Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра программного обеспечения информационных технологий

Дисциплина: Компьютерные системы и сети

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовой работе

на тему

ПРИЛОЖЕНИЕ ДЛЯ ВИДЕО-ЗВОНКОВ

БГУИР КР 6 - 05 - 06 12 01 013 ПЗ

Выполнил: студент группы 310901 Чиж А. В.

Проверила: Болтак С. В.

Минск 2025

СОДЕРЖАНИЕ

[**ВВЕДЕНИЕ** 3](#_Toc199280227)

[**1** **АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ** 4](#_Toc199280228)

[1.1 Обзор аналогов 4](#_Toc199280229)

[1.2 Постановка задачи 6](#_Toc199280230)

[**2** **ПРОЕКТИРОВАНИЕ ЗАДАЧИ**. 7](#_Toc199280231)

[2.1 Структура программы 7](#_Toc199280232)

[2.2 Проектирование интерфейса программного средства 9](#_Toc199280233)

[**2.2.1** **Подключение к видеозвонку** 10](#_Toc199280234)

[**2.2.2** **Основной процесс видеозвонка** 10](#_Toc199280235)

[2.3 Проектирование функционала программного средства 10](#_Toc199280236)

[**3** **РАЗРАБОТКА ПРОГРАММНОГО СРЕДСТВА** 12](#_Toc199280237)

[3.1 Архитектура взаимодействия клиентов и сервера 12](#_Toc199280238)

[**3.2** **Процесс установки соединения** 12](#_Toc199280239)

[3.3 Управление медиапотоками 13](#_Toc199280240)

[3.4 Завершение сеанса связи 13](#_Toc199280241)

[**4** **ТЕСТИРОВАНИЕ** 15](#_Toc199280242)

[**5** **ПРИМЕНЕНИЕ ПРОГРАММЫ** 16](#_Toc199280243)

[**ЗАКЛЮЧЕНИЕ** 20](#_Toc199280244)

[**СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ** 21](#_Toc199280245)

[**ПРИЛОЖЕНИЕ А** 22](#_Toc199280246)

# **ВВЕДЕНИЕ**

Интернет и компьютерные технологии прочно вошли в современную жизнь, став неотъемлемой частью работы, обучения и личного общения. Особую важность приобрели приложения для видеосвязи, позволяющие людям видеть и слышать друг друга в реальном времени независимо от расстояния. Эти технологии кардинально изменили подход к дистанционному взаимодействию, сделав его более естественным и эффективным.

Разработка приложений для видео-звонков представляет собой сложную техническую задачу, требующую глубоких знаний в области сетевых технологий, обработки мультимедиа и создания пользовательских интерфейсов. Современные решения должны обеспечивать стабильное соединение, минимальные задержки и высокое качество передачи аудио- и видеопотоков при работе в различных сетевых условиях.

В последние годы наблюдается устойчивый рост спроса на надежные и функциональные решения для видеоконференций. Такие платформы, как *Zoom, Microsoft Teams* и *Google Meet*, стали стандартом де-факто, демонстрируя важность удобных и производительных инструментов для удаленного взаимодействия.

Данная курсовая работа посвящена разработке приложения для видео-звонков с использованием языка C# и технологии *WPF* (*Windows Presentation Foundation*). Выбор этой платформы обусловлен ее мощными возможностями для создания современных пользовательских интерфейсов, высокой производительностью и удобством интеграции с мультимедийными библиотеками *Windows*. Для реализации сетевого взаимодействия и обработки видеопотоков используются технологии и библиотеки .*NET*.

Основной целью проекта является создание стабильно работающего приложения для видеосвязи, поддерживающего индивидуальные и групповые вызовы.

Актуальность разработки подтверждается растущей потребностью в качественных решениях для дистанционного общения, особенно в сферах бизнеса, образования и личных коммуникаций. Реализованное приложение продемонстрирует возможности платформы .*NET* и технологии *WPF* для создания современных мультимедийных приложений.

# **АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ**

## **Обзор аналогов**

В настоящее время на рынке представлено множество приложений для видеосвязи, каждое из которых предлагает базовый набор функций: видеозвонки, текстовый чат, управление медиапотоками и создание конференций. Рассмотрим два популярных решения **—***Zoom* и *Microsoft Teams*.

*Zoom* — одна из самых распространенных платформ для видеоконференций, сочетающая простоту использования с широкими функциональными возможностями. Интерфейс приложения представлен на рисунке 1.1.



Рисунок 1.1 – Интерфейс приложения 1.1

Ключевые особенности Zoom:

* Групповые видеозвонки – поддержка до 1000 участников (в зависимости от тарифного плана) с возможностью разделения на комнаты (Breakout Rooms).
* Высокое качество связи – адаптивная настройка качества видео в зависимости от скорости интернета.
* Дополнительные инструменты – демонстрация экрана, виртуальный фон, запись звонков, текстовый чат с возможностью отправки файлов.
* Кроссплатформенность – доступ на ПК (*Windows, macOS*), мобильных устройствах (*iOS, Android*) и через веб-браузер.
* Интеграция с календарями – синхронизация с *Google Calendar* и *Outlook* для удобного планирования встреч.

*Zoom* также предоставляет расширенные настройки безопасности: шифрование данных, контроль доступа к конференциям, модерацию участников. Приложение активно используется в бизнесе, образовании и для личного общения.

*Microsoft Teams* — корпоративное решение для видеосвязи, входящее в экосистему Microsoft 365. Интерфейс приложения показан на рисунке 1.2.



Рисунок 1.2 – Интерфейс приложения *Microsoft Teams*

Основные возможности *Microsoft Teams*:

* Гибкие форматы конференций – поддержка аудио- и видеозвонков, вебинаров и трансляций.
* Глубокая интеграция с *Office 365* – совместная работа над документами *Word, Excel, PowerPoint* прямо во время звонка.
* Функции для командной работы – каналы, задачи, облачное хранилище (*OneDrive, SharePoint*).
* Искусственный интеллект – фоновое размытие, подавление шума, автоматическая транскрипция.
* Безопасность – двухфакторная аутентификация, защита данных в соответствии с корпоративными стандартами.

Teams ориентирован в первую очередь на бизнес-сегмент, но также подходит для образовательных учреждений благодаря инструментам для дистанционного обучения.

## **Постановка задачи**

В рамках данной курсовой работы необходимо разработать приложение для видеозвонков с использованием *C*# и *WPF*. В перечень ключевых задач входит:

* Изучить принципы работы современных систем видеосвязи, включая передачу аудио- и видеопотоков в реальном времени.
* Провести анализ существующих решений (*Zoom, Microsoft Teams* и др.) и выделить их основные функциональные возможности.
* Разработать удобный и интуитивно понятный пользовательский интерфейс.
* Сформулировать функциональные требования к приложению.

Основные функциональные требования

Приложение должно обеспечивать следующие базовые возможности:

* Установка видеосоединения между двумя и более участниками.
* Передача аудио и видео в реальном времени.
* Стабильное сетевое взаимодействие с минимальными задержками.

Приложение должно включать следующие экраны и элементы интерфейса:

* Главное окно с возможностью начала/приема вызова.
* Окно видеозвонка, отображающее видеопотоки участников.

Приложение подразумевает использование сервера для маршрутизации вызовов реализующее корректное координирование клиентов между собой.

# **ПРОЕКТИРОВАНИЕ ЗАДАЧИ**.

## **Структура программы**

Модуль *Models* содержит класс *User*, который представляет данные пользователя, включая идентификатор, имя и статус подключения, а также предоставляет методы для управления соединением. Также содержит класс *Contact*, который представляет данные о контактах пользователя. Модуль Services объединяет несколько специализированных классов: *NetworkService* отвечает за сетевое взаимодействие и управление звонками, *AuthService* обеспечивает процессы аутентификации и авторизации, *AudioService* и *MicrophoneService* контролируют работу с аудиопотоками, включая регулировку громкости и управление микрофоном, а *CameraService* занимается обработкой видеопотока, предоставляя функции включения/выключения камеры и переключения между устройствами. Модуль *Views* включает два основных графических интерфейса: *LoginWindow* для ввода учетных данных и *MainWindow* - основное окно видеозвонка с элементами управления медиапотоками и интерфейсом чата. Все модули тесно взаимодействуют: после успешной аутентификации через *LoginWindow* и *AuthService*, *MainWindow* использует *NetworkService* для установки соединения и совместно с медиа-сервисами организует процесс видеосвязи, обеспечивая полный цикл работы приложения от входа пользователя до проведения видеозвонка. Методы и поля классов, описанных выше, представлены в таблице 2.1.

Таблица 2.1

|  |  |  |
| --- | --- | --- |
| 1.Название | 2. Поля | 3. Методы |
| *User* | *Guid Id,*  *string Username,*  *string PasswordHash,*  *DateTime CreatedAt,*  *DateTime? LastOnline,*  *bool IsActive* |  |
| *Contact* | *Guid Id,*  *Guid UserId,*  *Guid ContactId,*  *DateTime CreatedAt* |  |

Продолжение таблицы 2.1

|  |  |  |
| --- | --- | --- |
| 1. | 2. | 3. |
| *AuthService* | *Client \_supabase* |  |
|  | *VideoCaptureDevice \_videoSource,*  *bool \_isRunning,*  *NetworkService \_networkService* | *Register(),*  *Login(),*  *Logout(),*  *HashPassword()* |
| *CameraService* | *VideoCaptureDevice \_videoSource,*  *bool \_isRunning,*  *NetworkService \_networkService* | *StartCamera(),*  *SetNetworkTarget(),*  *OnNewFrame(),*  *ConvertBitmapToBitmapImage(),*  *StopCamera()* |
| *MicrophoneService* | *WaveInEvent \_waveIn,*  *bool \_isRecording,*  *WasapiCapture \_audioCapture,*  *NetworkService \_networkService,*  *bool \_isRunning* | *SetNetworkTarget(),*  *StartCapture(),*  *OnAudioData()* |
| *AudioService* | *WaveOutEvent \_waveOut,*  *BufferedWaveProvider \_waveProvider,*  *ConcurrentQueue \_audioQueue,*  *bool \_isPlaying* | *StartPlayback(),*  *StopPlayback(),*  *EnqueueAudioData(),*  *ProcessAudioQueue()* |
| *NetworkService* | *TcpClient \_tcpClient,*  *UdpClient \_udpClient,*  *string \_serverIp,*  *int \_serverPort,*  *int \_udpPort,*  *VideoCodec \_videoCodec,*  *AudioCodec \_audioCodec* | *ConnectAsync(),*  *UpdateClientAsync(),*  *InitiateCallAsync(),*  *HandleIncomingCalls(),*  *InitializeUdpClient(),*  *ListenForUdpData(),*  *ProcessIncomingData(),*  *StartStreaming(),*  *StopStreaming(),*  *StartReceiving(),*  *ProcessIncomingPacket()* |

Продолжение таблицы 2.1

|  |  |  |
| --- | --- | --- |
| 1. | 2. | 3. |
| *LoginWindow* | *AuthService \_authService* | *LoginButton\_Click(),*  *RegisterButton\_Click()* |
| *MainWindow* | *Guid \_currentCallId,*  *NetworkService \_networkService,*  *ObservableCollection \_contacts,*  *User \_selectedUser* | *LoadContacts(),*  *DisplayUserProfile(),*  *ShowCallUI(),*  *CallButton\_Click(),*  *StartNewCall(),*  *AcceptCallButton\_Click(),*  *RejectCallButton\_Click(),*  *EndCallButton\_Click(),*  *StartMediaDevices(),*  *EndCall()* |

## **Проектирование интерфейса программного средства**

Разработка пользовательского интерфейса приложения для видеозвонков осуществляется с использованием технологии *Windows Presentation Foundation (WPF*), которая является частью платформы .*NET*. *WPF* предоставляет мощные инструменты для создания современных, визуально привлекательных и функциональных интерфейсов с поддержкой сложной графики, анимации и стилизации элементов.

При проектировании интерфейса особое внимание уделяется следующим аспектам:

1. Минималистичный и интуитивно понятный дизайн, позволяющий пользователям быстро освоить основные функции приложения без дополнительного обучения. Все ключевые элементы управления (кнопки вызова, настройки микрофона и камеры) расположены в зоне легкой доступности.
2. Адаптивная верстка, обеспечивающая корректное отображение интерфейса на экранах различных размеров и разрешений. Используются механизмы WPF для автоматического масштабирования и перераспределения элементов управления.
3. Использование шаблонов *MVVM (Model-View-ViewModel*), что позволяет четко разделить логику приложения и его визуальное представление. Это упрощает поддержку и дальнейшее развитие кода.
4. Стилизация элементов через *XAML*-разметку.
5. Интеграция с системными *API* для управления мультимедийными устройствами (камерами, микрофонами) и сетевыми функциями.

### **Подключение к видеозвонку**

Процесс подключения пользователя к видеозвонку состоит из нескольких этапов. При запуске приложения пользователь видит окно авторизации (Рисунок 2.1), где необходимо ввести свое имя и идентификатор видеоконференции. Интерфейс содержит:

* Поле для ввода имени пользователя
* Поле для указания *ID* конференции
* Кнопку "Подключиться"

### **Основной процесс видеозвонка**

После старта конференции открывается основное окно видеозвонка, содержащее:

1. Область отображения видеоучастников
2. Панель управления с кнопкой завершения звонка

**2.2.3 Завершение звонка**

При завершении видеозвонка:

1. Появляется подтверждающее окно
2. Приложение возвращается в начальное состояние

## **Проектирование функционала программного средства**

Эффективное проектирование программного обеспечения требует четкого определения функциональных требований и разработки оптимальных алгоритмов их реализации. Для приложения видеозвонков был определен следующий базовый функционал:

1. Авторизация пользователя
2. Управление медиапотоками (аудио/видео)
3. Добавления новых контактов

**2.3.1 Авторизация пользователя**

Процесс авторизации в системе видеозвонков активируется при первом запуске приложения или при истечении срока действия текущей сессии. Алгоритм включает:

* Проверку сохраненных учетных данных в локальном хранилище
* Ввод логина и пароля при отсутствии валидных данных
* Аутентификацию через сервер *Supabase*
* Инициализацию пользовательского профиля

При успешной авторизации:

* Загружается список контактов пользователя
* Обновляется статус подключения (онлайн)

**2.3.2 Установка видеосоединения**

Процесс установки соединения активируется при нажатии кнопки "Начать звонок" или автоматически при получении входящего вызова. Алгоритм включает:

* Проверку доступности сетевого подключения
* Инициализацию медиаустройств (камера, микрофон)
* Установку соединения через сервер сигнализации
* Создание каналов передачи данных

При успешном соединении:

* Активируются потоки видео и аудио
* Обновляется интерфейс (отображаются участники)
* Запускается мониторинг качества соединения

**2.3.3 Добавление новых контактов**

Процесс добавления контактов в адресную книгу осуществляется через интерфейс приложения. Алгоритм включает:

* Отправку запроса на добавление в контакты
* Синхронизацию изменений между устройствами

При успешном добавлении:

* Контакт появляется в общем списке
* Активируются уведомления об изменении статуса

# **РАЗРАБОТКА ПРОГРАММНОГО СРЕДСТВА**

## **Архитектура взаимодействия клиентов и сервера**

В основе системы видеозвонков лежит распределенная клиент-серверная архитектура, где центральным элементом выступает серверное приложение, выполняющее функции координатора соединений. Сервер реализован на технологии *ASP.NET Core* и использует современный стек технологий для обеспечения надежной и масштабируемой работы. Для хранения данных система интегрирована с облачной базой данных *Supabase*, которая предоставляет полный набор функциональности включая аутентификацию пользователей, хранение профилей и журналирование сеансов связи. Взаимодействие между клиентами и сервером организовано через два основных канала связи - *TCP* и *UDP* соединения*. TCP* используется для выполнения операций аутентификации, управления профилями пользователей и получения метаданных о текущих сеансах связи. *UDP* соединения служат для реального обмена данными между участниками видеозвонка и обеспечивают минимально возможную задержку при передаче служебной информации. Сервер поддерживает постоянное соединение с каждым активным клиентом через *UDP*, что позволяет мгновенно уведомлять пользователей о входящих вызовах, изменениях в составе участников и других значимых событиях.

Ключевой функцией сервера является координация клиентов для установления *P2P*-соединений между ними. Сервер выступает в роли *ICE*-агента, собирая кандидаты от всех участников звонка и координируя процесс установления оптимального маршрута соединения.

### **Процесс установки соединения**

Процедура установки видеосоединения представляет собой многоэтапный процесс, начинающийся с инициализации клиентского приложения. При запуске приложение автоматически устанавливает фоновое *TCP*-соединение с сервером. Это соединение остается активным на протяжении всего сеанса работы пользователя и служит основным каналом для оперативного обмена служебными сообщениями. Параллельно клиент выполняет аутентификацию через интегрированный *Supabase* *Auth*, передавая логин и пароль.

Серверная часть, получив запрос на инициацию видеозвонка, выполняет комплексную проверку параметров вызова. Сначала анализируется список приглашенного участника - сервер проверяет его доступность через *Supabase*, учитывая не только текущий статус (онлайн/оффлайн).

При инициации видеозвонка инициатор выбирает участников из списка контактов, после чего клиентская система отправляет на сервер детали предстоящего сеанса. Сервер создает соответствующую запись в базе данных, рассылает уведомления выбранным участникам и возвращает инициатору технические данные, необходимые для установки прямых *P2P*-соединений между всеми участниками видеоконференции.

Процесс завершения соединения также тщательно оптимизирован. При получении сигнала о завершении звонка сервер инициирует плавное прекращение медиапотоков с сохранением служебного канала связи еще на 15-30 секунд. Это позволяет моментально восстановить соединение, если пользователь случайно завершил вызов. Все статистические данные о продолжительности звонка, качестве соединения и использованных кодеках сохраняются в Supabase для последующего анализа и улучшения работы системы.

## **Управление медиапотоками**

Система видеозвонков реализует комплексный подход к управлению медиапотоками, обеспечивающий стабильную работу даже в условиях нестабильных сетевых соединений. На уровне клиентского приложения реализован многослойный механизм захвата и обработки медиаданных, начинающийся с оптимизированного доступа к аппаратным устройствам. Для видеопотока применяется интеллектуальный алгоритм адаптивного захвата, который динамически регулирует разрешение, частоту кадров и параметры кодирования в зависимости от вычислительных возможностей устройства и текущей сетевой нагрузки.

Транспортный уровень системы построен на реализации *UDP*. Для видеокодирования используется адаптивная кодековая система, автоматически выбирающая между *VP8*, *VP9* и *H*.*264* в зависимости от возможностей устройств участников и текущих сетевых условий.

## **Завершение сеанса связи**

Процедура завершения видеозвонка инициируется клиентом и обрабатывается сервером по строгому протоколу. Серверная часть сначала обновляет статус сеанса в базе данных, сохраняя метаинформацию о продолжительности звонка, затем уведомляет всех участников о завершении сессии и освобождает задействованные сетевые ресурсы. Клиентские приложения получают подтверждение о корректном завершении сеанса и закрывают установленные соединения.

# **ТЕСТИРОВАНИЕ**

В ходе тестирования приложения для видеозвонков была обнаружена серьезная проблема с утечкой ресурсов камеры - после завершения звонка камера продолжала работать в фоновом режиме, что приводило к повышенному энергопотреблению и потенциальным проблемам с конфиденциальностью. Анализ показал, что проблема заключалась в отсутствии корректного освобождения ресурсов при завершении сеанса связи.

Основная ошибка была в классе *VideoService*, который управлял работой камеры, но не реализовывал интерфейс *IDisposable* и не содержал явных методов освобождения ресурсов.

Решение обеспечивает:

1. Корректное освобождение ресурсов камеры при завершении звонка
2. Защиту от утечек памяти
3. Возможность повторного использования камеры
4. Автоматическое освобождение ресурсов даже при аварийном завершении

Дополнительно были добавлены проверки состояния камеры при повторной инициализации и логирование всех операций с видеоустройством для упрощения диагностики подобных проблем в будущем. Тестирование показало, что после внесения изменений проблема с "вечной" работой камеры полностью устранена.

# **ПРИМЕНЕНИЕ ПРОГРАММЫ**

Окно авторизации представляет собой первое окно, которое видит пользователь при запуске приложения. Оно содержит следующие элементы:

* заголовок «Видеочат»;
* поле ввода имени пользователя;
* поле ввода пароля;
* кнопку входа в систему;
* кнопку перехода на окно регистрации.

Внешний вид окна авторизации представлен на рисунке 5.1.

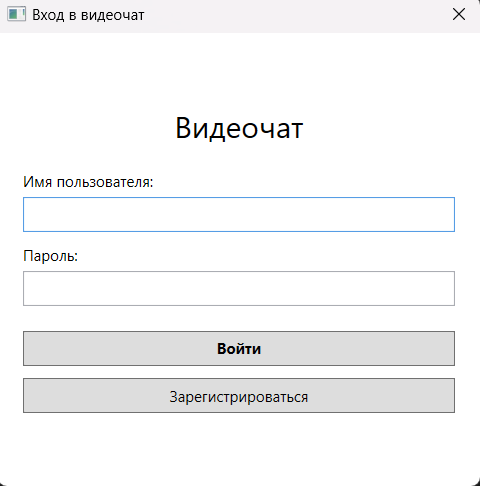


Рисунок 5.1 – Окно авторизации

Главное окно видеочата разделено на две основные части:

* левая часть — список пользователей и панель поиска;
* правая часть — область отображения информации о выбранном пользователе.

В левой части находится поле для поиска пользователей и список доступных собеседников. При выборе пользователя в правой части отображается его имя, аватар и история предыдущих звонков, если таковые имеются.

Также в нижней части правой панели находится кнопка «Позвонить», предназначенная для начала видеовызова.

Внешний вид главного окна представлен на рисунке 5.2.

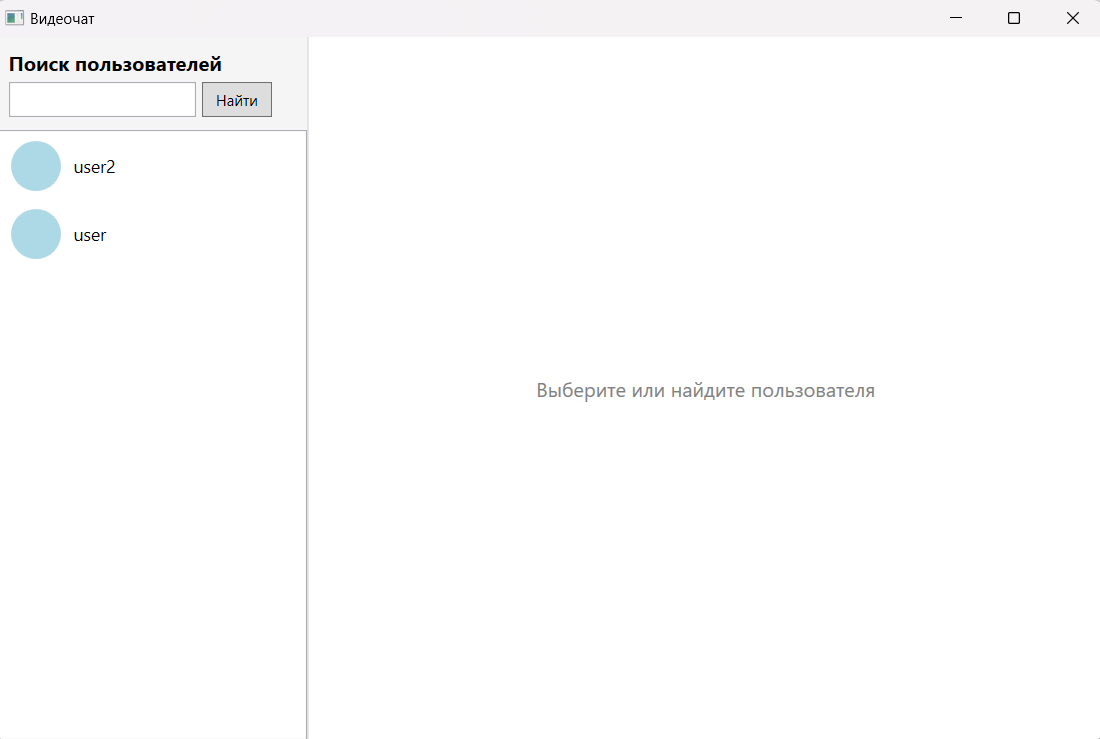


Рисунок 5.2 – Главное окно

После выбора пользователя из списка, отображается окно с информацией о нем:

* имя пользователя;
* аватар;
* кнопка для добавления в контакты (если не добавлен ранее);
* кнопка для начала звонка.

Это окно позволяет пользователю быстро получить информацию о предыдущих взаимодействиях и при необходимости повторно связаться с собеседником. Внешний вид окна профиля представлен на рисунке 5.3.

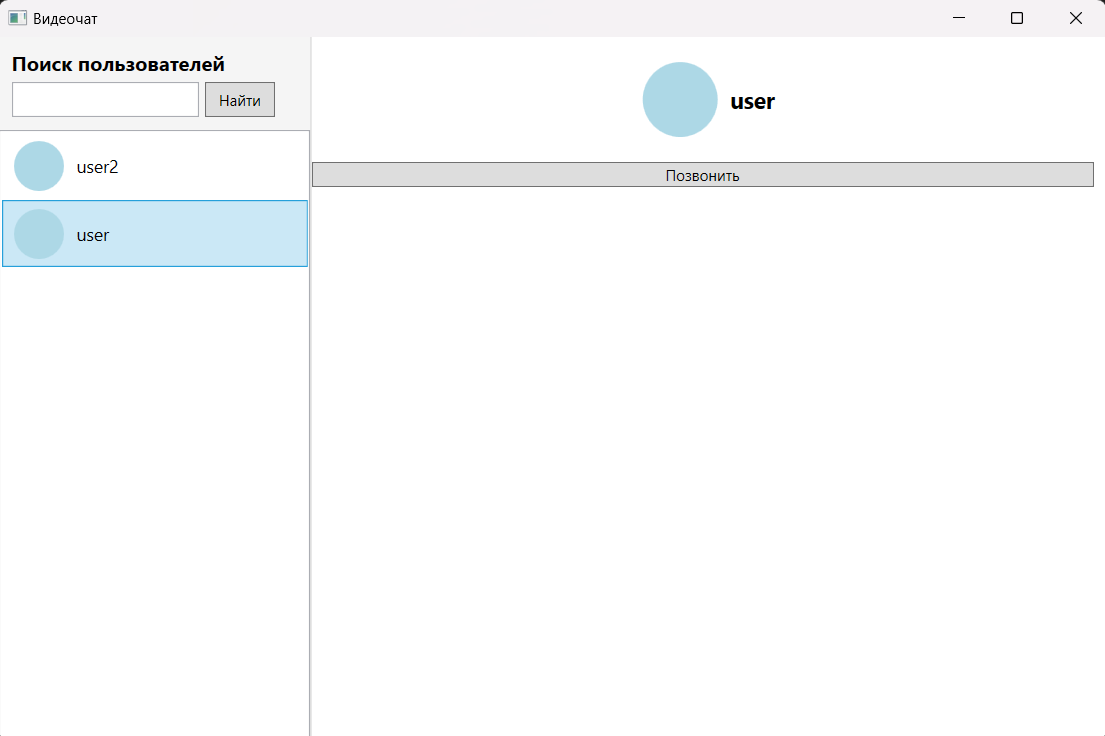


Рисунок 5.3 – Окно профиля

Окно видеозвонка активируется после успешного соединения с выбранным пользователем. Оно представляет собой полноэкранное окно, в котором основной элемент интерфейса — это видеопоток с камеры собеседника, отображаемый на весь экран.

В правом нижнем углу отображается уменьшенное окно с изображением, передаваемым с камеры текущего пользователя. Таким образом обеспечивается визуальная обратная связь и контроль за собственным изображением.

В нижней центральной части окна расположена кнопка «Завершить звонок», выделенная контрастным цветом для быстрого доступа. Нажатие на неё приводит к разрыву соединения и возврату в главное окно приложения.

Внешний вид окна видеозвонка представлен на рисунке 5.4.

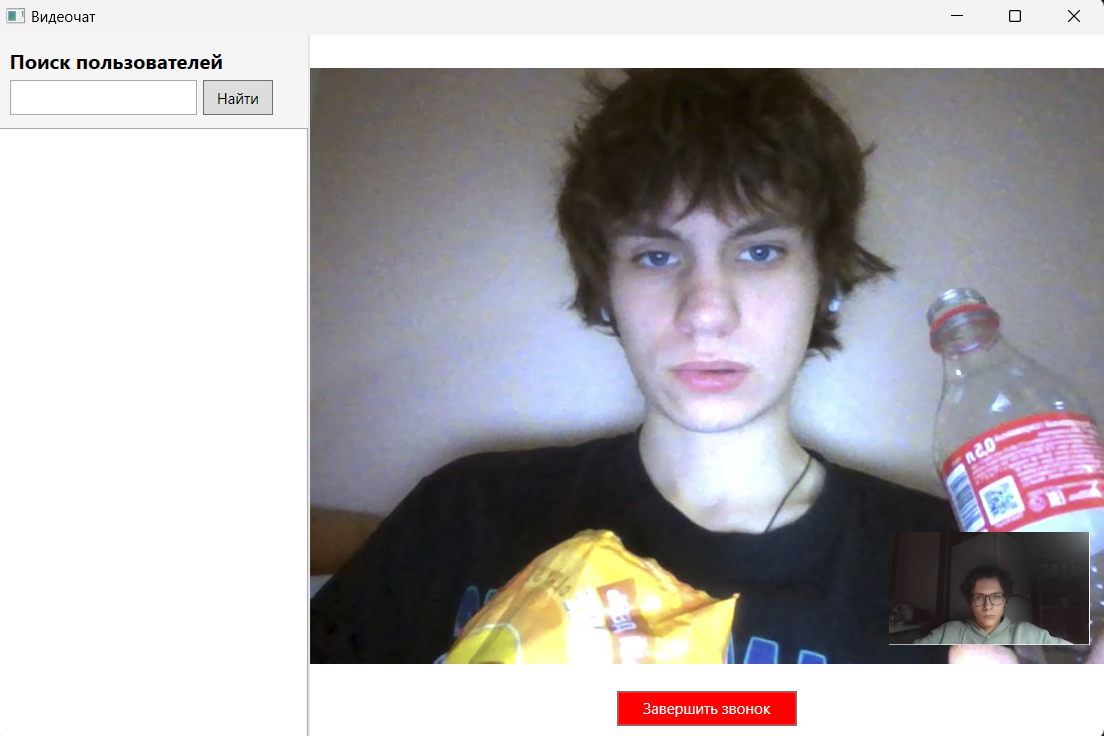


Рисунок 5.4 – Окно видеозвонка

# **ЗАКЛЮЧЕНИЕ**

В ходе выполнения курсовой работы по созданию приложения для видеозвонков было разработано программное решение, предоставляющее пользователям возможность совершать индивидуальные и групповые видеовызовы с высоким качеством связи. Разработанное приложение сочетает в себе простоту использования с мощным функционалом, обеспечивая комфортное общение как для частных пользователей, так и для бизнес-коммуникаций.

При сравнении с популярными решениями (*Zoom*, *Microsoft* *Teams*) разработанное приложение демонстрирует ряд конкурентных преимуществ:

1. В отличие от *Zoom*, где базовый функционал ограничен 40 минутами в бесплатной версии, наше решение предлагает неограниченную продолжительность звонков
2. По сравнению с *Microsoft* *Teams*, требующим корпоративной учетной записи для полного доступа к функциям, наше приложение предоставляет все возможности без подписки
3. В отличие от обоих сервисов, наше решение не навязывает дополнительных платных услуг и имеет прозрачную бизнес-модель

Технические особенности реализации:

Практическая значимость проекта заключается в демонстрации возможности создания полнофункционального решения для видеосвязи с открытым исходным кодом, которое может быть адаптировано под конкретные потребности пользователей. Разработанное приложение показало, что современные технологии позволяют создавать конкурентоспособные продукты в области видеокоммуникаций без использования проприетарных решений.

Проект подтвердил, что даже в условиях высокой конкуренции на рынке видеокоммуникаций возможно создание оригинальных решений, сочетающих в себе простоту использования, богатый функционал и уважение к приватности пользователей.

# **СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

[1] Стандарт предприятия. Разработка программного обеспечения. СТП 02-2024 / Петров И.А., Сидоров В.К. – Минск : БГУИР, 2024. – 215 с.

[2] WebRTC: официальная документация [Электронный ресурс]. – Режим доступа: <https://webrtc.org/>

[3] Supabase: платформа разработки [Электронный ресурс]. – Режим доступа: <https://supabase.com/>

[4] Microsoft .NET Framework [Электронный ресурс]. – Режим доступа: <https://dotnet.microsoft.com/>

[5] Иванов, А.В. Принципы разработки сетевых приложений на C# / Современные информационные технологии: сборник материалов 60-й научной конференции БГУИР, Минск, 15-19 апреля 2024 г. / Иванов А.В. – Минск, 2024. – С. 112-115.

[6] Visual Studio Code [Электронный ресурс]. – Режим доступа: <https://code.visualstudio.com/>

[7] Разработка клиент-серверных приложений на C#: учеб.-метод. пособие / С.П. Козлов. – Минск: БГУИР, 2021. – 78 с.: ил.

[8] Windows Presentation Foundation: практическое руководство / Е.Н. Волкова. – Минск: БГУИР, 2019. – 95 с.: ил.

[9] Postman API Platform [Электронный ресурс]. – Режим доступа: <https://www.postman.com/>

[10] Figma: инструмент проектирования интерфейсов [Электронный ресурс]. – Режим доступа: <https://www.figma.com/>

[11] GitHub: платформа для разработки ПО [Электронный ресурс]. – Режим доступа: <https://github.com/>

[12] Смит, Дж. Проектирование пользовательских интерфейсов / Дж. Смит. – М.: Техносфера, 2023. – 320 с.

# **ПРИЛОЖЕНИЕ А**

**(обязательное)**

**Листинг кода**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Data;

using System.Windows.Documents;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Media.Imaging;

using System.Windows.Shapes;

using VideoChat\_Client.Services;

using Supabase;

using DotNetEnv;

namespace VideoChat\_Client.Views

{

public partial class LoginWindow : Window

{

private readonly AuthService \_authService;

public LoginWindow()

{

InitializeComponent();

Env.TraversePath().Load();

string SupabaseUrl = Environment.GetEnvironmentVariable("SUPABASE\_URL")!;

string SupabaseKey = Environment.GetEnvironmentVariable("SUPABASE\_KEY")!;

var supabase = new Client(

SupabaseUrl,

SupabaseKey);

\_authService = new AuthService(supabase);

}

private async void LoginButton\_Click(object sender, RoutedEventArgs e)

{

var username = UsernameBox.Text.Trim();

var password = PasswordBox.Password;

if (string.IsNullOrEmpty(username) || string.IsNullOrEmpty(password))

{

ShowError("Имя пользователя и пароль обязательны");

return;

}

var user = await \_authService.Login(username, password);

if (user != null)

{

App.CurrentUser = user;

var mainWindow = new MainWindow();

mainWindow.Show();

this.Close();

}

else

{

ShowError("Неверное имя пользователя или пароль");

}

}

private async void RegisterButton\_Click(object sender, RoutedEventArgs e)

{

var username = UsernameBox.Text.Trim();

var password = PasswordBox.Password;

if (string.IsNullOrEmpty(username) || string.IsNullOrEmpty(password))

{

ShowError("Имя пользователя и пароль обязательны");

return;

}

if (password.Length < 4)

{

ShowError("Пароль должен быть не менее 4 символов");

return;

}

var success = await \_authService.Register(username, password);

if (success)

{

LoginButton\_Click(sender, e);

}

else

{

ShowError("Имя пользователя уже занято");

}

}

private void ShowError(string message, bool isError = true)

{

MessageText.Text = message;

MessageText.Foreground = isError ? Brushes.Red : Brushes.Green;

MessageText.Visibility = Visibility.Visible;

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using VideoChat\_Client.Models;

using System.Security.Cryptography;

using Supabase;

using System.Diagnostics;

namespace VideoChat\_Client.Services

{

class AuthService

{

private readonly Client \_supabase;

public AuthService(Client supabase)

{

\_supabase = supabase;

}

public async Task<bool> Register(string username, string password)

{

var existingUser = await \_supabase

.From<Models.User>()

.Where(x => x.Username == username)

.Single();

if (existingUser != null)

return false;

var passwordHash = HashPassword(password);

var newUser = new Models.User

{

Username = username,

PasswordHash = passwordHash

};

var response = await \_supabase

.From<Models.User>()

.Insert(newUser);

return response.ResponseMessage.IsSuccessStatusCode;

}

public async Task<Models.User?> Login(string username, string password)

{

var user = await \_supabase

.From<Models.User>()

.Where(x => x.Username == username)

.Single();

if (user == null)

return null;

var inputHash = HashPassword(password);

if (inputHash != user.PasswordHash)

return null;

await \_supabase

.From<Models.User>()

.Where(x => x.Id == user.Id)

.Set(x => x.IsActive, true)

.Update();

var updatedUser = await \_supabase

.From<Models.User>()

.Where(x => x.Id == user.Id)

.Single();

return updatedUser;

}

public async Task<bool> Logout(Guid userId)

{

try

{

var response = await \_supabase

.From<Models.User>()

.Where(x => x.Id == userId)

.Set(x => x.IsActive, false)

.Set(x => x.LastOnline, DateTime.UtcNow)

.Update();

return response.ResponseMessage.IsSuccessStatusCode;

}

catch (OperationCanceledException)

{

Debug.WriteLine("Logout timeout exceeded");

return false;

}

catch (Exception ex)

{

Debug.WriteLine($"Logout error: {ex.Message}");

return false;

}

}

private string HashPassword(string password)

{

using var sha256 = SHA256.Create();

var saltedPassword = password;

var bytes = sha256.ComputeHash(Encoding.UTF8.GetBytes(saltedPassword));

return Convert.ToBase64String(bytes);

}

}

}

using System.Text;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Data;

using System.Windows.Documents;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Media.Imaging;

using System.Windows.Navigation;

using System.Windows.Shapes;

using VideoChat\_Client.Services;

using VideoChat\_Client.Models;

using Supabase;

using DotNetEnv;

using System.Diagnostics;

using System.Collections.ObjectModel;

using System.ComponentModel;

namespace VideoChat\_Client.Views

{

public partial class MainWindow : Window

{

private readonly ContactsService \_contactsService;

private readonly CallsService \_callsService;

private User \_selectedUser;

private Client \_supabaseClient;

private ObservableCollection<User> \_contacts = new ObservableCollection<User>();

private CameraService \_cameraService;

private MicrophoneService \_microphoneService;

private NetworkService \_networkService;

private AudioService \_audioService;

private Guid \_currentCallId;

private enum CallState { None, Outgoing, Incoming, Active }

private CallState \_currentCallState = CallState.None;

public MainWindow()

{

InitializeComponent();

Env.TraversePath().Load();

string SupabaseUrl = Environment.GetEnvironmentVariable("SUPABASE\_URL")!;

string SupabaseKey = Environment.GetEnvironmentVariable("SUPABASE\_KEY")!;

var supabase = new Client(

SupabaseUrl,

SupabaseKey);

\_contactsService = new ContactsService(supabase);

\_callsService = new CallsService(supabase);

ContactsListView.ItemsSource = \_contacts;

\_cameraService = new CameraService();

\_microphoneService = new MicrophoneService();

\_audioService = new AudioService();

\_cameraService.FrameReady += OnCameraFrameReady;

\_microphoneService.AudioDataAvailable += OnAudioDataAvailable;

int port;

if (App.Port != 0)

port = App.Port;

else

port = 12345;

\_networkService = new NetworkService(App.CurrentUser.Id, Environment.GetEnvironmentVariable("SERVER\_IP"), int.Parse(Environment.GetEnvironmentVariable("SERVER\_PORT")), port);

\_ = Task.Run(() => \_networkService.ConnectAsync());

\_networkService.OnIncomingCallRequest += OnIncomingCall;

\_networkService.OnCallAccepted += OnCallAccepted;

\_networkService.OnCallRejected += OnCallRejected;

\_networkService.OnCallEnded += OnCallEnded;

\_networkService.VideoFrameReceived += OnRecievedVideo;

\_networkService.AudioDataReceived += OnRecievedAudio;

LoadContacts();

SetupEventHandlers();

}

private void SetupEventHandlers()

{

SearchTextBox.KeyDown += (sender, e) =>

{

if (e.Key == System.Windows.Input.Key.Enter)

{

SearchButton\_Click(sender, e);

}

};

}

//

// Работа с контактами

//

private async void LoadContacts()

{

if (App.CurrentUser == null) return;

try

{

var contacts = await \_contactsService.GetUserContactsWithDetails(App.CurrentUser.Id);

Dispatcher.Invoke(() =>

{

\_contacts.Clear();

foreach (var contact in contacts)

{

\_contacts.Add(contact);

}

Console.WriteLine($"Обновлено контактов: {\_contacts.Count}");

});

}

catch (Exception ex)

{

Console.WriteLine($"Ошибка загрузки контактов: {ex.Message}");

}

}

private async void SearchButton\_Click(object sender, RoutedEventArgs e)

{

string searchTerm = SearchTextBox.Text.Trim();

if (string.IsNullOrEmpty(searchTerm))

{

ShowError("Введите имя пользователя для поиска");

return;

}

try

{

var foundUsers = await \_contactsService.SearchUsers(searchTerm);

var user = foundUsers.FirstOrDefault();

if (user == null)

{

ShowUserNotFound();

return;

}

DisplayUserProfile(user);

await LoadCallHistory(user.Id);

}

catch (Exception ex)

{

ShowError($"Ошибка поиска: {ex.Message}");

}

}

private async void ContactsListView\_SelectionChanged(object sender, SelectionChangedEventArgs e)

{

if (ContactsListView.SelectedItem is User selectedContact)

{

\_selectedUser = selectedContact;

DisplayUserProfile(selectedContact);

}

}

private async void DisplayUserProfile(User user)

{

\_selectedUser = user;

FoundUsernameText.Text = user.Username;

AddContactButton.Visibility = ShouldHideAddButton(user.Id)

? Visibility.Collapsed

: Visibility.Visible;

await LoadCallHistory(user.Id);

UserPanel.Visibility = Visibility.Visible;

DefaultMessageText.Visibility = Visibility.Collapsed;

ErrorText.Visibility = Visibility.Collapsed;

}

private async Task LoadCallHistory(Guid contactId)

{

try

{

//if (App.CurrentUser == null) return;

//var history = await \_callsService.GetCallHistory(

// App.CurrentUser.Id,

// contactId);

//CallHistoryListView.ItemsSource = history

// .OrderByDescending(c => c.StartedAt)

// .ToList();

}

catch (Exception ex)

{

ShowError($"Ошибка загрузки истории: {ex.Message}");

}

}

private void ShowUserNotFound()

{

UserPanel.Visibility = Visibility.Collapsed;

DefaultMessageText.Visibility = Visibility.Visible;

ErrorText.Text = "Пользователь не найден";

ErrorText.Visibility = Visibility.Visible;

}

private async void AddContactButton\_Click(object sender, RoutedEventArgs e)

{

if (\_selectedUser == null || App.CurrentUser == null) return;

try

{

AddContactButton.IsEnabled = false;

AddContactButton.Content = "Добавление...";

bool success = await \_contactsService.AddContact(

App.CurrentUser.Id,

\_selectedUser.Id);

if (success)

{

LoadContacts();

AddContactButton.Visibility = ShouldHideAddButton(\_selectedUser.Id) ? Visibility.Visible : Visibility.Hidden;

}

else

{

ShowError("Контакт уже существует");

}

}

catch (Exception ex)

{

ShowError($"Ошибка: {ex.Message}");

}

finally

{

AddContactButton.IsEnabled = true;

AddContactButton.Content = "Добавить в контакты";

}

}

private bool ShouldHideAddButton(Guid contactId)

{

return \_contacts.Any(c => c.Id == contactId);

}

//

// Работа со звонком

//

private void OnRecievedVideo(BitmapImage image)

{

Dispatcher.Invoke(() =>

{

RemoteVideoDisplay.Source = image;

});

}

private void OnRecievedAudio(byte[] data)

{

Dispatcher.Invoke(() =>

{

\_audioService.EnqueueAudioData(data);

});

}

private void OnCameraFrameReady(BitmapImage image)

{

Dispatcher.Invoke(() =>

{

LocalVideoPreview.Source = image;

});

}

private void ShowCallUI(CallState state, string statusText = "")

{

Dispatcher.Invoke(() =>

{

DefaultMessageText.Visibility = Visibility.Collapsed;

UserPanel.Visibility = Visibility.Collapsed;

CallGrid.Visibility = Visibility.Visible;

CallStatusText.Text = statusText;

switch (state)

{

case CallState.Outgoing:

IncomingCallButtons.Visibility = Visibility.Collapsed;

EndCallButton.Visibility = Visibility.Visible;

CallStatusText.Text = "Вызов...";

break;

case CallState.Incoming:

IncomingCallButtons.Visibility = Visibility.Visible;

EndCallButton.Visibility = Visibility.Collapsed;

CallStatusText.Text = "Входящий вызов";

break;

case CallState.Active:

IncomingCallButtons.Visibility = Visibility.Collapsed;

EndCallButton.Visibility = Visibility.Visible;

CallStatusText.Text = "Звонок активен";

break;

case CallState.None:

CallGrid.Visibility = Visibility.Collapsed;

DefaultMessageText.Visibility =

ContactsListView.SelectedItem == null ? Visibility.Visible : Visibility.Collapsed;

UserPanel.Visibility =

ContactsListView.SelectedItem != null ? Visibility.Visible : Visibility.Collapsed;

break;

}

\_currentCallState = state;

});

}

private String GetCallerName(Guid callId)

{

return "usr";

}

private void OnIncomingCall(Guid callId)

{

if (\_currentCallState.Equals(CallState.Incoming) || \_currentCallState.Equals(CallState.Active))

return;

ShowCallUI(CallState.Incoming, $"Входящий вызов от {GetCallerName(callId)}");

}

private void OnCallAccepted(Guid callId)

{

if (\_currentCallState == CallState.Active)

return;

ShowCallUI(CallState.Active);

StartMediaDevices();

\_cameraService.SetNetworkTarget(\_networkService);

\_microphoneService.SetNetworkTarget(\_networkService);

\_networkService.StartStreaming();

}

private void OnCallRejected(Guid callId)

{

if (\_currentCallState == CallState.Active)

return;

ShowCallUI(CallState.None);

MessageBox.Show($"Вызов отклонен");

}

private void OnCallEnded(Guid callId)

{

ShowCallUI(CallState.None);

\_networkService.StopStreaming();

StopMediaDevices();

}

private async void CallButton\_Click(object sender, RoutedEventArgs e)

{

if (\_selectedUser == null || App.CurrentUser == null)

{

ShowError("Пользователь не выбран");

return;

}

DateTime callStartTime = DateTime.UtcNow;

Guid callId = Guid.Empty;

try

{

CallButton.IsEnabled = false;

ErrorText.Visibility = Visibility.Collapsed;

CallButton.Content = "Завершить звонок";

// Создаем запись о звонке

//var call = await \_callsService.StartCall(

// App.CurrentUser.Id,

// \_selectedUser.Id,

// GetLocalIpAddress(),

// 12345);

await StartNewCall(\_selectedUser.Id);

// Рассчитываем длительность

TimeSpan callDuration = DateTime.UtcNow - callStartTime;

// Обновляем статус и длительность звонка

//await \_callsService.UpdateCallStatus(

// callId,

// "completed",

// GetLocalIpAddress(),

// 12346);

await \_callsService.UpdateCallDuration(callId, callDuration);

await LoadCallHistory(\_selectedUser.Id);

}

catch (Exception ex)

{

if (callId != Guid.Empty)

{

TimeSpan callDuration = DateTime.UtcNow - callStartTime;

await \_callsService.UpdateCallStatus(callId, "failed");

await \_callsService.UpdateCallDuration(callId, callDuration);

}

ShowError($"Ошибка звонка: {ex.Message}");

}

finally

{

CallButton.IsEnabled = true;

CallButton.Content = "Позвонить";

}

}

private async Task StartNewCall(Guid tartgetId)

{

//// Создаем запись о звонке

//var call = await \_callsService.StartCall(

// App.CurrentUser.Id,

// \_selectedUser.Id,

// GetLocalIpAddress(),

// \_networkManager.UdpPort);

// Обновляем UI

CallButton.Content = "Завершить звонок";

await \_networkService.UpdateClientAsync();

\_networkService.InitiateCallAsync(tartgetId);

\_ = Task.Run(() => \_networkService.RequestCall(\_currentCallId));

}

private async void AcceptCallButton\_Click(object sender, RoutedEventArgs e)

{

await \_networkService.AcceptCall(\_currentCallId);

ShowCallUI(CallState.Active);

StartMediaDevices();

\_cameraService.SetNetworkTarget(\_networkService);

\_microphoneService.SetNetworkTarget(\_networkService);

\_networkService.StartStreaming();

}

private async void RejectCallButton\_Click(object sender, RoutedEventArgs e)

{

await \_networkService.RejectCall(\_currentCallId);

ShowCallUI(CallState.None);

StopMediaDevices();

}

private async void EndCallButton\_Click(object sender, RoutedEventArgs e)

{

await \_networkService.EndCall(\_currentCallId);

await EndCall(\_currentCallId);

ShowCallUI(CallState.None);

StopMediaDevices();

}

private void StartMediaDevices()

{

try

{

\_cameraService.FrameReady += OnCameraFrameReady;

\_cameraService.StartCamera();

LocalVideoPreview.Visibility = Visibility.Visible;

}

catch (Exception ex)

{

ShowError($"Ошибка запуска устройств: {ex.Message}");

}

try

{

\_microphoneService.AudioDataAvailable += OnAudioDataAvailable;

\_microphoneService.StartCapture();

}

catch (Exception ex)

{

ShowError($"Ошибка запуска устройств: {ex.Message}");

}

}

private void OnAudioDataAvailable(byte[] audioData)

{

Debug.WriteLine($"Получено аудиоданных: {audioData.Length} байт");

}

private string FormatDuration(TimeSpan duration)

{

if (duration.TotalHours >= 1)

return duration.ToString(@"hh\:mm\:ss");

return duration.ToString(@"mm\:ss");

}

private async Task EndCall(Guid callId)

{

try

{

StopMediaDevices();

// Обновляем статус звонка

//await \_callsService.UpdateCallStatus(

// callId,

// "completed",

// GetLocalIpAddress(),

// 12346);

await LoadCallHistory(\_selectedUser.Id);

}

catch (Exception ex)

{

ShowError($"Ошибка завершения звонка: {ex.Message}");

}

}

private void StopMediaDevices()

{

try

{

\_cameraService.StopCamera();

\_cameraService.FrameReady -= OnCameraFrameReady;

LocalVideoPreview.Source = null;

LocalVideoPreview.Visibility = Visibility.Collapsed;

\_microphoneService.Dispose();

\_microphoneService.AudioDataAvailable -= OnAudioDataAvailable;

//\_audioService.StopPlayback();

}

catch (Exception ex)

{

ShowError($"Ошибка остановки устройств: {ex.Message}");

}

}

private void ShowError(string message)

{

ErrorText.Text = message;

ErrorText.Visibility = Visibility.Visible;

}

protected override async void OnClosed(EventArgs e)

{

if (App.CurrentUser != null)

{

Env.TraversePath().Load();

string SupabaseUrl = Environment.GetEnvironmentVariable("SUPABASE\_URL")!;

string SupabaseKey = Environment.GetEnvironmentVariable("SUPABASE\_KEY")!;

var supabase = new Client(

SupabaseUrl,

SupabaseKey);

var authService = new AuthService(supabase);

await authService.Logout(App.CurrentUser.Id);

}

base.OnClosed(e);

}

}

}

using System.Text;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Data;

using System.Windows.Documents;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Media.Imaging;

using System.Windows.Navigation;

using System.Windows.Shapes;

using VideoChat\_Client.Services;

using VideoChat\_Client.Models;

using Supabase;

using DotNetEnv;

using System.Diagnostics;

using System.Collections.ObjectModel;

using System.ComponentModel;

namespace VideoChat\_Client.Views

{

public partial class MainWindow : Window

{

private readonly ContactsService \_contactsService;

private readonly CallsService \_callsService;

private User \_selectedUser;

private Client \_supabaseClient;

private ObservableCollection<User> \_contacts = new ObservableCollection<User>();

private CameraService \_cameraService;

private MicrophoneService \_microphoneService;

private NetworkService \_networkService;

private AudioService \_audioService;

private Guid \_currentCallId;

private enum CallState { None, Outgoing, Incoming, Active }

private CallState \_currentCallState = CallState.None;

public MainWindow()

{

InitializeComponent();

Env.TraversePath().Load();

string SupabaseUrl = Environment.GetEnvironmentVariable("SUPABASE\_URL")!;

string SupabaseKey = Environment.GetEnvironmentVariable("SUPABASE\_KEY")!;

var supabase = new Client(

SupabaseUrl,

SupabaseKey);

\_contactsService = new ContactsService(supabase);

\_callsService = new CallsService(supabase);

ContactsListView.ItemsSource = \_contacts;

\_cameraService = new CameraService();

\_microphoneService = new MicrophoneService();

\_audioService = new AudioService();

\_cameraService.FrameReady += OnCameraFrameReady;

\_microphoneService.AudioDataAvailable += OnAudioDataAvailable;

int port;

if (App.Port != 0)

port = App.Port;

else

port = 12345;

\_networkService = new NetworkService(App.CurrentUser.Id, Environment.GetEnvironmentVariable("SERVER\_IP"), int.Parse(Environment.GetEnvironmentVariable("SERVER\_PORT")), port);

\_ = Task.Run(() => \_networkService.ConnectAsync());

\_networkService.OnIncomingCallRequest += OnIncomingCall;

\_networkService.OnCallAccepted += OnCallAccepted;

\_networkService.OnCallRejected += OnCallRejected;

\_networkService.OnCallEnded += OnCallEnded;

\_networkService.VideoFrameReceived += OnRecievedVideo;

\_networkService.AudioDataReceived += OnRecievedAudio;

LoadContacts();

SetupEventHandlers();

}

private void SetupEventHandlers()

{

SearchTextBox.KeyDown += (sender, e) =>

{

if (e.Key == System.Windows.Input.Key.Enter)

{

SearchButton\_Click(sender, e);

}

};

}

//

// Работа с контактами

//

private async void LoadContacts()

{

if (App.CurrentUser == null) return;

try

{

var contacts = await \_contactsService.GetUserContactsWithDetails(App.CurrentUser.Id);

Dispatcher.Invoke(() =>

{

\_contacts.Clear();

foreach (var contact in contacts)

{

\_contacts.Add(contact);

}

Console.WriteLine($"Обновлено контактов: {\_contacts.Count}");

});

}

catch (Exception ex)

{

Console.WriteLine($"Ошибка загрузки контактов: {ex.Message}");

}

}

private async void SearchButton\_Click(object sender, RoutedEventArgs e)

{

string searchTerm = SearchTextBox.Text.Trim();

if (string.IsNullOrEmpty(searchTerm))

{

ShowError("Введите имя пользователя для поиска");

return;

}

try

{

var foundUsers = await \_contactsService.SearchUsers(searchTerm);

var user = foundUsers.FirstOrDefault();

if (user == null)

{

ShowUserNotFound();

return;

}

DisplayUserProfile(user);

await LoadCallHistory(user.Id);

}

catch (Exception ex)

{

ShowError($"Ошибка поиска: {ex.Message}");

}

}

private async void ContactsListView\_SelectionChanged(object sender, SelectionChangedEventArgs e)

{

if (ContactsListView.SelectedItem is User selectedContact)

{

\_selectedUser = selectedContact;

DisplayUserProfile(selectedContact);

}

}

private async void DisplayUserProfile(User user)

{

\_selectedUser = user;

FoundUsernameText.Text = user.Username;

AddContactButton.Visibility = ShouldHideAddButton(user.Id)

? Visibility.Collapsed

: Visibility.Visible;

await LoadCallHistory(user.Id);

UserPanel.Visibility = Visibility.Visible;

DefaultMessageText.Visibility = Visibility.Collapsed;

ErrorText.Visibility = Visibility.Collapsed;

}

private async Task LoadCallHistory(Guid contactId)

{

try

{

//if (App.CurrentUser == null) return;

//var history = await \_callsService.GetCallHistory(

// App.CurrentUser.Id,

// contactId);

//CallHistoryListView.ItemsSource = history

// .OrderByDescending(c => c.StartedAt)

// .ToList();

}

catch (Exception ex)

{

ShowError($"Ошибка загрузки истории: {ex.Message}");

}

}

private void ShowUserNotFound()

{

UserPanel.Visibility = Visibility.Collapsed;

DefaultMessageText.Visibility = Visibility.Visible;

ErrorText.Text = "Пользователь не найден";

ErrorText.Visibility = Visibility.Visible;

}

private async void AddContactButton\_Click(object sender, RoutedEventArgs e)

{

if (\_selectedUser == null || App.CurrentUser == null) return;

try

{

AddContactButton.IsEnabled = false;

AddContactButton.Content = "Добавление...";

bool success = await \_contactsService.AddContact(

App.CurrentUser.Id,

\_selectedUser.Id);

if (success)

{

LoadContacts();

AddContactButton.Visibility = ShouldHideAddButton(\_selectedUser.Id) ? Visibility.Visible : Visibility.Hidden;

}

else

{

ShowError("Контакт уже существует");

}

}

catch (Exception ex)

{

ShowError($"Ошибка: {ex.Message}");

}

finally

{

AddContactButton.IsEnabled = true;

AddContactButton.Content = "Добавить в контакты";

}

}

private bool ShouldHideAddButton(Guid contactId)

{

return \_contacts.Any(c => c.Id == contactId);

}

//

// Работа со звонком

//

private void OnRecievedVideo(BitmapImage image)

{

Dispatcher.Invoke(() =>

{

RemoteVideoDisplay.Source = image;

});

}

private void OnRecievedAudio(byte[] data)

{

Dispatcher.Invoke(() =>

{

\_audioService.EnqueueAudioData(data);

});

}

private void OnCameraFrameReady(BitmapImage image)

{

Dispatcher.Invoke(() =>

{

LocalVideoPreview.Source = image;

});

}

private void ShowCallUI(CallState state, string statusText = "")

{

Dispatcher.Invoke(() =>

{

DefaultMessageText.Visibility = Visibility.Collapsed;

UserPanel.Visibility = Visibility.Collapsed;

CallGrid.Visibility = Visibility.Visible;

CallStatusText.Text = statusText;

switch (state)

{

case CallState.Outgoing:

IncomingCallButtons.Visibility = Visibility.Collapsed;

EndCallButton.Visibility = Visibility.Visible;

CallStatusText.Text = "Вызов...";

break;

case CallState.Incoming:

IncomingCallButtons.Visibility = Visibility.Visible;

EndCallButton.Visibility = Visibility.Collapsed;

CallStatusText.Text = "Входящий вызов";

break;

case CallState.Active:

IncomingCallButtons.Visibility = Visibility.Collapsed;

EndCallButton.Visibility = Visibility.Visible;

CallStatusText.Text = "Звонок активен";

break;

case CallState.None:

CallGrid.Visibility = Visibility.Collapsed;

DefaultMessageText.Visibility =

ContactsListView.SelectedItem == null ? Visibility.Visible : Visibility.Collapsed;

UserPanel.Visibility =

ContactsListView.SelectedItem != null ? Visibility.Visible : Visibility.Collapsed;

break;

}

\_currentCallState = state;

});

}

private String GetCallerName(Guid callId)

{

return "usr";

}

private void OnIncomingCall(Guid callId)

{

if (\_currentCallState.Equals(CallState.Incoming) || \_currentCallState.Equals(CallState.Active))

return;

ShowCallUI(CallState.Incoming, $"Входящий вызов от {GetCallerName(callId)}");

}

private void OnCallAccepted(Guid callId)

{

if (\_currentCallState == CallState.Active)

return;

ShowCallUI(CallState.Active);

StartMediaDevices();

\_cameraService.SetNetworkTarget(\_networkService);

\_microphoneService.SetNetworkTarget(\_networkService);

\_networkService.StartStreaming();

}

private void OnCallRejected(Guid callId)

{

if (\_currentCallState == CallState.Active)

return;

ShowCallUI(CallState.None);

MessageBox.Show($"Вызов отклонен");

}

private void OnCallEnded(Guid callId)

{

ShowCallUI(CallState.None);

\_networkService.StopStreaming();

StopMediaDevices();

}

private async void CallButton\_Click(object sender, RoutedEventArgs e)

{

if (\_selectedUser == null || App.CurrentUser == null)

{

ShowError("Пользователь не выбран");

return;

}

DateTime callStartTime = DateTime.UtcNow;

Guid callId = Guid.Empty;

try

{

CallButton.IsEnabled = false;

ErrorText.Visibility = Visibility.Collapsed;

CallButton.Content = "Завершить звонок";

// Создаем запись о звонке

//var call = await \_callsService.StartCall(

// App.CurrentUser.Id,

// \_selectedUser.Id,

// GetLocalIpAddress(),

// 12345);

await StartNewCall(\_selectedUser.Id);

// Рассчитываем длительность

TimeSpan callDuration = DateTime.UtcNow - callStartTime;

// Обновляем статус и длительность звонка

//await \_callsService.UpdateCallStatus(

// callId,

// "completed",

// GetLocalIpAddress(),

// 12346);

await \_callsService.UpdateCallDuration(callId, callDuration);

await LoadCallHistory(\_selectedUser.Id);

}

catch (Exception ex)

{

if (callId != Guid.Empty)

{

TimeSpan callDuration = DateTime.UtcNow - callStartTime;

await \_callsService.UpdateCallStatus(callId, "failed");

await \_callsService.UpdateCallDuration(callId, callDuration);

}

ShowError($"Ошибка звонка: {ex.Message}");

}

finally

{

CallButton.IsEnabled = true;

CallButton.Content = "Позвонить";

}

}

private async Task StartNewCall(Guid tartgetId)

{

//// Создаем запись о звонке

//var call = await \_callsService.StartCall(

// App.CurrentUser.Id,

// \_selectedUser.Id,

// GetLocalIpAddress(),

// \_networkManager.UdpPort);

// Обновляем UI

CallButton.Content = "Завершить звонок";

await \_networkService.UpdateClientAsync();

\_networkService.InitiateCallAsync(tartgetId);

\_ = Task.Run(() => \_networkService.RequestCall(\_currentCallId));

}

private async void AcceptCallButton\_Click(object sender, RoutedEventArgs e)

{

await \_networkService.AcceptCall(\_currentCallId);

ShowCallUI(CallState.Active);

StartMediaDevices();

\_cameraService.SetNetworkTarget(\_networkService);

\_microphoneService.SetNetworkTarget(\_networkService);

\_networkService.StartStreaming();

}

private async void RejectCallButton\_Click(object sender, RoutedEventArgs e)

{

await \_networkService.RejectCall(\_currentCallId);

ShowCallUI(CallState.None);

StopMediaDevices();

}

private async void EndCallButton\_Click(object sender, RoutedEventArgs e)

{

await \_networkService.EndCall(\_currentCallId);

await EndCall(\_currentCallId);

ShowCallUI(CallState.None);

StopMediaDevices();

}

private void StartMediaDevices()

{

try

{

\_cameraService.FrameReady += OnCameraFrameReady;

\_cameraService.StartCamera();

LocalVideoPreview.Visibility = Visibility.Visible;

}

catch (Exception ex)

{

ShowError($"Ошибка запуска устройств: {ex.Message}");

}

try

{

\_microphoneService.AudioDataAvailable += OnAudioDataAvailable;

\_microphoneService.StartCapture();

}

catch (Exception ex)

{

ShowError($"Ошибка запуска устройств: {ex.Message}");

}

}

private void OnAudioDataAvailable(byte[] audioData)

{

Debug.WriteLine($"Получено аудиоданных: {audioData.Length} байт");

}

private string FormatDuration(TimeSpan duration)

{

if (duration.TotalHours >= 1)

return duration.ToString(@"hh\:mm\:ss");

return duration.ToString(@"mm\:ss");

}

private async Task EndCall(Guid callId)

{

try

{

StopMediaDevices();

// Обновляем статус звонка

//await \_callsService.UpdateCallStatus(

// callId,

// "completed",

// GetLocalIpAddress(),

// 12346);

await LoadCallHistory(\_selectedUser.Id);

}

catch (Exception ex)

{

ShowError($"Ошибка завершения звонка: {ex.Message}");

}

}

private void StopMediaDevices()

{

try

{

\_cameraService.StopCamera();

\_cameraService.FrameReady -= OnCameraFrameReady;

LocalVideoPreview.Source = null;

LocalVideoPreview.Visibility = Visibility.Collapsed;

\_microphoneService.Dispose();

\_microphoneService.AudioDataAvailable -= OnAudioDataAvailable;

//\_audioService.StopPlayback();

}

catch (Exception ex)

{

ShowError($"Ошибка остановки устройств: {ex.Message}");

}

}

private void ShowError(string message)

{

ErrorText.Text = message;

ErrorText.Visibility = Visibility.Visible;

}

protected override async void OnClosed(EventArgs e)

{

if (App.CurrentUser != null)

{

Env.TraversePath().Load();

string SupabaseUrl = Environment.GetEnvironmentVariable("SUPABASE\_URL")!;

string SupabaseKey = Environment.GetEnvironmentVariable("SUPABASE\_KEY")!;

var supabase = new Client(

SupabaseUrl,

SupabaseKey);

var authService = new AuthService(supabase);

await authService.Logout(App.CurrentUser.Id);

}

base.OnClosed(e);

}

}

}

using Postgrest.Attributes;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using Postgrest.Models;

namespace VideoChat\_Client.Models

{

[Table("users")]

public class User : BaseModel

{

[PrimaryKey("id", false)]

public Guid Id { get; set; }

[Column("username")]

public string Username { get; set; }

[Column("password\_hash")]

public string PasswordHash { get; set; }

[Column("created\_at")]

public DateTime CreatedAt { get; set; }

[Column("last\_online")]

public DateTime? LastOnline { get; set; }

[Column("is\_active")]

public bool IsActive { get; set; }

}

}