальном времени. (Рисунок 13).

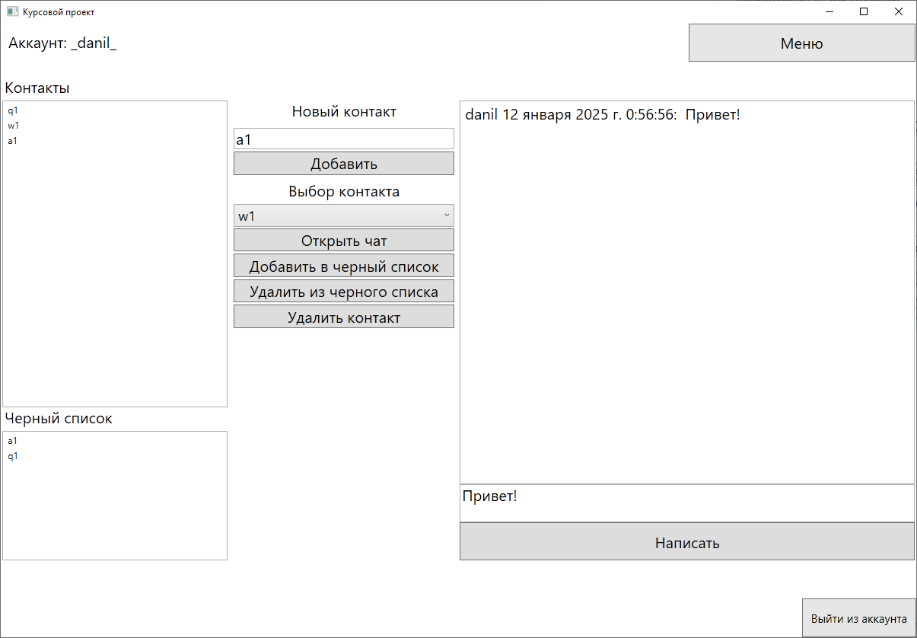


Рисунок 13 – Страница контактов

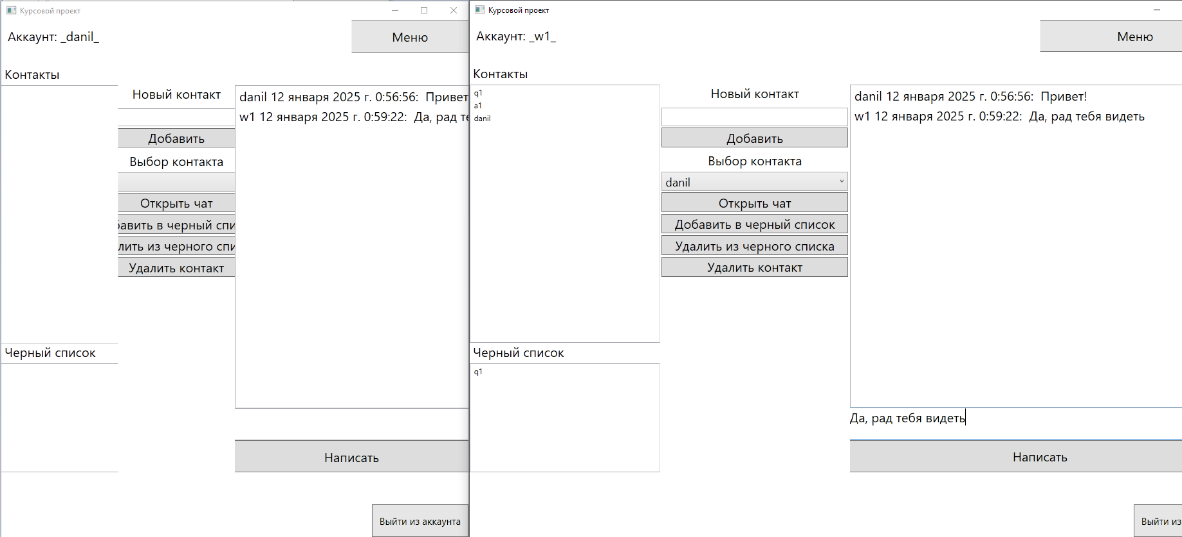


Рисунок 14 – Переписка двух пользователей в реальном времени

Общение двух пользователей происходит в реальном времени, это достигнуто с помощью постоянного обновления списка сообщений в чате на каждом клиенте. Также постоянно обновляется список контактов пользователя, список и контактов черном чате.

## 2.6 Серверная часть приложения

Серверная часть приложения – это библиотека классов WCF, где хранится весь код с логикой работы приложения, и простое консольное приложение, целью которого является только запуск библиотеки классов и настройка доступа клиента к серверу, по средствам настройки сетевых параметров.

Интерфейс консольного приложения представлен на рисунке 15.

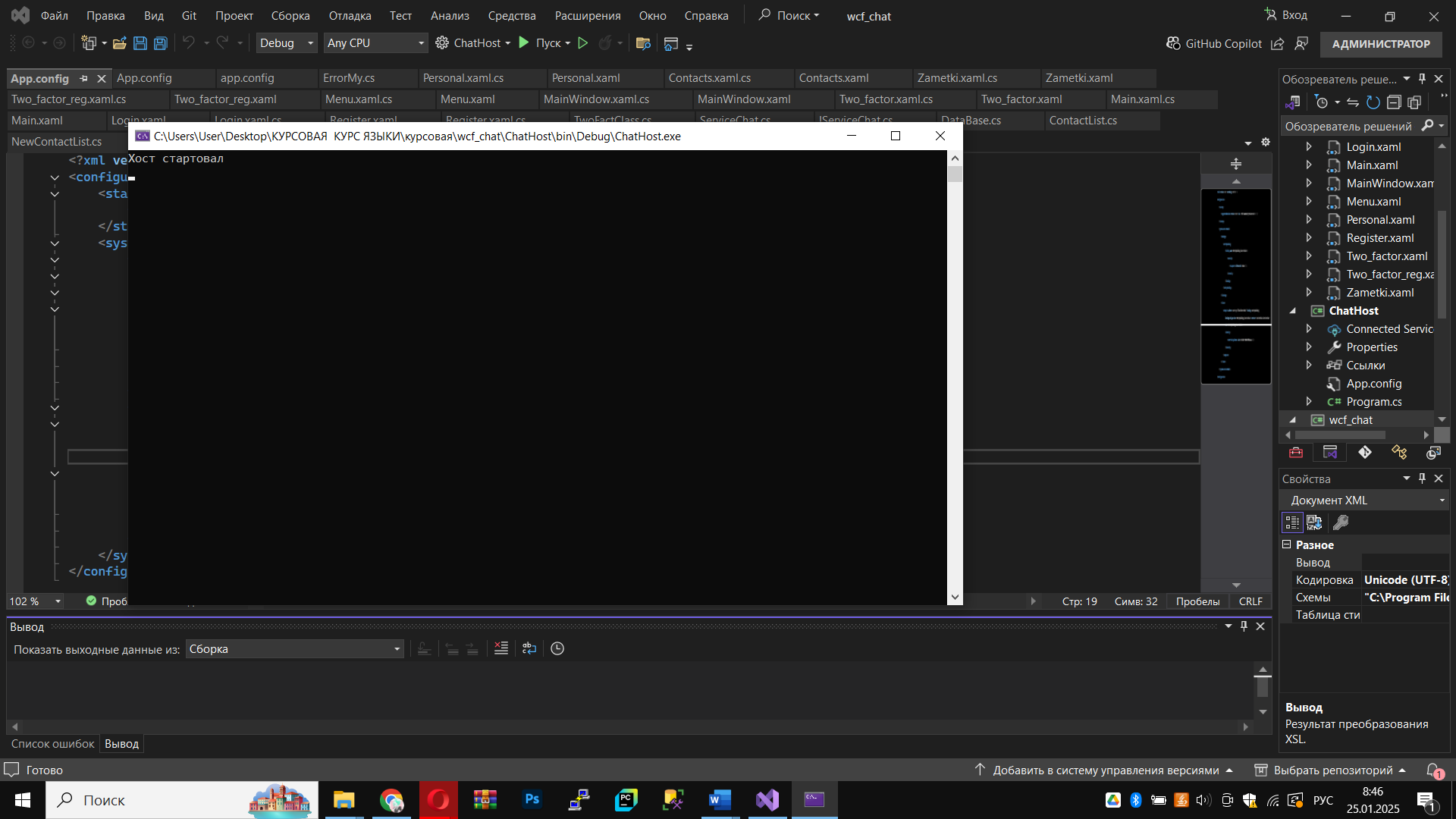


Рисунок 15 – Консольное приложение сервер

При запуске приложения, в случае успешного старта и если приложение работает правильно, в консоли должно отобразиться сообщение «Хост стартовал». Какой-либо ручной ввод сообщений в консоль приведет к немедленному закрытию приложения-хоста, все подключенные клиенты потеряют соединение с сервером.

Также, если любым пользователем будет использована функция двухфакторной аутентификации, в консоль будет выведено сообщение об отправке письма с кодом на почту. Демонстрация оправки сообщения на почту представлена на рисунке 16.

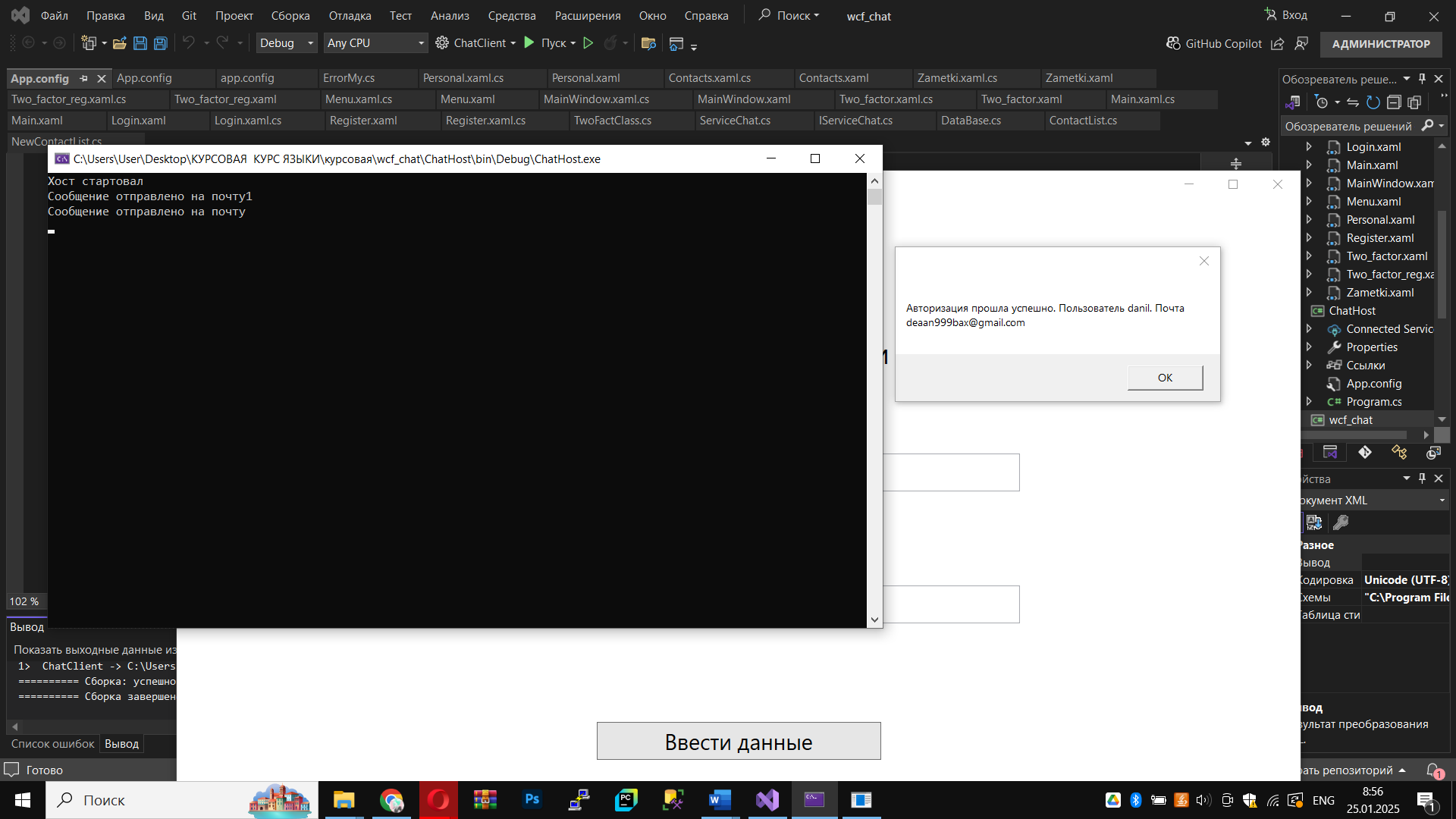


Рисунок 16 – Сообщение об отправке письма на почту

Приложение-хостинг не имеет какого-либо функционала, кроме запуска библиотеки классов WCF. Но при необходимости, приложение может быть развито, с добавлением функционала администратора через данную консоль, или через отдельное приложение.

# Заключение

Выполнение данной курсового проекта потребовало больших усилий и большого количества времени. В ходе реализации проекта, были изучены технологии WPF и WCF, была изучена клиент-серверная архитектура.

В рамках курсового проекта была выполнена разработка Windows-приложения мессенджера с использованием технологий WPF и WCF. Целью проекта являлось создание удобного, безопасного и функционального инструмента для обмена сообщениями в режиме реального времени. Разработанное приложение демонстрирует удобный пользовательский интерфейс и надёжное взаимодействие между клиентами через сервер.

Результаты работы включают:

- Проектирование и реализация клиент-серверной архитектуры.

- Реализация базового функционала мессенджера, включая регистрацию пользователей, управление профилями, отправку сообщений и хранение истории переписки.

- Использование принципов ООП для улучшения читаемости и поддержки кода.

Перспективы развития:

**- Кроссплатформенность:** добавление поддержки мобильных устройств и веб-версии.

**- Мультимедийные возможности:** отправка файлов, изображений и видео.

**- Улучшение безопасности:** внедрение двухфакторной аутентификации и шифрования сообщений.

**- Масштабируемость:** интеграция с облачными сервисами для хранения данных.

Данный проект демонстрирует возможности современных технологий разработки программного обеспечения, а также готовую основу для последующего масштабирования и усовершенствования.

# Список литературы

1. [Троелсен Э.](https://djvu.online/author/%D0%A2%D1%80%D0%BE%D0%B5%D0%BB%D1%81%D0%B5%D0%BD+%D0%AD.) , [Джепикс Ф.](https://djvu.online/author/%D0%94%D0%B6%D0%B5%D0%BF%D0%B8%D0%BA%D1%81+%D0%A4.) Язык программирования С# 7 и платформы .NET и .NET Core, 8-е изд. : Пер. с англ. - СПб. : ООО "Диалектика", 2018 - 1328 с. : ил. - Парал. тит. англ.
2. Скит Д. C# для профессионалов: тонкости программирования, 3-е изд.: Вильямс, 2019-608с.
3. C# Programming Guide [Электронный ресурс] - Режимы доступа: <https://learn.microsoft.com/en-us/dotnet/csharp/programming-guide/interop/how-to-access-office-onterop-objects>/ - (Дата обращения 27.12.2024).
4. СМК-СМГТУ-29-06. Система менеджмента качества. Стандарт организации. Структура, содержание и изложение, правила оформления и обозначения документации системы менеджмента качества.
5. СМК-МИ 29.2-06. Общие требования к построению, содержанию, оформлению, обозначению и управлению Стандартом университета (организации) и Методической инструкцией.

# Приложение А

(Обязательное)

Листинг кода класса IServiceChat.cs

// Установка начальных параметров класса интерфейса.

[ServiceContract(CallbackContract = typeof(IServerChatCallback))]

public interface IServiceChat

{

// Метод для создания новой строки с данными о пользователе

[OperationContract]

string NewUserData(string name, string last\_name, string status);

// Метод для получения данных о пользователе

[OperationContract]

(string, string, string) GetUserData();

// Метод для добавления нового сообщения в таблицу чата

[OperationContract]

string NewMessChat(string log, string message);

// Метод для получения списка контактов

[OperationContract]

(List<string>, List<string>) GetContList();

// Метод для добавления нового контакта

[OperationContract]

string NewCont(string log);

// Метод для получения списка сообщений чата из базы данных

[OperationContract]

List<string> GetChatMesList(string log);

// Метод для добавления контакта в черный список

[OperationContract]

string NewBlackCont(string log);

// Метод для удаления контакта из черного списка

Продолжение листинга IServiceChat.cs

[OperationContract]

string DelBlackCont(string log);

// Метод для удаления контакта

[OperationContract]

string DelCont(string log);

// Метод для авторизации пользователя

[OperationContract]

(string log, string email, string twoStatus) Login(string log, string pas);

// Метод для проверки кода двухфакторной аутентификации

[OperationContract]

string TwoFact(string UserCode);

// Метод для получения списка заметок пользователя

[OperationContract]

List<string> GetUserNotes();

// Метод для добавления новой заметки в таблицу с заметками пользователя

[OperationContract]

string AddUserNote(string note);

// Метод для очистки таблицы заметок пользователя

[OperationContract]

string DellUserNote();

// Метод для получения статуса двухфакторной аутентификации пользователя

[OperationContract]

string two\_access();

// Метод для изменения статуса двух

[OperationContract]

void two\_access\_izm(string status);

// Метод для получения логина активного пользователя

[OperationContract]

Продолжение листинга IServiceChat.cs

string get\_active\_login();

// Метод для регистрации пользователя

[OperationContract]

(string, string) Register(string log, string pass, string email);

}

# Приложение Б

(Обязательное)

Листинг кода класса ServiceChat.cs

// Класс для реализации интерфейса, написаного в родиельском классе

[ServiceBehavior(InstanceContextMode = InstanceContextMode.PerSession)]

public class ServiceChat : IServiceChat {

DataBase database = new DataBase();

// Логин активного пользователя

private string active\_user\_login = null;

// Почта активного пользователя

public string active\_user\_email = null;

// Переменная для хранения кода двухфакторной аутентификации пользователя

public string myCode = null;

// Переменная для хранения правильного кода двухфакторной аутентификации

public string temporary\_code = null;

// Переменная для хранения статуса двухфакторной аутентификации

public string two\_access = "NO";

// Переменная для хранения списка заметок пользователя

public List<string> user\_notes = null;

// Экземпляр класса для работы с контактами

ContactList contLists = null;

// Экземпляр класса обработки ошибок

ErrorMy myError = new ErrorMy();

// Метод для создания новой строки с данными о пользователе в таблице с данными о пользователях.

public string NewUserData(string name, string last\_name, string status) {

try {

Продолжение листинга ServiceChat.cs

// Классы, необходимые для работы с базой данных

SqlDataAdapter adapter = new SqlDataAdapter();

DataTable table = new DataTable();

// Строка запроса

string queryString = $"use kurs; update usersData set user\_first\_name = '{name}', user\_last\_name = '{last\_name}', user\_status = '{status}' where user\_login = '{active\_user\_login}';";

// Объект класса sqlCommand

SqlCommand command1 = new SqlCommand(queryString, database.getConnection());

// Работа с адаптером

adapter.SelectCommand = command1;

adapter.Fill(table);

return "OK"; }

catch { return "NOTOK"; } }

// Метод для получения данных о пользователе

public (string, string, string) GetUserData() {

string name = "";

string last\_name = "";

string status = "";

// Классы, необходимые для работы с базой данных

SqlDataAdapter adapter = new SqlDataAdapter();

DataTable table = new DataTable();

// Строка запроса

string queryString = $"use kurs; select user\_first\_name, user\_last\_name, user\_status from usersData where user\_login = '{active\_user\_login}';";

// Объект класса sqlCommand

SqlCommand command1 = new SqlCommand(queryString, database.getConnection());

Продолжение листинга ServiceChat.cs

// Работа с адаптером

adapter.SelectCommand = command1;

adapter.Fill(table);

DataRow[] rows = table.Select();

name = rows[0][0].ToString();

last\_name = rows[0][1].ToString();

status = rows[0][2].ToString();

return (name, last\_name, status); }

// Метод для добавления нового сообщения в таблицу чата

public string NewMessChat(string log, string message) {

string newMess = $"{active\_user\_login + " " + DateTime.Now.Date.ToLongDateString() + " " + DateTime.Now.ToLongTimeString() + ": " + message}";

string status = contLists.NewMessage(log, active\_user\_login, newMess);

return status; }

// Метод для получения списка сообщений чата из базы данных

public List<string> GetChatMesList(string log) {

List<string> listMess = new List<string>();

listMess = contLists.GetChatMesList(log, active\_user\_login);

return listMess; }

// Метод для добавления нового контакта

public string NewCont(string log) {

string status = contLists.NewContact(log, active\_user\_login);

return status; }

// Метод для добавления контакта в черный список

public string NewBlackCont(string log) {

string status = contLists.NewBlackContact(log, active\_user\_login);

return status; }

// Метод для удаления контакта из черного списка

Продолжение листинга ServiceChat.cs

public string DelBlackCont(string log) {

contLists.DelBlackContact(log, active\_user\_login);

return "OK"; }

// Метод для удаления контакта

public string DelCont(string log) {

contLists.DelContact(log, active\_user\_login);

return "OK"; }

// Метод для получения списка контактов

public (List<string>, List<string>) GetContList() {

contLists = null;

contLists = new ContactList(active\_user\_login);

List<string> listCont = new List<string>();

List<string> listBlackCont = new List<string>();

listCont = contLists.GetContactLists().Item1;

listBlackCont = contLists.GetContactLists().Item2;

return (listCont, listBlackCont); }

// Метод для получения списка заметок пользователя

public List<string> GetUserNotes() {

user\_notes = null;

user\_notes = new List<string>();

// Классы, необходимые для работы с базой данных

SqlDataAdapter adapter = new SqlDataAdapter();

DataTable table = new DataTable();

// Строка запроса

string queryString = $"use kurs; select note\_date, note\_text from {active\_user\_login + "Notes"};";

// Объект класса sqlCommand

SqlCommand command1 = new SqlCommand(queryString, database.getConnection());

Продолжение листинга ServiceChat.cs

// Работа с адаптером

adapter.SelectCommand = command1;

adapter.Fill(table);

DataRow[] rows = table.Select();

for (int i = 0; i < rows.Length; i++) { user\_notes.Add(rows[i][0].ToString() + " " + rows[i][1].ToString()); }

database.closeConnection();

return user\_notes; }

// Метод для добавления новой заметки в таблицу с заметками пользователя

public string AddUserNote(string note) {

// Классы, необходимые для работы с базой данных

SqlDataAdapter adapter = new SqlDataAdapter();

DataTable table = new DataTable();

// Строка запроса

string queryString = $"use kurs; insert into {active\_user\_login + "Notes"} (user\_login, note\_date, note\_text) values ('{active\_user\_login}', '{DateTime.Now.Date.ToLongDateString() + " " + DateTime.Now.ToLongTimeString()}', '{note}');";

// Объект класса sqlCommand

SqlCommand command1 = new SqlCommand(queryString, database.getConnection());

// Работа с адаптером

adapter.SelectCommand = command1;

adapter.Fill(table);

database.closeConnection();

return "OK"; }

// Метод для очистки таблицы заметок пользователя

public string DellUserNote() {

// Классы, необходимые для работы с базой данных

Продолжение листинга ServiceChat.cs

SqlDataAdapter adapter = new SqlDataAdapter();

DataTable table = new DataTable();

// Строка запроса

string queryString = $"use kurs; delete {active\_user\_login + "Notes"};";

// Объект класса sqlCommand

SqlCommand command1 = new SqlCommand(queryString, database.getConnection());

// Работа с адаптером

adapter.SelectCommand = command1;

adapter.Fill(table);

database.closeConnection();

return "OK"; }

// Метод для получения логина активного пользователя

string IServiceChat.get\_active\_login() {

database.closeConnection();

string al;

al = $"{active\_user\_login}";

return al; }

// Метод для получения статуса двухфакторной аутентификации пользователя

string IServiceChat.two\_access()

{

return two\_access;

}

// Метод для изменения статуса двух

void IServiceChat.two\_access\_izm(string status) {

two\_access = status; }

// Метод для регистрации пользователя

public (string, string) Register(string log, string pass, string email) {

Продолжение листинга ServiceChat.cs

database.closeConnection();

string newLogin = log;

//Хэширование пароля

string newPassword = Md5.hashPassword(pass);

string newEmail = email;

string statusReg = "NO";

string codeSend = null;

// Классы, необходимые для работы с базой данных

SqlDataAdapter adapter = new SqlDataAdapter();

DataTable table = new DataTable();

// Строка запроса

string queryString1 = $"use kurs; select user\_login from users where user\_login = '{newLogin}';";

// Объект класса sqlCommand

SqlCommand command1 = new SqlCommand(queryString1, database.getConnection());

// Работа с адаптером

adapter.SelectCommand = command1;

adapter.Fill(table);

if (table.Rows.Count == 0) {

statusReg = "Успешно добавлен пользователь 1";

if (newEmail != ""){

TwoFactClass two = new TwoFactClass(newEmail);

//codeSend = $"{two.GetCode()}";

codeSend = $"{two.GetCode()}";

two.status = null;

two = null; }

NewContactList newContactList1 = new NewContactList(newLogin);

Продолжение листинга ServiceChat.cs

string queryString2 = $"use kurs; insert into users (user\_login, user\_password, user\_email) values ('{newLogin}', '{newPassword}', '{newEmail}');";

string queryString3 = $"use kurs; create table {newLogin + "Notes"} (note\_id int identity(1,1) not null, user\_login varchar(20) not null, note\_date varchar(30) not null, note\_text varchar(200) not null primary key (note\_id));";

string queryString4 = $"use kurs; insert into usersData (user\_login, user\_first\_name, user\_last\_name, user\_status) values ('{newLogin}', ' ', ' ', 'Новый пользователь');";

// Объект класса sqlCommand

SqlCommand command2 = new SqlCommand(queryString2, database.getConnection());

SqlCommand command3 = new SqlCommand(queryString3, database.getConnection());

SqlCommand command4 = new SqlCommand(queryString4, database.getConnection());

// Работа с адаптером

adapter.SelectCommand = command2;

adapter.Fill(table);

adapter.SelectCommand = command3;

adapter.Fill(table);

adapter.SelectCommand = command4;

adapter.Fill(table);

statusReg = "Успешно добавлен пользователь"; }

else { statusReg = myError.registerAlreadyExistingAccount(); }

database.closeConnection();

return (statusReg, codeSend); }

// Метод для проверки кода двухфакторной аутентификации

public string TwoFact(string UserCode) {

string usCode = UserCode;

Продолжение листинга ServiceChat.cs

if (myCode == usCode) {

return "OK"; }

return myError.wrongTwoCode(); }

// Метод для авторизации пользователя

public (string log, string email, string twoStatus) Login(string log, string pas) {

database.closeConnection();

string login = log;

string password = Md5.hashPassword(pas);

// Классы, необходимые для работы с базой данных

SqlDataAdapter adapter = new SqlDataAdapter();

DataTable table = new DataTable();

// Строка запроса

string queryString = $"use kurs; select user\_login, user\_password, user\_email from users where user\_login = '{login}' and user\_password = '{password}'";

// Объект класса sqlCommand

SqlCommand command = new SqlCommand(queryString, database.getConnection());

// Работа с адаптером

adapter.SelectCommand = command;

adapter.Fill(table);

DataRow[] rows = table.Select();

string sqlStr = "";

for (int i = 0; i < rows.Length; i++) { sqlStr += rows[i][2]; }

database.closeConnection();

string twoOk = null;

if (table.Rows.Count == 1) {

try { active\_user\_email = rows[0][2].ToString(); }

catch { return ("Ошибка2", "Ошибка2", "Ошибка2"); }

active\_user\_login = $"{login}";

Продолжение листинга ServiceChat.cs

if (active\_user\_email == "") { active\_user\_email = "132"; }

else {

TwoFactClass two = new TwoFactClass(active\_user\_email);

twoOk = $"{two.status}";

myCode = two.cod;

two.status = null; }

active\_user\_login = log;

return (active\_user\_login, active\_user\_email, twoOk); }

else { return ("Ошибка1", "Ошибка1", "Ошибка1"); } } }

# Приложение В

(Обязательное)

Листинг кода класса ContactList

// Класс для работы с контактами и чатами

internal class ContactList

{

// Объект для работы с базой данных

DataBase database = new DataBase();

// Создание переменной с списком контактов

private List<string> contList;

// Создание переменной с черным списком

private List<string> contBlackList;

// Создание переменной с логином

private string login;

// Создание переменной с списком всех пользователей в базе

private List<string> allUsersList;

// Главный конструктор

public ContactList(string login)

{

this.login = login;

contList = new List<string>();

contBlackList = new List<string>();

allUsersList = new List<string>();

// Классы, необходимые для работы с базой данных

SqlDataAdapter adapter = new SqlDataAdapter();

DataTable table = new DataTable();

// Строка запроса

string queryString = $"use kurs; select contact\_login from {login + "Contacts"};";

Продолжение листинга ContactList

string queryBlackString = $"use kurs; select contact\_login from {login + "BlackContacts"};";

// Объект класса sqlCommand

SqlCommand command1 = new SqlCommand(queryString, database.getConnection());

SqlCommand commandBlack = new SqlCommand(queryBlackString, database.getConnection());

// Работа с адаптером

adapter.SelectCommand = command1;

adapter.Fill(table);

DataRow[] rows = table.Select();

for (int i = 0; i < rows.Length; i++) { contList.Add(rows[i][0].ToString()); }

// Работа с адаптером

adapter.SelectCommand = commandBlack;

adapter.Fill(table);

DataRow[] rowsBlack = table.Select();

for (int i = 0; i < rowsBlack.Length; i++) { contBlackList.Add(rowsBlack[i][0].ToString()); }

database.closeConnection();

}

// Метод для создания нового сообщения в чате

public string NewMessage(string log, string actLog, string message)

{

try

{

List<string> listMess = new List<string>();

SqlDataAdapter adapter = new SqlDataAdapter();

DataTable table = new DataTable();

string log1 = actLog;

Продолжение листинга ContactList

string log2 = log;

// Сравнение того, какая строка выше по словарю.

// Чтобы создавалась таблица с корректным названием

int sravn = string.Compare(log1, log2);

if (sravn > 0)

{

string per = log1;

log1 = log2;

log2 = per;

}

string queryString = $"use kurs; insert into {log1 + "CHAT" + log2} (mess) values ('{message}');";

SqlCommand commandBlack = new SqlCommand(queryString, database.getConnection());

// Работа с адаптером

adapter.SelectCommand = commandBlack;

adapter.Fill(table);

database.closeConnection();

return "OK";

}

catch

{

database.closeConnection();

return "NOTOK";

}

}

// Метод для получения списка сообщений в чате

public List<string> GetChatMesList(string log, string activeLogin)

{

Продолжение листинга ContactList

List<string> listMess = new List<string>();

SqlDataAdapter adapter = new SqlDataAdapter();

DataTable table = new DataTable();

string log1 = activeLogin;

string log2 = log;

// Сравнение того, какая строка выше по словарю.

// Чтобы создавалась таблица с корректным названием

int sravn = string.Compare(log1, log2);

if (sravn > 0)

{

string per = log1;

log1 = log2;

log2 = per;

}

string queryString = $"use kurs; select mess from {log1 + "CHAT" + log2};";

SqlCommand commandBlack = new SqlCommand(queryString, database.getConnection());

// Работа с адаптером

adapter.SelectCommand = commandBlack;

adapter.Fill(table);

DataRow[] rows = table.Select();

for (int i = 0; i < rows.Length; i++) { listMess.Add(rows[i][0].ToString()); }

database.closeConnection();

return listMess;

}

// Метод для получения списка всех пользователей

public List<string> GetAllUsersList()

{

SqlDataAdapter adapter = new SqlDataAdapter();

Продолжение листинга ContactList

DataTable table = new DataTable();

// Строка запроса

string queryString = $"use kurs; select user\_login from users;";

SqlCommand commandBlack = new SqlCommand(queryString, database.getConnection());

// Работа с адаптером

adapter.SelectCommand = commandBlack;

adapter.Fill(table);

allUsersList.Clear();

DataRow[] rows = table.Select();

for (int i = 0; i < rows.Length; i++) { allUsersList.Add(rows[i][0].ToString()); }

database.closeConnection();

return allUsersList;

}

// Метод для получения списка контактов пользователя

public (List<string>, List<string>) GetContactLists()

{

contList = new List<string>();

// Классы, необходимые для работы с базой данных

SqlDataAdapter adapter = new SqlDataAdapter();

DataTable table = new DataTable();

// Строка запроса

string queryString = $"use kurs; select contact\_login from {login + "Contacts"};";

// Объект класса sqlCommand

SqlCommand command1 = new SqlCommand(queryString, database.getConnection());

// Работа с адаптером

adapter.SelectCommand = command1;

Продолжение листинга ContactList

adapter.Fill(table);

DataRow[] rows = table.Select();

for (int i = 0; i < rows.Length; i++) { contList.Add(rows[i][0].ToString()); }

contBlackList = new List<string>();

SqlDataAdapter adapter1 = new SqlDataAdapter();

DataTable table1 = new DataTable();

string queryBlackString = $"use kurs; select contact\_login from {login + "BlackContacts"};";

database.closeConnection();

SqlCommand commandBlack = new SqlCommand(queryBlackString, database.getConnection());

// Работа с адаптером

adapter1.SelectCommand = commandBlack;

adapter1.Fill(table1);

DataRow[] rowsBlack = table1.Select();

for (int i = 0; i < rowsBlack.Length; i++) { contBlackList.Add(rowsBlack[i][0].ToString()); }

database.closeConnection();

return (contList, contBlackList);

}

// Метод для добавления нового контакта

public string NewContact(string log, string activeLogin)

{

string actLog = $"{activeLogin}";

contList = new List<string>();

contList = GetContactLists().Item1;

allUsersList = GetAllUsersList();

int o = 0;

// Проверка на то, есть ли уже такой контакт

Продолжение листинга ContactList

for (int i = 0; i < contList.Count; i++) { if (contList[i] == log) { o = 1; } }

int h = 0;

// Проверка на то, есть ли такой пользователь

for (int i = 0; i < allUsersList.Count; i++)

{

if (allUsersList[i] == log)

{

h = 2;

}

}

if (o == 0 & h == 2)

{

// Классы, необходимые для работы с базой данных

SqlDataAdapter adapter = new SqlDataAdapter();

DataTable table = new DataTable();

// Строка запроса

string queryString = $"use kurs; insert into {actLog + "Contacts"} (contact\_login) values ('{log}');";

string queryObrString = $"use kurs; insert into {log + "Contacts"} (contact\_login) values ('{actLog}');";

string log1 = actLog;

string log2 = log;

// Сравнение того, какая строка выше по словарю.

// Чтобы создавалась таблица с корректным названием

int sravn = string.Compare(log1, log2);

if (sravn > 0)

{

string per = log1;

log1 = log2;

Продолжение листинга ContactList

log2 = per;

}

string queryString2 = $"use kurs; create table {log1 + "CHAT" + log2} (mess\_id int identity(1, 1) not null, mess varchar(80) not null, primary key(mess\_id));";

// Объект класса sqlCommand

SqlCommand command1 = new SqlCommand(queryString, database.getConnection());

// Работа с адаптером

adapter.SelectCommand = command1;

adapter.Fill(table);

database.closeConnection();

SqlCommand command2 = new SqlCommand(queryString2, database.getConnection());

SqlDataAdapter adapter2 = new SqlDataAdapter();

DataTable table2 = new DataTable();

// Работа с адаптером

adapter2.SelectCommand = command2;

adapter2.Fill(table2);

database.closeConnection();

if (log != actLog)

{

SqlCommand command3 = new SqlCommand(queryObrString, database.getConnection());

SqlDataAdapter adapter3 = new SqlDataAdapter();

DataTable table3 = new DataTable();

// Работа с адаптером

adapter3.SelectCommand = command3;

adapter3.Fill(table3);

Продолжение листинга ContactList

database.closeConnection();

}

return "OK";

}

else if (h == 0)

{

return "NOTUSERS";

}

else

{

return "NOTOK";

}

}

// Метод для удаления контакта

public string DelContact(string log, string activeLogin)

{

try

{

string actLog = $"{activeLogin}";

// Классы, необходимые для работы с базой данных

SqlDataAdapter adapter = new SqlDataAdapter();

DataTable table = new DataTable();

// Строка запроса

string queryString = $"use kurs; delete {actLog + "Contacts"} where contact\_login = '{log}';";

// Объект класса sqlCommand

SqlCommand command1 = new SqlCommand(queryString, database.getConnection());

Продолжение листинга ContactList

// Удаление контакта активного пользователя из списка контактов второго пользователя

// Работа с адаптером

adapter.SelectCommand = command1;

adapter.Fill(table);

database.closeConnection();

SqlDataAdapter adapter4 = new SqlDataAdapter();

DataTable table4 = new DataTable();

// Строка запроса

string queryString4 = $"use kurs; delete {log + "Contacts"} where contact\_login = '{actLog}';";

// Объект класса sqlCommand

SqlCommand command4 = new SqlCommand(queryString4, database.getConnection());

// Работа с адаптером

adapter4.SelectCommand = command4;

adapter4.Fill(table4);

database.closeConnection();

string log1 = actLog;

string log2 = log;

// Сравнение того, какая строка выше по словарю.

// Чтобы создавалась таблица с корректным названием

int sravn = string.Compare(log1, log2);

if (sravn > 0)

{

string per = log1;

log1 = log2;

log2 = per;

Продолжение листинга ContactList

}

string queryString2 = $"use kurs; drop table {log1 + "CHAT" + log2};";

// Работа с адаптером

adapter.SelectCommand = command1;

adapter.Fill(table);

database.closeConnection();

SqlCommand command2 = new SqlCommand(queryString2, database.getConnection());

SqlDataAdapter adapter2 = new SqlDataAdapter();

DataTable table2 = new DataTable();

// Работа с адаптером

adapter2.SelectCommand = command2;

adapter2.Fill(table2);

database.closeConnection();

return "OK";

}

catch

{

database.closeConnection();

return "NOTOK";

}

}

// Метод для добавления контакта в черный список

public string NewBlackContact(string log, string activeLogin)

{

string actLog = $"{activeLogin}";

contBlackList = new List<string>();

contBlackList = GetContactLists().Item2;

Продолжение листинга ContactList

int o = 0;

// Проверка на то, есть ли уже такой контакт

for (int i = 0; i < contBlackList.Count; i++)

{

if (contBlackList[i] == log)

{

o = 1;

}

}

if (o == 0)

{

// Классы, необходимые для работы с базой данных

SqlDataAdapter adapter = new SqlDataAdapter();

DataTable table = new DataTable();

// Строка запроса

string queryString = $"use kurs; insert into {actLog + "BlackContacts"} (contact\_login) values ('{log}');";

// Объект класса sqlCommand

SqlCommand command1 = new SqlCommand(queryString, database.getConnection());

// Работа с адаптером

adapter.SelectCommand = command1;

adapter.Fill(table);

database.closeConnection();

return "OK";

}

else

{

return "NOTOK";

Продолжение листинга ContactList

}

}

// Метод для удаления контакта из черного списка

public string DelBlackContact(string log, string activeLogin)

{

try

{

string actLog = $"{activeLogin}";

// Классы, необходимые для работы с базой данных

SqlDataAdapter adapter = new SqlDataAdapter();

DataTable table = new DataTable();

// Строка запроса

string queryString = $"use kurs; delete {actLog + "BlackContacts"} where contact\_login = '{log}';";

// Объект класса sqlCommand

SqlCommand command1 = new SqlCommand(queryString, database.getConnection());

// Работа с адаптером

adapter.SelectCommand = command1;

adapter.Fill(table);

database.closeConnection();

return "OK";

}

catch

{

database.closeConnection();

return "NOTOK"; } }}

# Приложение Г

Листинг кода класса Contacts.xaml.cs

namespace ChatClient

{

public partial class Contacts : Page

{

ServiceChatClient client;

public Contacts()

{

InitializeComponent();

client = MainWindow.client;

}

// Список контактов

private List<string> contList;

// Черный список

private List<string> contBlackList;

// Событие нажатия кнопки добавления контакта в черный список

private void newBlackButton\_Click(object sender, RoutedEventArgs e)

{

contBlackList = new List<string>();

contBlackList = new List<string>();

MessageBox.Show(client.NewBlackCont(comboBoxCont.SelectedValue.ToString()));

listBlack.Items.Clear();

contBlackList = client.GetContList().Item2.ToList();

for (int i = 0; i < contBlackList.Count; i++)

{

listBlack.Items.Add(contBlackList[i]);

}

Продолжение листинга Contacts.xaml.cs

}

// Событие нажатия кнопки добавления контакта

private void newContactButton\_Click(object sender, RoutedEventArgs e)

{

contList = new List<string>();

MessageBox.Show(client.NewCont(newContBox.Text));

listContacts.Items.Clear();

comboBoxCont.Items.Clear();

contList = client.GetContList().Item1.ToList();

for (int i = 0; i < contList.Count; i++)

{

listContacts.Items.Add(contList[i]);

comboBoxCont.Items.Add(contList[i]);

}

}

// Функция обновления чата

public void updateChat()

{

// Проверка на то, не пуст ли выбор контакта

if (comboBoxCont.SelectedValue != null )

{

// Проверка на то, находится ли контакт в черном списке

int j = 0;

for (int i = 0; i < contBlackList.Count; i++)

{

if (comboBoxCont.SelectedValue.ToString() == contBlackList[i]) j = 1;

}

if (j == 0)

{

Продолжение листинга Contacts.xaml.cs

try

{

List<string> messList = new List<string>();

messListBox.ItemsSource = messList;

string comb = comboBoxCont.SelectedValue.ToString();

int k1 = 0;

contBlackList.Clear();

contBlackList = client.GetContList().Item2.ToList();

for (int i = 0; i < contBlackList.Count; i++)

{

if (contBlackList[i] == comb)

{

k1 = 1;

}

}

if (k1 == 1)

{

MessageBox.Show("Контакт в черном списке. ");

return;

}

else

{

messList.Clear();

messList = client.GetChatMesList(comb).ToList();

}

messListBox.ItemsSource = messList;

}

catch { }

}

Продолжение листинга Contacts.xaml.cs

}

}

// Событие нажатия кнопки открытия чата

private void openChatButton\_Click(object sender, RoutedEventArgs e)

{

updateChat();

}

// Событие загрузки страницы - вывод актуальных данных

private void Grid\_Loaded(object sender, RoutedEventArgs e)

{

activeUserTextBlock.Text = $"Аккаунт: \_{client.get\_active\_login()}\_";

listContacts.Items.Clear();

listBlack.Items.Clear();

comboBoxCont.Items.Clear();

contList = new List<string>();

contBlackList = new List<string>();

contList = client.GetContList().Item1.ToList();

contBlackList = client.GetContList().Item2.ToList();

for (int i = 0; i < contBlackList.Count; i++)

{

listBlack.Items.Add(contBlackList[i]);

}

for (int i = 0; i < contList.Count; i++)

{

listContacts.Items.Add(contList[i]);

comboBoxCont.Items.Add(contList[i]);

}

}

// Событие нажатия кнопки выхода из аккаунта

Продолжение листинга Contacts.xaml.cs

private void backButton\_Click(object sender, RoutedEventArgs e)

{

// Переход к Main

NavigationService.Navigate(MainWindow.listPage[0]);

MainWindow.active\_user\_email = null;

MainWindow.active\_user\_login = null;

}

// Событие нажатяи кнопки перехода в меню

private void menuButton\_Click(object sender, RoutedEventArgs e)

{

NavigationService.Navigate(MainWindow.listPage[4]);

}

// Событие нажатия кнопки удаление контакта из черного списка

private void DelBlackButton\_Click(object sender, RoutedEventArgs e)

{

string comb = comboBoxCont.SelectedValue.ToString();

MessageBox.Show(client.DelBlackCont(comb));

listBlack.Items.Clear();

contBlackList = client.GetContList().Item2.ToList();

for (int i = 0; i < contBlackList.Count; i++)

{

listBlack.Items.Add(contBlackList[i]);

}

}

// Событие нажатия кнопки удаления аккаунта

private void DelButton\_Click(object sender, RoutedEventArgs e)

{

string comb = comboBoxCont.SelectedValue.ToString();

MessageBox.Show(client.DelCont(comb));

Продолжение листинга Contacts.xaml.cs

listContacts.Items.Clear();

comboBoxCont.Items.Clear();

contList = client.GetContList().Item1.ToList();

for (int i = 0; i < contList.Count; i++)

{

listContacts.Items.Add(contList[i]);

comboBoxCont.Items.Add(contList[i]);

}

}

// Событие нажатия кнопки отправки сообщения

private void newMessChatButton\_Click(object sender, RoutedEventArgs e)

{

string newMessage = $" {newMessageBlock.Text}";

string comb1 = comboBoxCont.SelectedValue.ToString();

string status = client.NewMessChat(comb1, newMessage);

List<string> listStr = new List<string>();

messListBox.ItemsSource = listStr;

string comb = comboBoxCont.SelectedValue.ToString();

int k1 = 0;

contBlackList.Clear();

contBlackList = client.GetContList().Item2.ToList();

for (int i = 0; i < contBlackList.Count; i++) { if (contBlackList[i] == comb) { k1 = 1;} }

if (k1 == 1)

{

MessageBox.Show("Контакт в черном списке. Вы не можете открыть этот чат");

}

Else

Продолжение листинга Contacts.xaml.cs

{

List<string> messList = new List<string>();

messList = client.GetChatMesList(comb).ToList();

messListBox.ItemsSource = messList;

}

}

// Событие движения мыши - для обновления данных в списке сообщений чата

private void Grid\_MouseMove(object sender, MouseEventArgs e)

{

listContacts.Items.Clear();

updateChat();

contList = new List<string>();

contList = client.GetContList().Item1.ToList();

for (int i = 0; i < contList.Count; i++)

{

listContacts.Items.Add(contList[i]);

}

}

}

}