МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет	*Кафедра*
вычислительной техники	«Вычислительная техника»

Направление подготовки: 09.03.01 «Информатика и вычислительная техника»

Профиль подготовки: «Программное обеспечение средств вычислительной техники и автоматизированных систем»

БАКАЛАВРСКАЯ РАБОТА

на тему

РАСПРЕДЕЛЕННАЯ ИНТЕЛЛЕКТУАЛЬНАЯ СИСТЕМА

Студент _____ Агапов Илья Евгеньевич
(подпись, дата) 19.06.2026

Руководитель _____ С.А. Зинкин
(подпись, дата) 19.06.2026

Нормоконтролёр _____ А.С. Бычков
(подпись, дата) 19.06.2026

Работа допущена к защите (протокол заседания кафедры от 22.05.2026 г. №14)

Заведующий кафедрой _____ М.А. Митрохин
(подпись)

Работа защищена с отметкой хорошо (протокол заседания ГЭК от 24.06.26 № 44)

Секретарь ГЭК _____ Е.И. Гурин
(подпись)

Пенза 2026

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет
вычислительной техники

Кафедра
«Вычислительная техника»

«Утверждаю»
Заведующий кафедрой
М.А. Митрохин
«07» апреля 2026г.

ЗАДАНИЕ
НА БАКАЛАВРСКУЮ РАБОТУ

1. Студент Агапов Илья Евгеньевич гр.22ВВП2 факультета вычислительной техники.

2.Тема работы: «Распределенная интеллектуальная система»

Тема утверждена приказом ПГУ №347от "07" апреля 2026 г.

3. Руководитель бакалаврской работы д.т.н., профессор каф. ВТ ПГУ Зинкин С. А.

4. Задание на работу (назначение разработки, исходные данные и т.п.)

Разработать и реализовать клиент-серверную распределённую интеллектуальную систему, обеспечивающую защищённый обмен текстовыми сообщениями и файлами между пользователями внутри корпоративной сети, хранение истории переписки в централизованной базе данных, а также интеграцию AI-ассистента для автоматизированного ответа на запросы пользователей через внешний API. Система должна функционировать под управлением ОС Windows, поддерживать как консольный, так и графический интерфейс, и предоставлять возможности регистрации/авторизации, пересылки и удаления сообщений, а также отображения статуса «в сети».

5. Перечень подлежащих разработке вопросов

Требуется провести анализ существующих аналогов. Выбрать и обосновать технологии реализации (среда разработки, язык программирования, стек сетевых технологий, СУБД, API для интеграции). Разработать общую архитектуру системы (клиент-серверная модель), определить протокол обмена сообщениями (формат команд, структура пакетов). Спроектировать структуру базы данных.

6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1	Изучение предметной области	20	7.04.2026 – 10.04.2026	
2	Проектирование базы данных и протокола обмена	62	11.04.2026 – 20.04.2026	
3	Разработка серверной части приложения	80	21.04.2026 – 01.05.2026	
4	Разработка клиента	58	02.05.2026 – 10.05.2026	
5	Отладка и тестирование	50	11.05.2026 – 25.05.2026	
6	Оформление пояснительной записки	54	16.05.2026 – 21.05.2026	

Дата выдачи задания "07" апреля 2026 г.

Руководитель бакалаврской работы _____ Зинкин С.А.
(подпись, дата)

Задание к исполнению принял студент _____ Агапов И.Е.
(подпись, дата)

Работу к защите допустить
Декан факультета 23.06.2026 Фионова Л.Р.
(подпись, дата)

Реферат

Бакалаврская работа содержит 151 лист, 45 рисунков. 5 таблиц, 4 источника и 3 приложения.

РАСПРЕДЕЛЕННАЯ ИНТЕЛЛЕКТУАЛЬНАЯ СИСТЕМА, WINDOWS, C++, WINSOCK2, ODBC, MICROSOFT VISUAL STUDIO, MICROSOFT SQL SERVER, КЛИЕНТ-СЕРВЕР, AI-АССИСТЕНТ, БАЗА ДАННЫХ

Цель работы – разработка клиент-серверной распределенной интеллектуальной системы под управлением ОС Windows с функциями обмена и пересылками сообщениями, передачи файлов и интеграции AI-ассистента.

Объектом разработки является серверная часть на C++ с использованием WinSock2 и ODBC, консольный клиент ClientConsole, графический клиент Vis, а также база данных Microsoft SQL Server, обеспечивающие регистрацию и авторизацию пользователей, отправку и получение текстовых сообщений, передачу файлов, пересылку и удаление сообщений, сохранение истории переписки и работу AI-чата через внешний API.

В результате проделанной работы разработано приложение, полностью соответствующее целям данной выпускной квалификационной работы.

Содержание

Перечень сокращений и обозначений 5
Введение 6
1 Распределенная интеллектуальная система 8
1.1 Анализ предметной области и постановка задачи 8
1.1.1 Анализ аналогов 8
1.1.2 Постановка задачи 11
1.2 Выбор технологий и среды разработки 12
1.2.1 Выбор среды разработки 12
1.2.2 Установка и настройка среды разработки Microsoft Visual Studio 15
1.2.3 Язык программирования 22
1.2.4 Сетевой стек WinSock2 22
1.2.5 СУБД Microsoft SQL Server 23
1.2.6 Технология доступа к БД ODBC 23
1.3 Выбор операционной системы 24
1.3.1 ОС Windows 24
1.3.2 ОС Linux 25
1.3.3 ОС macOS 25
1.4 Архитектура приложения 26
1.4.1 Общая клиент-серверная архитектура 26
1.4.2 Архитектура серверной части 27
1.4.3 Архитектура клиентской части 28
1.4.4 Протокол обмена сообщениями 29
1.4.5 Описание файлов проекта 30
1.4.6 Реализация AI-ассистента 31
1.5 Реализация приложения 32
1.5.1 Реализация серверной части 32
1.5.2 Реализация консольного клиента ClientConsole 32
1.5.3 Реализация графического клиента Vis 32
1.5.4 Интеграция AI-ассистента 33
1.5.5 Структура приложения 34

1.5.6 Структура базы данных 37
1.6 Тестирование приложения 39
1.6.1 Цели и задачи тестирования 39
1.6.2 Тестирование подключения и авторизации 39
1.6.3 Тестирование главного меню 43
1.6.4 Тестирование обмена сообщениями 45
1.6.5 Тестирование передачи файлов и пересылки сообщений 46
1.6.6 Тестирование AI- ассистента 50
2 Безопасность жизнедеятельности 52
2.1 Условия труда при разработке ПО 52
2.2 Анализ опасных и вредных факторов 52
2.3 Требования к организации рабочего места 52
2.4 Электробезопасность и пожарная безопасность 53
Заключение 54
Список использованных источников 55
Приложение А – UML-диаграммы 56
Приложение Б – Листинг программы 61
Приложение В – Презентация 144

Перечень сокращений и обозначений

БД – база данных

ИИ – искусственный интелект

ОС – операционная система

СУБД – система управления базами данных

AI – (англ. Artificial Intelligence) – искусственный интеллект

API – (англ. Application Programming Interface) – программный интерфейс приложения

GUI – (англ. Graphical User Interface) – графический интерфейс пользователя

IDE – (англ. Integrated Development Environment) – интегрированная среда разработки

ODBC – (англ. Open Database Connectivity) – открытый интерфейс доступа к базам данных

TCP – (англ. Transmission Control Protocol) – протокол управления передачей файлов

SQL – (англ. Structured Query Language) – структурированный язык запросов

UML – (англ. Unified Modeling Language) – унифицированный язык моделирования

Введение

В современном информационном обществе эффективная внутренняя коммуникация является одним из ключевых условий стабильной и результативной работы предприятий. Для решения коммуникационных задач используются различные инструменты, включая электронную почту, голосовую и видеосвязь, корпоративные порталы, однако наиболее оперативным и удобным способом повседневного обмена информацией остаются мессенджеры.

На практике широкое распространение получили Telegram, Вконтакте (Vk) поскольку они обеспечивают быстрый обмен текстовыми сообщениями, файлами и служебными уведомлениями. Одновременно с этим для многих организаций критичными становятся вопросы конфиденциальности, контроля доступа и хранения данных в рамках собственной инфраструктуры. По этой причине компании все чаще ориентируются на распределенные интеллектуальные системы, которые можно адаптировать под внутренние требования безопасности и бизнес-процессы.

Актуальность темы обусловлена возрастающей потребностью в специализированных средствах коммуникации, способных обеспечить не только удобство взаимодействия сотрудников, но и управляемость всей информационной среды.

В рамках настоящего проекта рассматривается разработка клиент-серверной распределенной интеллектуальной системы под управлением ОС Windows, включающего серверную часть ServerSide, консольный клиент ClientConsole и графический клиент Vis. Такая структура позволяет объединить удобство пользовательского интерфейса с возможностью технической отладки и контроля сетевого протокола.

В работе выполняется анализ существующих аналогов, формируются требования к системе, обосновывается выбор технологического стека и проектируется архитектура приложения. Дополнительно рассматриваются вопросы реализации ключевых функций, включая регистрацию и авторизацию

пользователей, обмен сообщениями, передачу файлов, хранение истории переписки в SQL Server и интеграцию AI-ассистента через внешний API.

Результатом работы является создание функциональной распределенной интеллектуальной системы, обеспечивающего безопасный обмен данными между пользователями, повышение качества внутренней коммуникации и технологическую основу для дальнейшего расширения системы.

1 Распределенная интеллектуальная система

1.1 Анализ предметной области и постановка задачи

Распределенная интеллектуальная система предназначена для организации оперативного обмена информацией внутри компании.

Ключевые требования к такой системе:

– контролируемая авторизация пользователей;

– надежная доставка сообщений;

– хранение истории переписки;

– поддержка обмена файлами;

– возможность расширения функциональности.

В отличие от публичных мессенджеров, решение на базе собственной инфраструктуры должно обеспечивать больший контроль над данными и возможность локального развертывания.

1.1.1 Анализ аналогов

На рынке представлены массовые платформы коммуникации. Их анализ показывает, что большинство решений обеспечивают удобный обмен сообщениями, однако не всегда дают полный контроль над хранением данных и серверной инфраструктурой. Среди аналогов можно выделить наиболее популярные:

– «Telegram»;

– «Вконтакте(VK)».

Рассмотрим мессенджер – Telegram (рисунок 1).

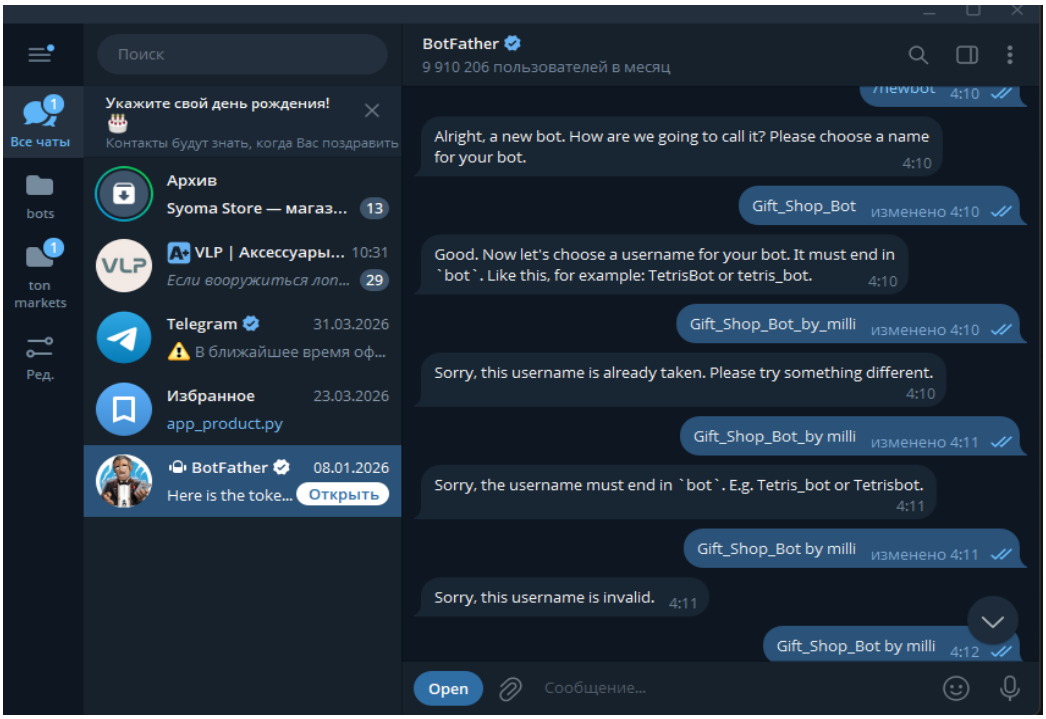


Рисунок 1 – Интерфейс мессенджера «Telegram»

Рассмотрим преимущества и недостатки данного мессенджера.

Преимущества Telegram:

– высокая скорость доставки сообщений;

– мульти-платформенность;

– удобный интерфейс.

Недостатки:

– отсутствие возможности развертывания на собственных серверах, что создает риски утечки коммерческой тайны и делает невозможным полный аудит безопасности силами компании;

– сквозное шифрование (E2EE) по умолчанию отсутствует в групповых чатах, каналах и облачных диалогах, а используется только в режиме «секретных чатов», что ограничивает повсеместную защиту конфиденциальной переписки;

– хранение метаданных и контактов на серверах Telegram снижает уровень анонимности и не позволяет службе безопасности организации контролировать цепочки взаимодействия сотрудников;

– зависимость от инфраструктуры и технической политики стороннего разработчика, включая риск внезапных изменений API или блокировок на уровне трафика;

– не работает в Российской Федерации.

Рассмотрим мессенджер – ВКонтакте (VK) (рисунок 2).

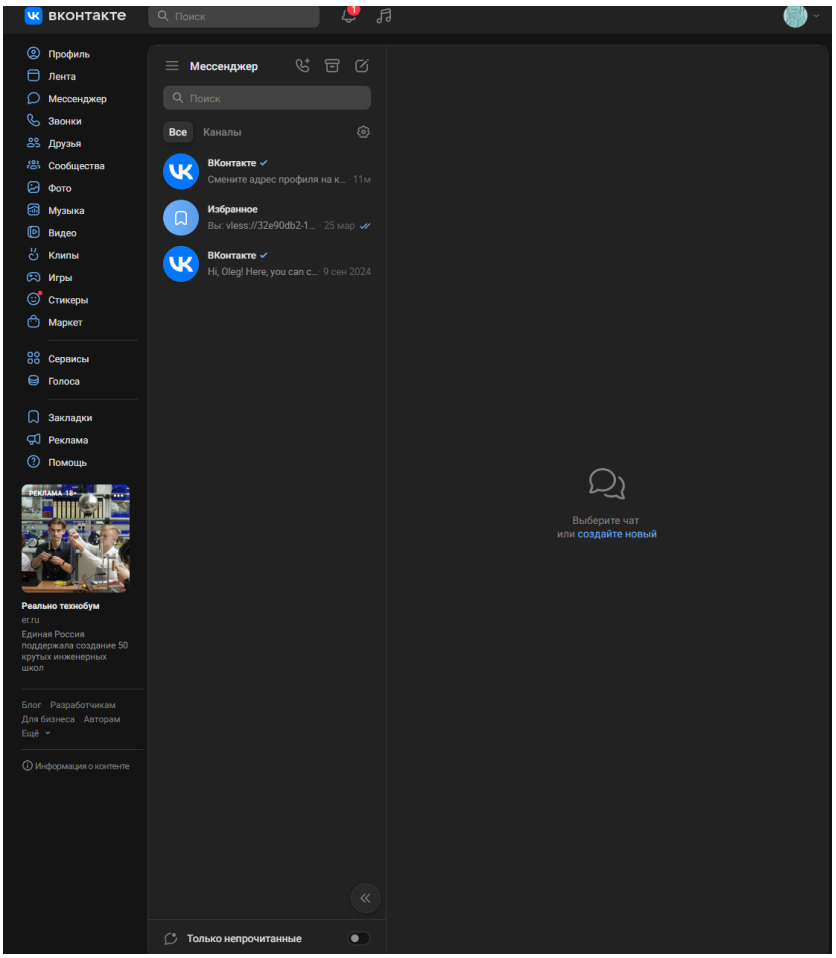


Рисунок 2 – Интерфейс мессенджер «Вконтакте(VK)»

Рассмотрим преимущества и недостатки данного мессенджера.

Преимущества:

– обширная пользовательская база в России и странах СНГ, что упрощает внешние коммуникации с клиентами и партнерами без необходимости установки дополнительного ПО;

– глубокая интеграция с экосистемой VK (почта, облачное хранилище, документы, звонки), позволяющая объединить общение, управление задачами и совместную работу в единой среде;

– поддержка сообществ, чат-ботов и авторских рассылок, что делает платформу удобным инструментом для клиентского сервиса и маркетинга внутри мессенджера.

Недостатки:

– серверная инфраструктура также расположена вне периметра организации: данные хранятся на мощностях VK, что не позволяет реализовать сценарий полного владения информацией и требует доверия провайдеру;

– гибкость кастомизации интерфейса и функционала под внутренние бизнес-процессы ограничена: брендирование и глубокая модификация клиента либо невозможны, либо требуют сложных согласований с вендором;

– масштабирование и интеграция с узкоспециализированным корпоративным ПО (ERP, самописные системы) часто требует написания нестандартных прослоек, а документация API может не покрывать всех потребностей крупной организации;

– нет ИИ-ассистента.

1.1.2 Постановка задачи

Необходимо разработать клиент-серверную распределенную интеллектуальную систему под Windows, обеспечивающую:

– регистрацию и авторизацию пользователей;

– обмен текстовыми сообщениями между клиентами;

– передачу файловых сообщений;

– пересылку и удаление сообщений;

– хранение истории переписки в SQL Server;

– поддержку AI-ассистента через внешний API;

– работу как с консольным, так и с графическим клиентом.

На рисунке 3 представлена диаграмма прецедентов распределенной интеллектуальной системы.

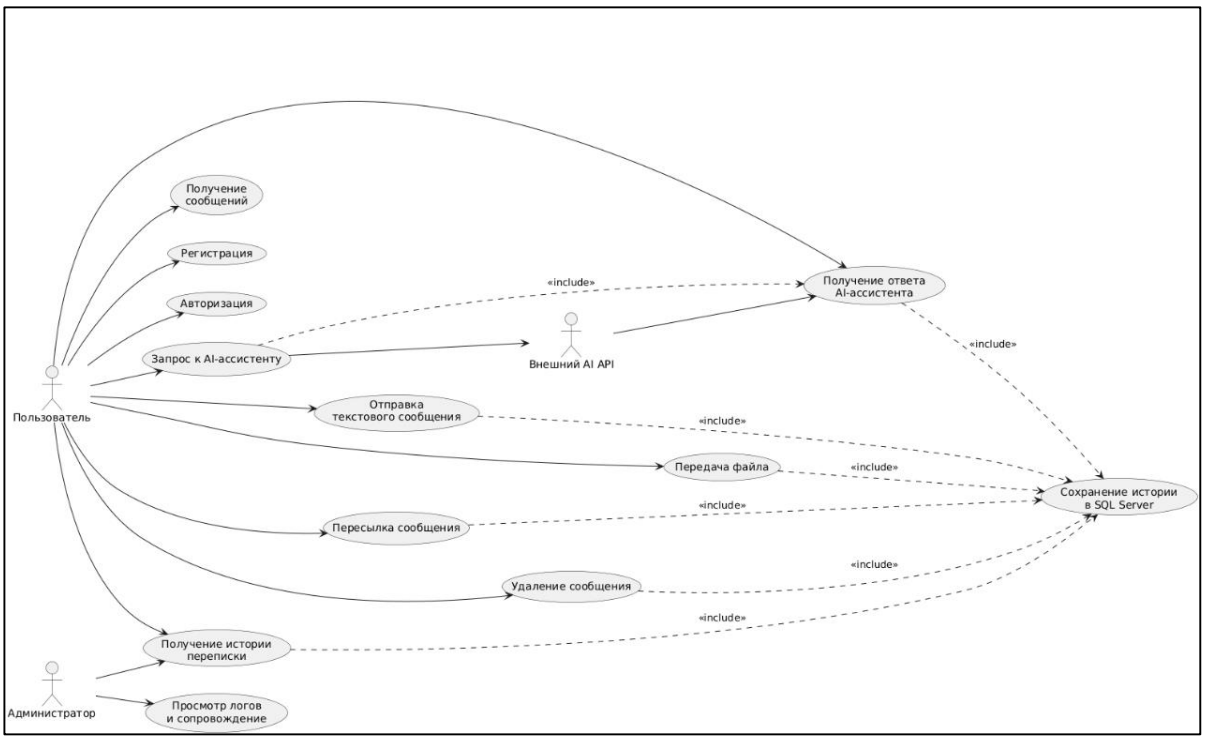


Рисунок 3 – UML-диаграмма прецедентов

1.2 Выбор технологий и среды разработки

Технологии выбирались с учетом целевой платформы Windows, удобства сопровождения и совместимости компонентов.

1.2.1 Выбор среды разработки

На данный момент в сфере разработки под Windows существует большое количество IDE и редакторов кода.

При выборе среды разработки для проекта были рассмотрены наиболее распространенные решения:

– Qt Creator;

– Eclipse;

– VS Code;

– Microsoft Visual Studio.

Qt Creator.

Преимущества:

– удобная работа с C++ и кроссплатформенными проектами;

– встроенные инструменты UI-разработки (особенно для Qt-приложений);

– сравнительно понятный интерфейс и быстрый старт.

Недостатки:

– основной фокус на экосистеме Qt, что не всегда удобно для Windows-специфичных проектов;

– менее глубокая интеграция с нативными инструментами Visual C++/MSBuild;

– в текущем проекте потребовалась бы дополнительная настройка под существующие .sln/.vcxproj.

Eclipse (CDT).

Преимущества:

– бесплатность и развитая экосистема плагинов;

– поддержка C/C++ через CDT;

– кроссплатформенность.

Недостатки:

– более сложная первоначальная настройка C++-окружения под Windows;

– менее удобная работа с нативными проектами Visual Studio;

– отладка и интеграция с Windows SDK обычно требуют больше ручных действий.

Visual Studio Code.

Преимущества:

– легковесность и высокая скорость запуска;

– гибкость за счет расширений;

– удобен как универсальный редактор для разных языков.

Недостатки:

– это редактор, а не полнофункциональная IDE;

– для C++ под Windows требуется отдельная настройка компилятора, отладчика и сборки;

– сложнее обеспечить единообразие среды для командной разработки без дополнительной конфигурации.

Microsoft Visual Studio.

Преимущества:

– наиболее зрелая поддержка C++ под Windows;

– встроенная отладка, профилирование и диагностика ошибок;

– нативная работа с MSBuild, Windows SDK, WinSock2, ODBC;

– прямое открытие и сопровождение файлов решений и проектов (.sln, .vcxproj);

– удобная работа с несколькими связанными проектами в одном решении.

Недостатки:

– более высокие требования к ресурсам ПК;

– часть возможностей доступна только в платных редакциях;

– больший объем установки по сравнению с легковесными редакторами.

Наглядно можно увидеть сравнение сред разработки в таблице 1.

Таблица 1 – Сравнение платформ для разработки приложения

Среда разработки	Преимущества	Недостатки
Qt Creator	– удобная работа с C++ и кроссплатформенными проектами; – встроенные инструменты UI-разработки; – понятный интерфейс.	– основной фокус на экосистеме Qt, что не всегда удобно для Windows проектов; – менее глубокая интеграция с нативными инструментами Visual C++/MSBuild;
Eclipse(CDT)	– бесплатность и развитая экосистема плагинов;	– более сложная первоначальная настройка

Среда разработки	Преимущества	Недостатки
	– поддержка C/C++ через CDT; – кроссплатформенность.	C++ окружения под Windows; – не удобная работа с нативными проектами Visual Studio.
Visual Studio Code	– легковесная и высокая скорость запуска; – гибкость за счёт расширений.	– это редактор кода, а не полноценная IDE; – для C++ под Windows разработки потребуется отдельная настройка компилятора.
Microsoft Visual Studio	– лучшая поддержка C++ под Windows; – встроенная откладка; – нативная работа с MSBuild, Windows SDK, WinSock2, ODBC; – прямое открытие решений и проектов(.sln, .vcxproj).	– более высокие требования к ресурсу ПК; – часть возможностей доступна только в платной версии.

После сравнения и подбора лучшей среды разработки был выбран Microsoft Visual Studio. Выбор обусловлен тем, что исходная структура проекта уже основана на решениях и проектах данной среды (ServerSide.sln и ClientOOS/ClientConsole.sln), а серверная и клиентская части используют Windows-ориентированный стек технологий (WinSock2, ODBC, Windows API). В этих условиях Visual Studio обеспечивает минимальные затраты на настройку сборки и отладки, снижает вероятность конфигурационных ошибок при совместной разработке серверных и клиентских модулей, а также упрощает сопровождение и демонстрацию проекта на этапе защиты.

1.2.2 Установка и настройка среды разработки Microsoft Visual Studio

Перед установкой необходимо скачать установщик Visual Studio Installer с официального сайта. Выбор был сделан в пользу Visual Studio 2022, так как новая

версия Visual Studio 2026 вышла относительно недавно и может работать не стабильно. Процесс установки представлен на рисунках 4 – 8.

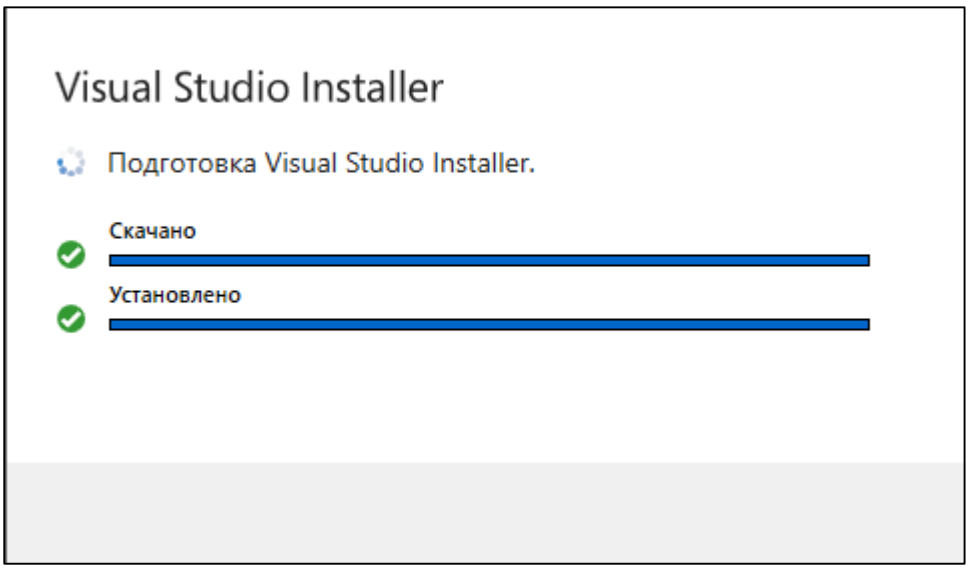


Рисунок 4 – Установка Visual Studio Installer

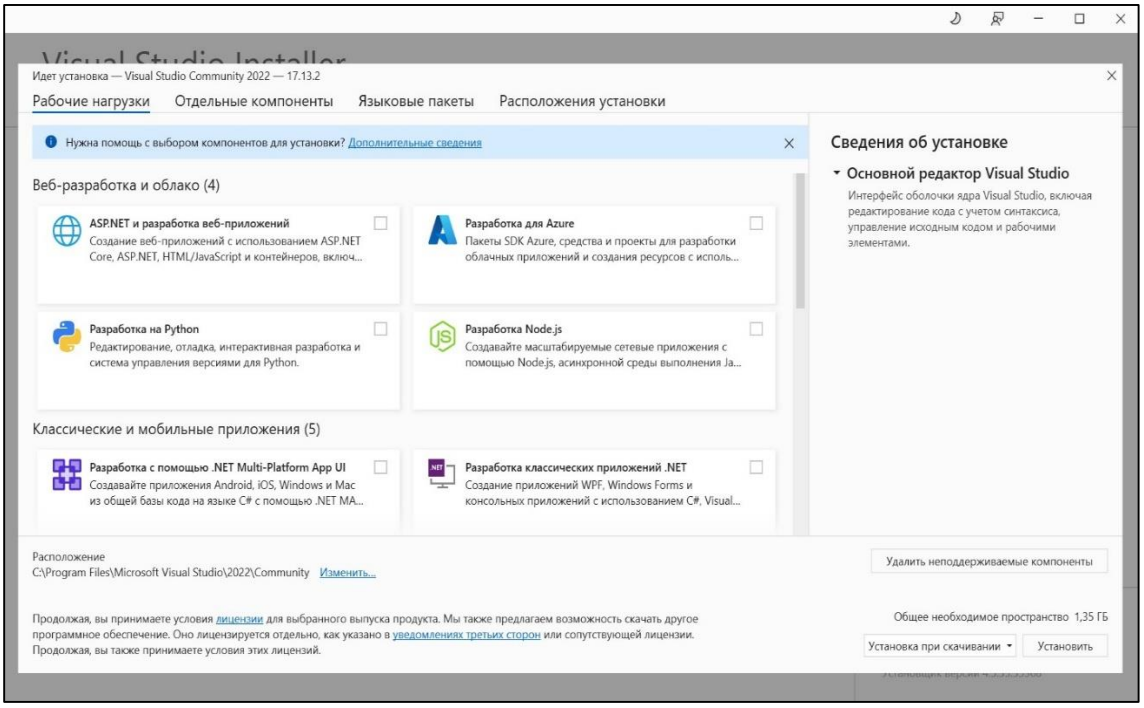


Рисунок 5 – Окно установщика

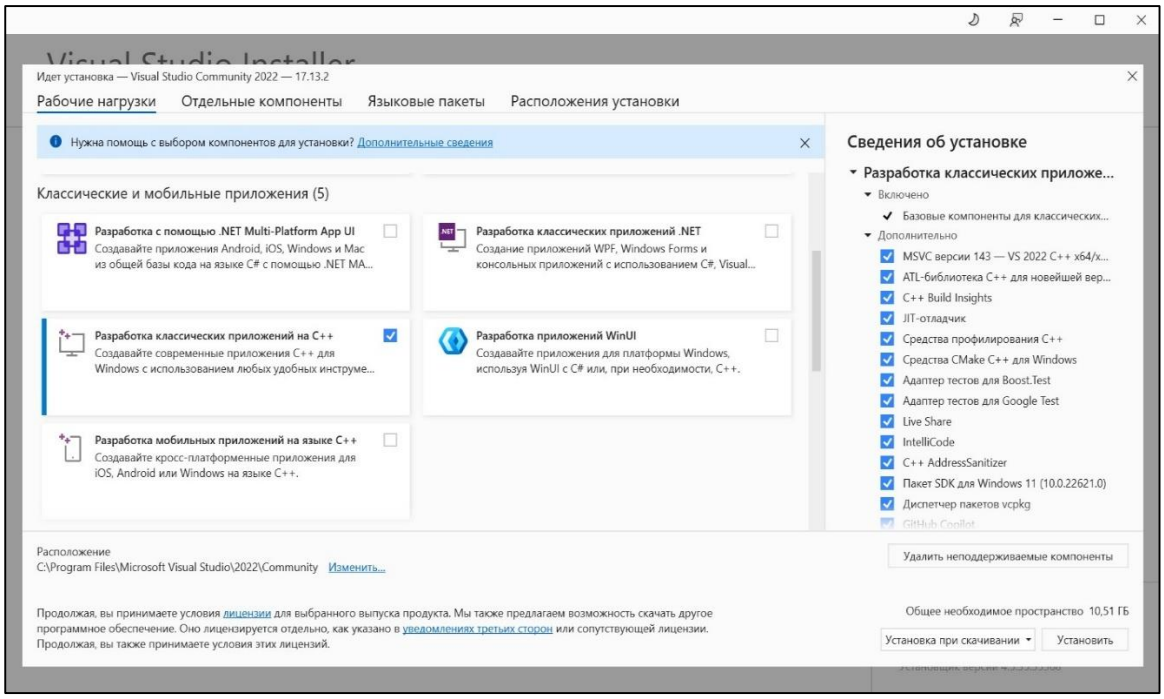


Рисунок 6 – Выбор компонентов для установки

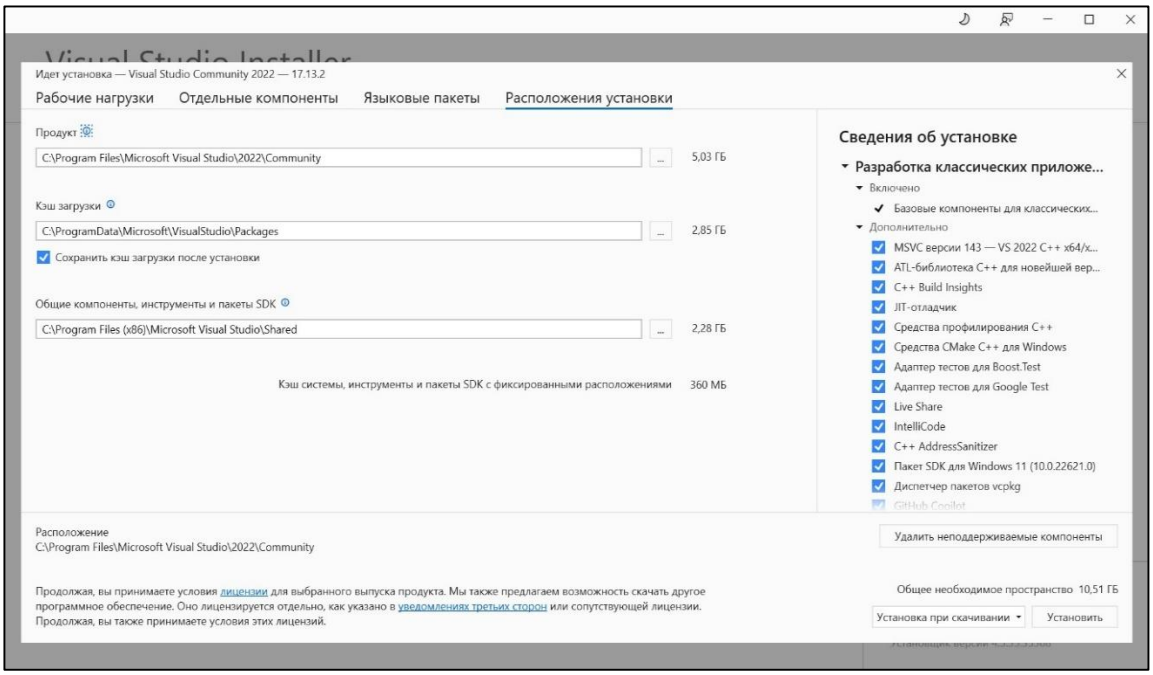


Рисунок 7 – Выбор путей загрузки

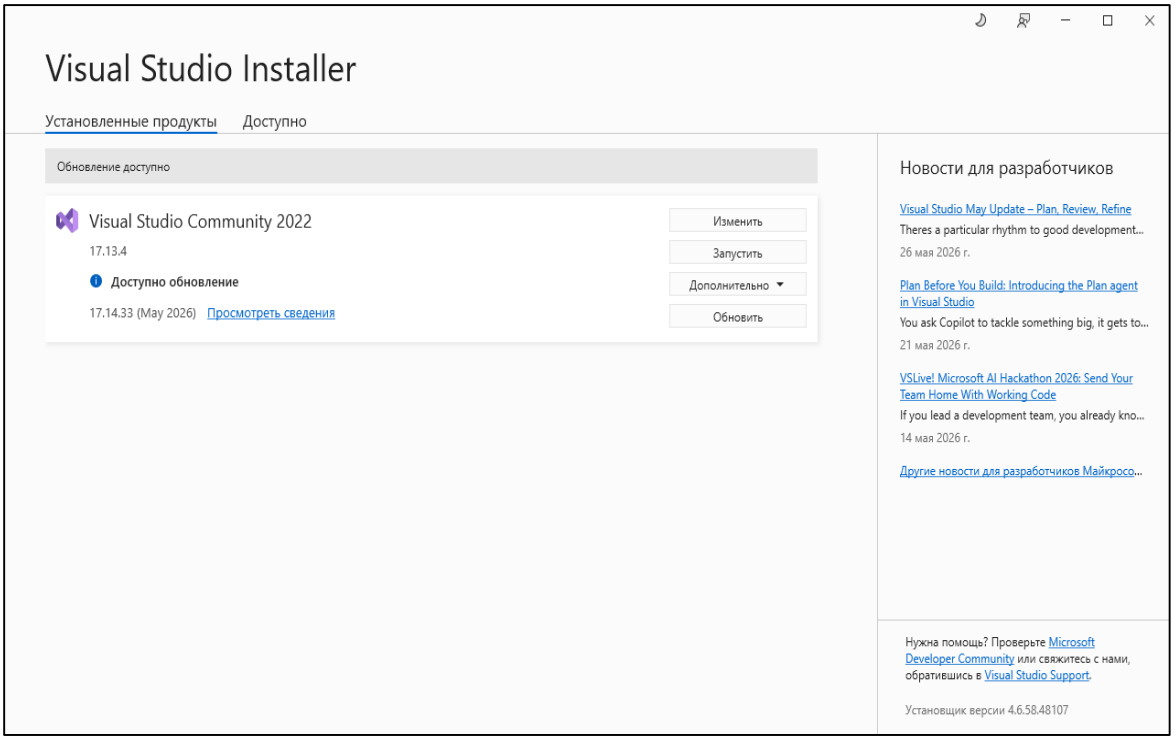


Рисунок 8 – Готовая установка Visual Studio 2022

После завершения установки Visual Studio 2022 перейдём к созданию и настройке проекта. Поскольку стандартные библиотеки Visual Studio не включают средства для работы с данной БД, необходимо скачать исходные файлы Microsoft SQL Server с официального сайта. Подробная настройка проекта представлена на рисунках 9 – 16.

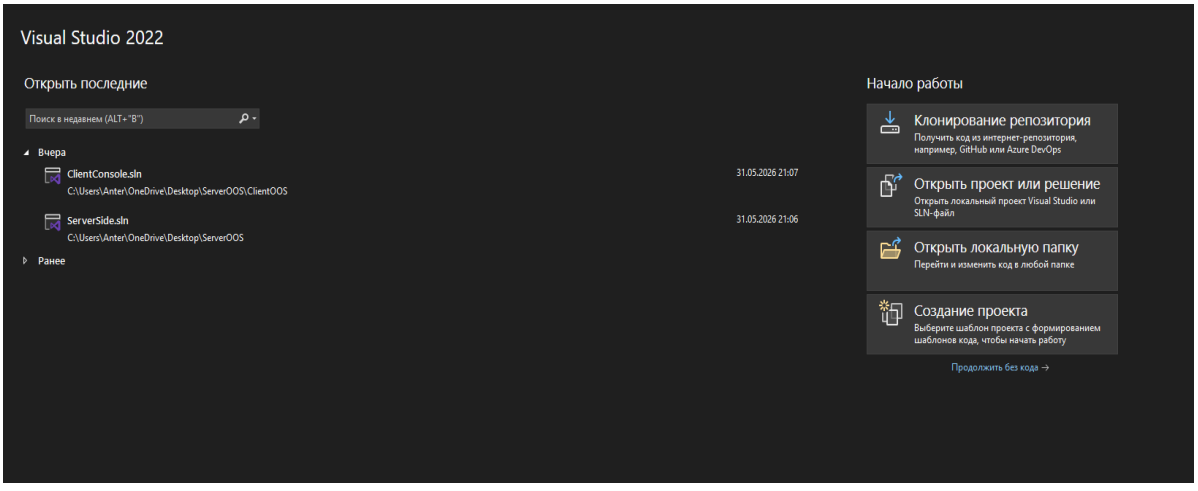


Рисунок 9 – Первый запуск Visual Studio 2022

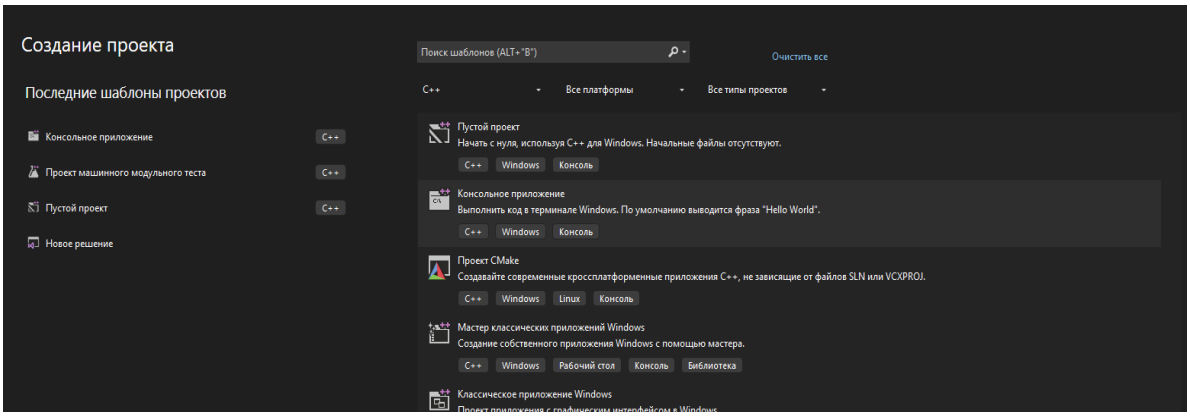


Рисунок 10 – Создание проекта

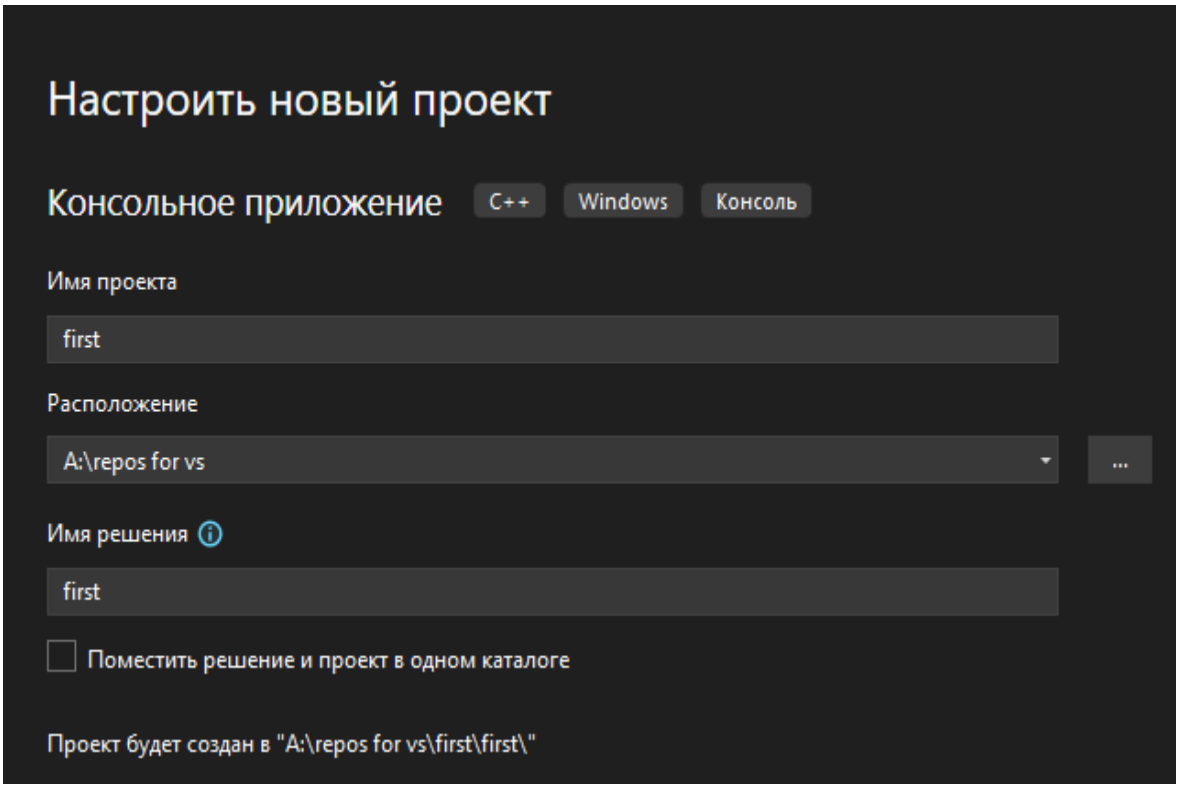


Рисунок 11 – Указание расположения и названия проекта

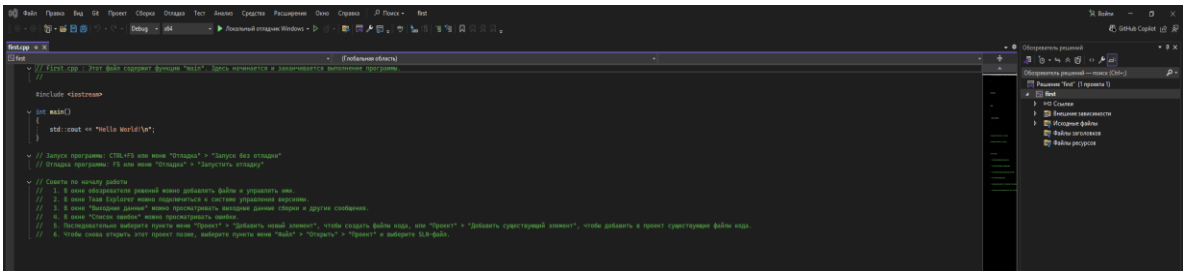
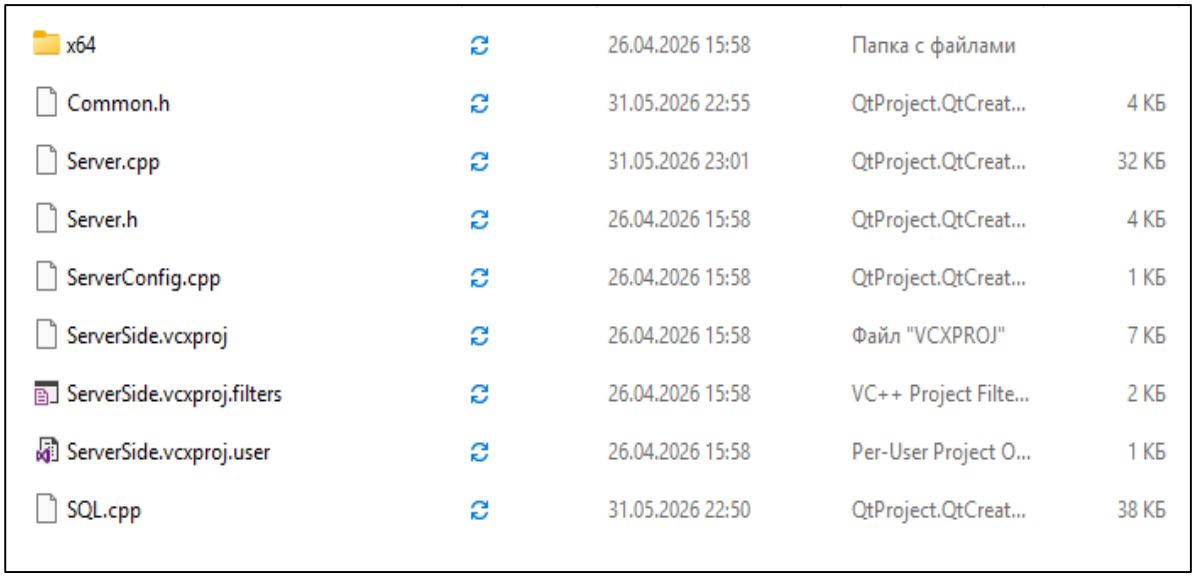
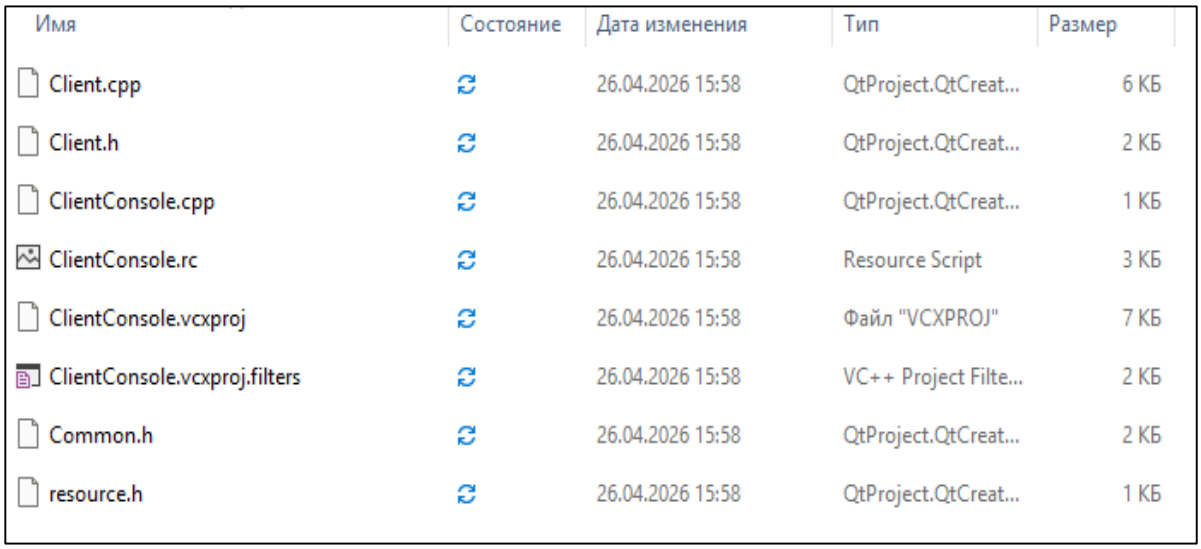


Рисунок 12 – Создание первого проекта



x64		26.04.2026 15:58	Папка с файлами	
Common.h		31.05.2026 22:55	QtProject.QtCreat...	4 КБ
Server.cpp		31.05.2026 23:01	QtProject.QtCreat...	32 КБ
Server.h		26.04.2026 15:58	QtProject.QtCreat...	4 КБ
ServerConfig.cpp		26.04.2026 15:58	QtProject.QtCreat...	1 КБ
ServerSide.vcxproj		26.04.2026 15:58	Файл "VCXPROJ"	7 КБ
ServerSide.vcxproj.filters		26.04.2026 15:58	VC++ Project Filte...	2 КБ
ServerSide.vcxproj.user		26.04.2026 15:58	Per-User Project O...	1 КБ
SQL.cpp		31.05.2026 22:50	QtProject.QtCreat...	38 КБ

Рисунок 13 – Копирование файлов сервера



Имя	Состояние	Дата изменения	Тип	Размер
Client.cpp		26.04.2026 15:58	QtProject.QtCreat...	6 КБ
Client.h		26.04.2026 15:58	QtProject.QtCreat...	2 КБ
ClientConsole.cpp		26.04.2026 15:58	QtProject.QtCreat...	1 КБ
ClientConsole.rc		26.04.2026 15:58	Resource Script	3 КБ
ClientConsole.vcxproj		26.04.2026 15:58	Файл "VCXPROJ"	7 КБ
ClientConsole.vcxproj.filters		26.04.2026 15:58	VC++ Project Filte...	2 КБ
Common.h		26.04.2026 15:58	QtProject.QtCreat...	2 КБ
resource.h		26.04.2026 15:58	QtProject.QtCreat...	1 КБ

Рисунок 14 – Копирование клиентских файлов

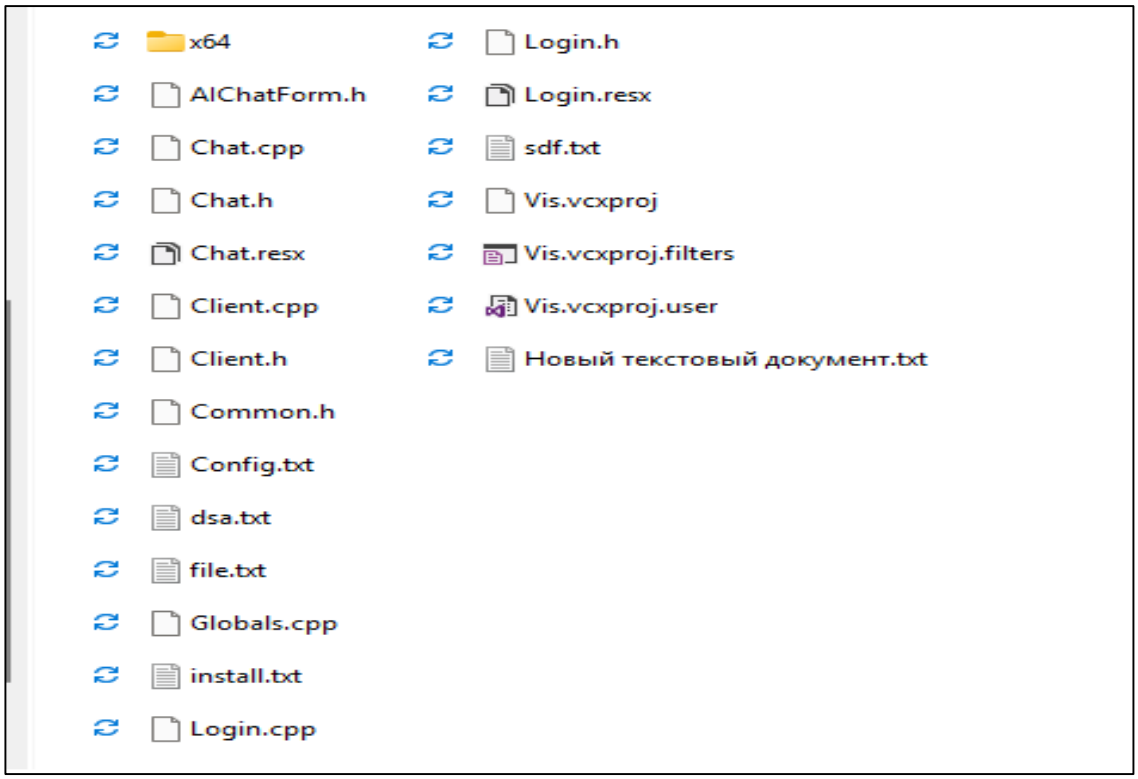


Рисунок 15 – Копирование клиентских GUI файлов

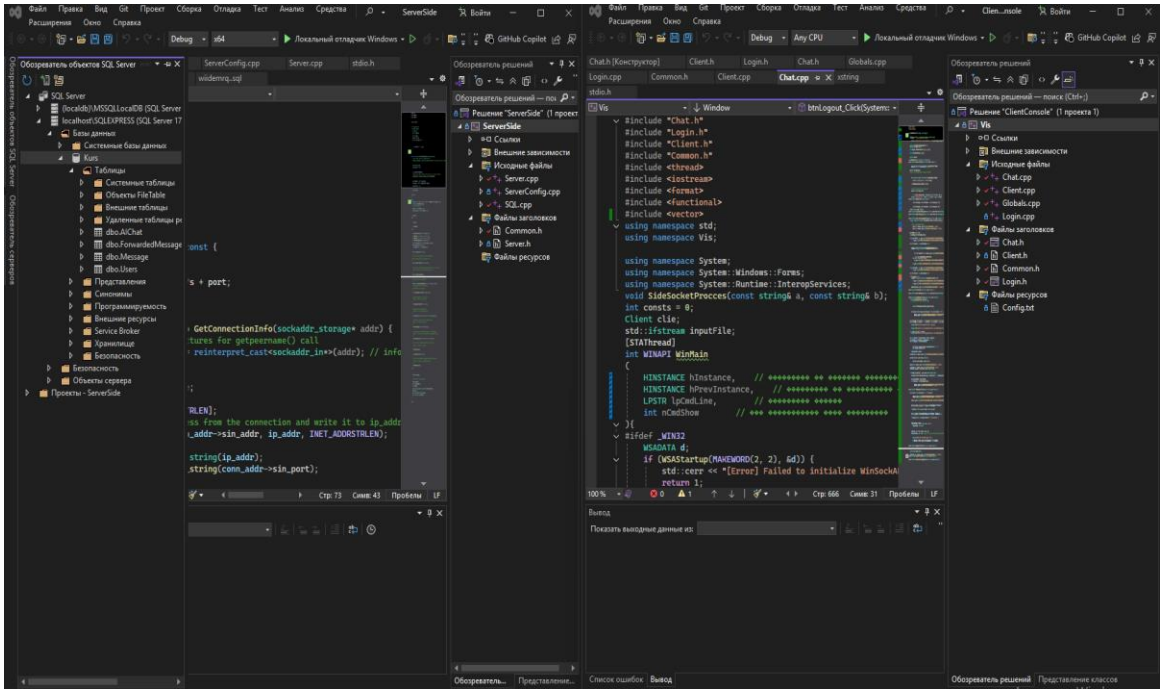


Рисунок 16 – Готовый проект

1.2.3 Язык программирования

C++ выбран в качестве основного языка реализации серверной и клиентской частей приложения, поскольку он обеспечивает высокую производительность, гибкое управление памятью и прямой доступ к системным API Windows.

Для распределенной интеллектуальной системы это критично: приложение должно стабильно работать с сокетами, обрабатывать множество последовательных запросов, выполнять сериализацию и разбор протокольных сообщений, а также взаимодействовать с БД без избыточных накладных расходов. По сравнению с высокоуровневыми языками (например, Python или JavaScript) C++ позволяет точнее контролировать ресурсопотребление и поведение низкоуровневых компонентов, что особенно важно для серверной части.

По сравнению с C# и Java, C++ дает более прямую работу с нативными библиотеками Windows в рамках выбранной архитектуры проекта. Дополнительным аргументом является то, что существующая кодовая база уже реализована на C++, включая серверные модули и оба клиента, поэтому выбор данного языка обеспечивает целостность проекта и снижает затраты на сопровождение.

1.2.4 Сетевой стек WinSock2

WinSock2 используется как базовый сетевой стек для реализации TCP-взаимодействия между сервером и клиентами.

На его основе реализованы подключение к серверу, прием и отправка сообщений, обработка ошибок соединения, а также поддержка непрерывного сетевого обмена в рамках пользовательских сессий.

Выбор WinSock2 обоснован целевой платформой Windows и глубокой интеграцией с C++-проектами в Visual Studio. Использование единого сетевого стека в ServerSide, ClientConsole и Vis уменьшает риск несовместимости при обмене данными и упрощает отладку протокола. Альтернативные кроссплатформенные библиотеки могли бы повысить переносимость, однако для

текущей версии проекта приоритетом были предсказуемость поведения и минимальная сложность конфигурации в Windows-среде.

1.2.5 СУБД Microsoft SQL Server

Microsoft SQL Server выбран в качестве СУБД для хранения учетных записей пользователей, истории сообщений, пересланных сообщений и данных AI-диалогов. Для распределенной интеллектуальной системы важны надежность хранения, целостность данных и возможность выполнения структурированных запросов к истории переписки, и SQL Server эффективно закрывает эти требования.

Ключевыми преимуществами выбора являются развитые средства администрирования, стабильная работа в экосистеме Windows, поддержка локальных сценариев развертывания (включая LocalDB), а также удобная интеграция с ODBC и Visual Studio.

По сравнению с SQLite SQL Server лучше подходит для многопользовательской клиент-серверной модели, где требуется централизованное хранилище и одновременная работа нескольких клиентов.

По сравнению с MySQL/PostgreSQL в рамках данного проекта SQL Server дает более естественную интеграцию с текущим стеком Windows.

1.2.6 Технология доступа к БД ODBC

Одним из наиболее распространённых средств, позволяющих унифицировать взаимодействие с различными СУБД, является интерфейс ODBC [1], ODBC представляет собой интерфейс доступа к базам данных в среде Windows, обеспечивающий независимость клиентского приложения от типа используемой СУБД. Доступ к БД осуществляется через специальный ODBC-драйвер, транслирующий запросы на язык конкретной СУБД. Для установления соединения технология использует ODBC-драйверы и источники данных (DSN – Data Source Name), которые содержат имя пользователя, пароль и иные параметры подключения. ODBC-драйвер реализован как динамически загружаемая библиотека, своя для каждого типа СУБД.

ODBC используется как стандартный интерфейс доступа C++-приложения к SQL Server и выступает связующим звеном между прикладной логикой сервера и системой хранения данных. Через ODBC в проекте реализованы ключевые операции: регистрация и авторизация пользователей, запись сообщений, выборка истории переписки, а также обработка служебных сценариев, связанных с AI-диалогами.

Выбор ODBC обусловлен его зрелостью, широкой поддержкой в Windows и хорошей совместимостью с SQL Server.

Для текущей архитектуры это практичное решение, которое позволяет прозрачно управлять SQL-запросами и поддерживать существующий код без внедрения дополнительного ORM-слоя. При этом в перспективе развития проекта целесообразно усилить безопасность SQL-взаимодействия (параметризованные запросы, централизованная обработка ошибок), что дополнительно повысит надежность серверной части.

1.3 Выбор операционной системы

1.3.1 ОС Windows

Плюсы:

– нативная поддержка WinSock2, WinHTTP и ODBC;

– удобная разработка и отладка в Visual Studio;

– хорошая интеграция с SQL Server;

– поддержка существующей структуры проекта на основе .sln/.vcxproj без дополнительной миграции;

– минимальные трудозатраты на настройку окружения для запуска сервера и клиентов;

– предсказуемое поведение Windows-специфичных библиотек в реальных сценариях эксплуатации.

Минусы:

– меньшая гибкость серверной автоматизации по сравнению с Linux;

– более высокие требования к ресурсам на некоторых конфигурациях;

– зависимость проекта от Windows-экосистемы, что ограничивает переносимость без доработок.

Дополнительно следует отметить, что Windows в данном проекте фактически является "родной" платформой, так как ключевые компоненты (сетевой стек, инструменты сборки и часть системных API) изначально подбирались именно под эту ОС. Это снижает число технических рисков на этапах реализации и тестирования.

1.3.2 ОС Linux

Плюсы:

– высокая стабильность, развитые средства автоматизации и мониторинга;

– отсутствие лицензионных затрат для большинства дистрибутивов;

– широкие возможности для контейнеризации и CI/CD-процессов;

– удобные инструменты администрирования для долгосрочной серверной эксплуатации;

– хорошая масштабируемость при росте нагрузки.

Минусы:

– текущая реализация опирается на Windows-специфичные API;

– для переноса потребуется переработка части кода;

– необходимо заменить или адаптировать модули, связанные с WinHTTP и отдельными Windows-зависимыми вызовами;

– потребуется дополнительное тестирование протокола и поведения клиентов в новой среде;

– увеличиваются сроки внедрения из-за миграции и повторной отладки.

Таким образом, Linux может быть перспективной целевой платформой для будущих версий, однако для текущего этапа проекта его использование связано с существенными затратами на портирование.

1.3.3 ОС macOS

Плюсы:

– удобная среда для разработки;

– стабильный UNIX-подобный инструментарий;

– качественная экосистема средств разработки и отладки;

– удобство локальной разработки для отдельных пользовательских сценариев.

Минусы:

– также требуется адаптация Windows-зависимых компонентов;

– ограниченная практическая целесообразность для промышленного серверного развертывания в данном проекте;

– потребуется отдельная валидация совместимости клиентских и серверных модулей;

– дополнительная сложность сопровождения при смешанной инфраструктуре.

С учетом архитектуры и выбранного стека технологий macOS может использоваться как вспомогательная среда разработки, но не как основной целевой контур развертывания для текущей версии системы.

Обоснование выбора – ОС Windows выбрана как наиболее практичная платформа для текущей реализации, так как сервер и клиенты используют WinSock2, а сервер дополнительно использует WinHTTP и ODBC. Это позволяет избежать лишних затрат на портирование, ускоряет отладку и упрощает сопровождение проекта.

1.4 Архитектура приложения

1.4.1 Общая клиент-серверная архитектура

Приложение построено по модели "клиент-сервер". Сервер принимает подключения и централизованно обрабатывает команды, клиенты формируют пользовательские запросы и отображают результат. При выполнении пользовательского действия на стороне клиента (например, авторизация, отправка

сообщения, запрос истории или передача файла) формируется пакет прикладного протокола, который передается серверу по TCP-соединению.

Сервер выполняет разбор полученной команды, запускает соответствующую бизнес-логику и, при необходимости, обращается к подсистеме хранения данных или внешнему AI API, после чего возвращает ответ инициатору операции и, если это требуется сценарием, целевому получателю сообщения.

На клиентской стороне полученный ответ интерпретируется, интерфейс обновляется, а пользователь получает результат операции в удобной форме. Выбранная архитектура обеспечивает централизованный контроль логики взаимодействия, единые правила аутентификации и обработки данных, согласованное хранение истории переписки и предсказуемую маршрутизацию сообщений между пользователями. Кроме того, такой подход упрощает развитие проекта: расширение протокола и добавление новых команд могут выполняться без изменения базовой схемы работы системы. Структурно система включает три взаимосвязанных компонента: серверную часть ServerSide, консольный клиент ClientOOS/ClientConsole и графический клиент ClientOOS/Vis.

1.4.2 Архитектура серверной части

Серверная часть реализует прием подключений, обработку сокетов, маршрутизацию команд протокола, выполнение операций с базой данных и взаимодействие с внешним AI API. Внутренняя организация сервера построена как логическое разделение на взаимосвязанные уровни. Сетевой уровень отвечает за создание сокета прослушивания, прием клиентских соединений и базовый обмен пакетами. Протокольный уровень выполняет сериализацию и десериализацию сообщений, а также определяет тип полученной команды.

Прикладной уровень содержит бизнес-логику авторизации, обработки текстовых и файловых сообщений, получения истории переписки и работы с AI-запросами. Уровень хранения инкапсулирует SQL-операции и обеспечивает взаимодействие с SQL Server через ODBC.

Подобная организация сервера позволяет разделить ответственность между подсистемами и упростить сопровождение проекта: изменения в механизмах хранения данных оказывают минимальное влияние на сетевой контур, а расширение протокола может выполняться локально на уровне обработчиков без переработки основного серверного цикла.

1.4.3 Архитектура клиентской части

Клиентская часть представлена двумя приложениями:

– консольный клиент (ClientConsole): предназначен для базового обмена сообщениями и отладки протокола и реализации подключения, отправки и прием сообщений в терминале.

При создании консольного клиента использовались подходы к построению консольных приложений в интегрированной среде Microsoft Visual Studio [2]. Данное пособие регламентирует базовые принципы организации проекта, настройки среды и отладки, что позволило обеспечить единообразную структуру клиентского кода в рамках принятой в проекте среды разработки.

Консольный клиент выполняет роль эталонного минимального клиента:

– позволяет быстро проверить доступность сервера и корректность TCP-подключения;

– удобен для низкоуровневой отладки протокольного обмена;

– используется при проверке реакций сервера на разные варианты входных команд.

– графический клиент (Vis): предоставляет формы авторизации, чаты, поддержку работы с AI-чатом и также ориентирование на основной пользовательский сценарий.

Графический клиент является пользовательским интерфейсом системы:

– содержит экран авторизации, рабочее окно чата и элементы управления сообщениями;

– отображает входящие/исходящие сообщения и историю переписки в удобном виде;

– обеспечивает более естественный сценарий эксплуатации по сравнению с консольным режимом.

Оба клиента используют общий подход к сетевому взаимодействию и работают с единым серверным протоколом, что обеспечивает совместимость в смешанных сценариях (например, консольный и GUI-клиент одновременно).

1.4.4 Протокол обмена сообщениями

Применяется единый прикладной протокол поверх TCP [4]. Клиент формирует команду, сервер разбирает пакет, выполняет бизнес-логику и возвращает ответ.

В протоколе выделяются группы команд:

– аутентификационные команды (регистрация, вход, подтверждение статуса);

– команды обмена сообщениями (личные сообщения, история, пересылка, удаление);

– команды передачи дополнительных данных (файлы, служебные поля);

– команды AI-взаимодействия (запрос к AI и возврат ответа).

С технологической точки зрения протокол реализует:

– компактное представление команд и полезной нагрузки;

– возможность расширения списка команд без нарушения совместимости базового формата;

– единый механизм обработки как для ClientConsole, так и для Vis.

В протокол включены команды:

– регистрация и авторизация;

– отправка текстовых сообщений;

– передача файловых сообщений;

– пересылка и удаление сообщений;

– получение истории;

– обращение к AI-чату.

1.4.5 Описание файлов проекта

Серверные файлы:

– ServerSide.sln – solution серверной части в Visual Studio.

– ServerSide/ServerSide.vcxproj – параметры сборки, подключаемые исходники и зависимости сервера;

– ServerSide/ServerSide.vcxproj.filters – структура отображения файлов в IDE;

– ServerSide/ServerConfig.cpp – точка входа и стартовая конфигурация сервера;

– ServerSide/Server.cpp – главный модуль сетевого цикла, обработчиков команд и маршрутизации;

– ServerSide/SQL.cpp – модуль доступа к SQL Server и AI API;

– ServerSide/Common.h – общий заголовок с протокольными структурами и константами.

Клиентские файлы (консольный клиент):

– ClientOOS/ClientConsole.sln – solution клиентской части;

– ClientOOS/ClientConsole/ClientConsole.cpp – точка входа консольного клиента и запуск соединения;

– ClientOOS/ClientConsole/Client.cpp – подключение к серверу, отправка/прием сообщений, рабочий цикл клиента;

– ClientOOS/ClientConsole/Client.h – интерфейс класса клиента и объявления основных методов;

– ClientOOS/ClientConsole/Common.h – общие сетевые определения и константы для консольного клиента;

– ClientOOS/ClientConsole/ClientConsole.vcxproj – параметры сборки консольного клиента.

Клиентские файлы (графический клиент):

– ClientOOS/Vis/Vis.vcxproj – параметры сборки графического клиента;

– ClientOOS/Vis/Client.cpp, ClientOOS/Vis/Client.h – сетевой слой GUI- клиента, обмен данными с сервером;

– ClientOOS/Vis/Login.h – форма и логика авторизации пользователя;

– ClientOOS/Vis/Chat.h – основная форма чата и пользовательские действия в диалоге;

– ClientOOS/Vis/AIChatForm.h – форма взаимодействия с AI-чатом;

– ClientOOS/Vis/Common.h – общие структуры и константы протокола для GUI-клиента;

– ClientOOS/Vis/Globals.cpp – глобальные переменные/состояния, используемые в клиенте;

– ClientOOS/Vis/Config.txt – конфигурационные параметры подключения и запуска.

1.4.6 Реализация AI-ассистента

Реализация AI-ассистента в проекте построена по клиент-серверному принципу и встроена в общий протокол обмена сообщениями [3]. Пользователь формирует запрос в клиентском приложении (в графическом клиенте через форму ClientOOS/Vis/AIChatForm.h, а на уровне серверной логики через соответствующую команду протокола), после чего запрос передается на сервер по стандартному TCP-каналу.

На серверной стороне запрос обрабатывается как отдельный сценарий прикладной логики. Модуль ServerSide/Server.cpp принимает команду AI-взаимодействия, выполняет разбор входных данных и передает их в подсистему интеграции, реализованную в ServerSide/SQL.cpp. В данном модуле формируется HTTP-запрос к внешнему AI API, отправляется запрос, принимается ответ и выполняется его постобработка для последующей передачи клиенту в формате внутреннего протокола.

Дополнительно сервер может сохранять историю AI-диалогов в базе данных SQL Server, что позволяет обеспечить целостность пользовательской сессии, повторный доступ к предыдущим сообщениям и единый механизм журналирования взаимодействий. Это особенно важно в распределенной

интеллектуальной системы, где необходимо контролировать историю обращений и поддерживать воспроизводимость рабочих сценариев.

С архитектурной точки зрения AI-ассистент реализован как расширение существующей системы, а не как изолированный внешний модуль. Такой подход позволил избежать дублирования сетевой инфраструктуры, сохранить единый механизм авторизации и централизованную обработку команд, а также упростить масштабирование функциональности в будущем (например, добавление новых AI-сценариев или смена внешнего провайдера API без изменения клиентской логики).

1.5 Реализация приложения

1.5.1 Реализация серверной части

Сервер реализован как монолитный обработчик подключений с диспетчеризацией команд.

Основные этапы реализации:

– запуск сервера и инициализация сетевого стека;

– обработка подключения клиентов;

– разбор команд протокола;

– обращение к SQL Server через ODBC;

– формирование и отправка ответов клиентам.

1.5.2 Реализация консольного клиента ClientConsole

Консольный клиент реализует базовую работу с сервером:

– запуск с параметрами адреса и порта;

– подключение к серверу;

– отправка введенных пользователем сообщений;

– прием данных от сервера в отдельном рабочем цикле.

Назначение клиента - тестирование и отладка сетевого протокола на ранних этапах.

1.5.3 Реализация графического клиента Vis

Графический клиент предоставляет пользовательский интерфейс:

– окно входа пользователя;

– окно чата с отправкой сообщений;

– отдельную форму для AI-ассистента.

Клиент обеспечивает более удобный рабочий сценарий по сравнению с консольным вариантом и предназначен для повседневного использования в рамках демонстрации проекта.

1.5.4 Интеграция AI-ассистента

Интеграция AI- ассистента реализовывалась как отдельный прикладной сценарий внутри существующей клиент-серверной архитектуры. На клиентской стороне пользователь формирует текст запроса в интерфейсе чата, после чего клиент упаковывает его в команду протокола и отправляет на сервер по TCP.

На сервере входящая команда обрабатывается в общем диспетчере сообщений: запрос валидируется, привязывается к текущему пользователю и передается в модуль интеграции с внешним AI API.

Далее сервер формирует HTTP-запрос в требуемом формате, отправляет его провайдеру AI-сервиса, принимает ответ и выполняет постобработку результата (проверка структуры, выделение текстовой части, подготовка к отправке клиенту). После этого итоговый AI-ответ преобразуется в формат внутреннего протокола и возвращается инициатору запроса. Таким образом сохраняется единая модель взаимодействия, где для клиента AI-ответ выглядит как стандартное сообщение от сервера.

При необходимости сервер также сохраняет историю AI-диалога в SQL Server, что позволяет восстанавливать контекст общения, анализировать предыдущие обращения и обеспечивать целостность данных в рамках пользовательской сессии. Реализация выполнялась поэтапно: сначала был настроен базовый запрос-ответ с внешним API, затем добавлена обработка ошибок и только после этого интегрировано сохранение истории в базе данных.

Структура всех классов представлена на рисунке 17.

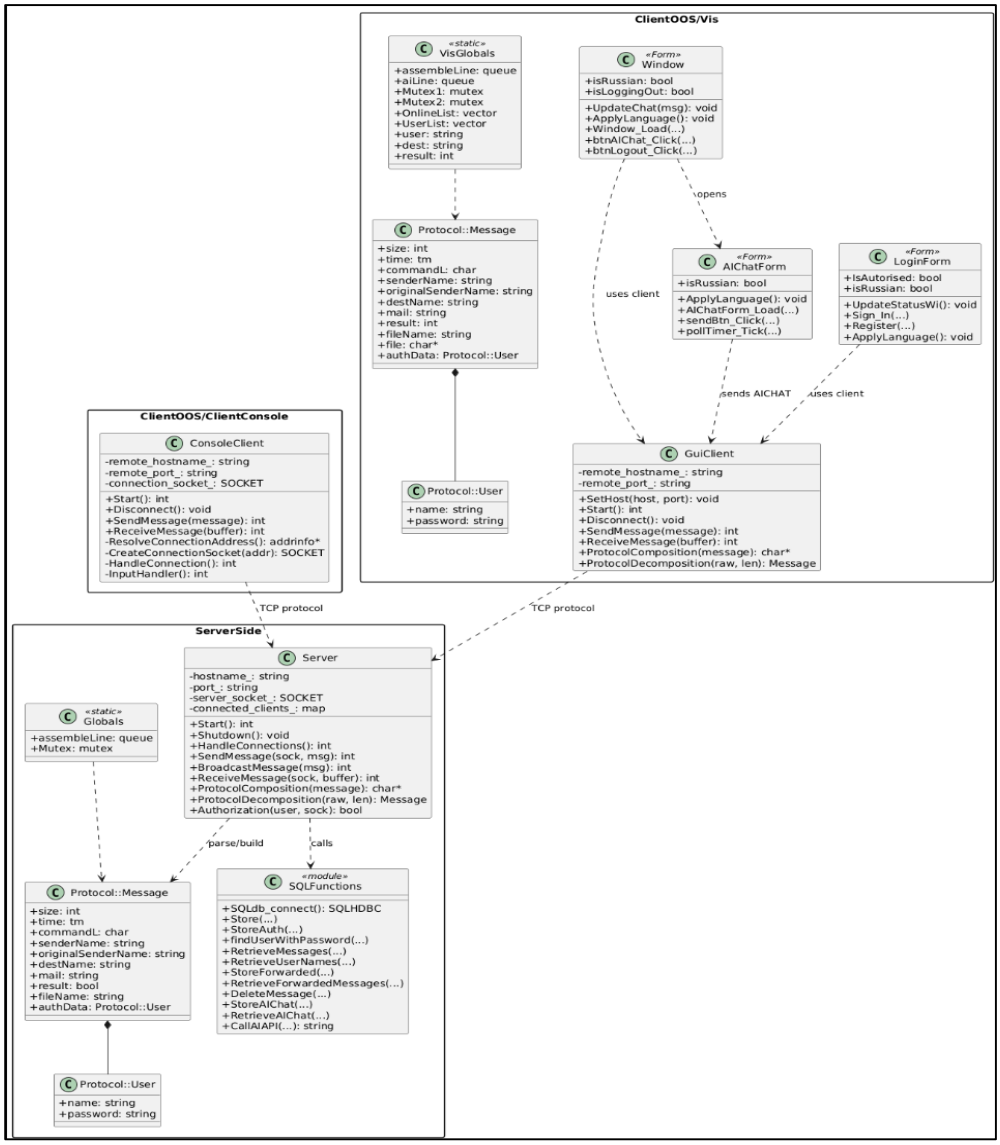


Рисунок 17 – UML-диаграмма всех классов приложения

1.5.5 Структура приложения

UML-диаграмма развёртывания представлена на рисунке 18.

Рисунок 18 – UML-диаграмма развёртывания

Принцип работы приложения основан на клиент-серверной архитектуре с централизованным хранением данных и интеграцией искусственного интеллекта.

Пользователь взаимодействует с программой через консольный клиент ClientConsole.exe либо графический клиент Vs.exe, работающие на рабочей станции под управлением Windows. Любое действие, будь то отправка сообщения, запрос истории или включение AI-функций, формирует запрос, который по протоколу TCP с использованием библиотеки WinSock2 направляется на серверную часть.

Сервер ServerSide.exe, развёрнутый на выделенном или локальном компьютере, принимает входящие соединения через свой TCP-сервер на заранее заданном порту (по умолчанию 127.0.0.1:4000, с возможностью работы по локальной сети или через интернет). При поступлении запроса сервер анализирует его тип: если требуется сохранить или извлечь данные, он через ODBC-слой формирует обращение к базе данных под управлением Microsoft SQL Server, где и хранится вся история переписки, учётные записи и служебная информация. При необходимости задействовать искусственный интеллект для генерации ответа или анализа текста сервер через HTTP-клиент WinHTTP отправляет HTTPS-запрос к внешнему API-сервису OpenRouter, получает результат и встраивает его в логику обработки.

После выполнения всех необходимых операций сервер формирует ответ и отправляет его обратно тому клиенту, который инициировал запрос. Клиентская часть принимает ответ, обрабатывает его и немедленно отображает изменения в пользовательском интерфейсе — обновляет список сообщений, уведомления или результат работы AI. Таким образом, вся бизнес-логика и работа с данными сосредоточены на серверной стороне, что обеспечивает единое хранилище, безопасность доступа и возможность одновременной работы нескольких клиентов с общей историей.

UML-диаграмма последовательности действий представлена на рисунке 19.

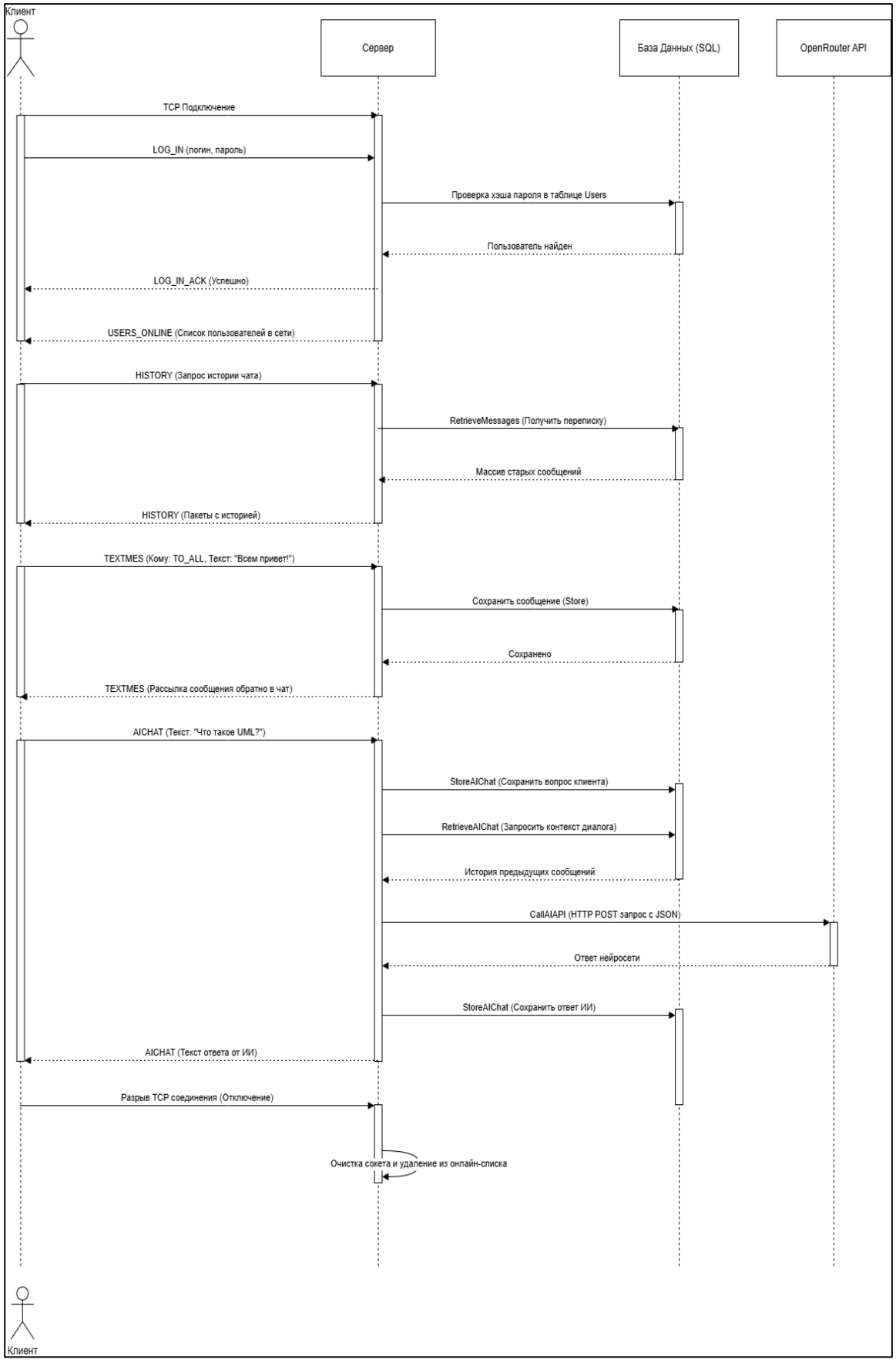


Рисунок 19 – UML-диаграмма последовательности

1.5.6 Структура базы данных

Для хранения данных приложения используется Microsoft SQL Server. Структура базы данных построена в соответствии с основными сценариями работы распределенной интеллектуальной системы: авторизация пользователей, обмен сообщениями, пересылка сообщений и работа AI-ассистента. В серверном коде (ServerSide/SQL.cpp) используются следующие таблицы:

– таблица Users, представленная в таблице 2. Данная таблица содержит учетные записи пользователей системы. В ней хранятся логин пользователя (username) и пароль с использованием хэширования SHA-256 (password), которые используются при регистрации и авторизации. Таблица участвует в проверке подлинности пользователя при входе и в формировании списка доступных пользователей в клиентской части;

– таблица Message, представленная в таблице 3. Данная таблица предназначена для хранения основной истории переписки между пользователями. В ней сохраняются служебные и прикладные поля сообщения, включая размер, время отправки, тип команды (commandL), имя отправителя (senderName), имя получателя (destName) и текст сообщения (mail). Записи этой таблицы используются для выдачи истории чата и восстановления диалога;

– таблица ForwardedMessage, представленная в таблице 4. Данная таблица хранит данные о пересланных сообщениях. Помимо стандартных полей сообщения, в ней дополнительно присутствует информация об оригинальном отправителе (originalSenderName), что позволяет сохранять контекст пересылки и корректно отображать источник сообщения в клиенте;

– таблица AIChat, представленная в таблице 5. Данная таблица предназначена для хранения истории взаимодействия пользователя с AI-ассистентом. В ней сохраняются имя пользователя (username), роль сообщения (role, например пользователь или AI) и текст сообщения (message). Данные таблицы используются для загрузки контекста AI-диалога и отображения истории в форме AI- чата.

Таблица 2 – Атрибуты сущности Users

Поле	Тип данных	Описание
ID	Int	Идентификатор пользователя
username	Varchar	Логин пользователя
password	Varchar	Пароль пользователя

Таблица 3 – Атрибуты сущности Message

Поле	Тип данных	Описание
ID	Int	Идентификатор сообщения
size	Int	Размер сообщения
time	Varchar	Время отправки
commandl	Char	Тип команды
senderName	Varchar	Имя отправителя
destName	Varchar	Имя получателя
mail	Varchar	Текст сообщения

Таблица 4 – Атрибуты сущности ForwardedMessage

Поле	Тип данных	Описание
ID	Int	Идентификатор сообщения
size	Int	Размер сообщения
time	Varchar	Время отправки
commandl	Char	Тип команды
senderName	Varchar	Имя отправителя
originalSenderName	Varchar	Имя оригинального отправителя
destName	Varchar	Имя получателя
mail	Varchar	Текст сообщения

Таблица 5 – Атрибуты сущности AIChat

Поле	Тип данных	Описание
ID	Int	Идентификатор пользователя
username	Varchar	Имя пользователя

Поле	Тип данных	Описание
role	Varchar	Роль сообщения
message	Varchar	Текст сообщения
time	Datetime	Дата отправки

Таким образом, структура БД покрывает все ключевые функциональные сценарии приложения: аутентификацию, обмен сообщениями, пересылку и AI-взаимодействие. Такое разделение на тематические таблицы упрощает сопровождение системы, повышает читаемость SQL-логики и позволяет масштабировать проект за счет расширения отдельных подсистем хранения данных.

1.6 Тестирование приложения

1.6.1 Цели и задачи тестирования

Цель – проверить корректность работы клиент-серверной системы в пользовательских сценариях.

Задачи:

– проверить подключение клиентов;

– проверить регистрацию и авторизацию;

– проверить доставку и сохранение сообщений;

– проверить передачу файлов и пересылку;

– проверить работу AI- ассистента.

1.6.2 Тестирование подключения и авторизации

Чтобы зайти в приложение нужно запустить сначала сервер, запуск представлен на рисунках 20 – 21.

```
[Debug] Resolving server hostname.
[Debug] Creating server socket object.
[Debug] Binding socket to the resolved address.
[Debug] Setting SO_REUSEADDR socket option to the server socket.
[Debug] Activating server listenning mode.
[Info] Server is listenning for incoming connections at 127.0.0.1:4000
Connected to database successfully!
```

Рисунок 20 – Запуск сервера

```
[Info] New connection from 127.0.0.1:22503
```

Рисунок 21 – Подключение клиента на сервер

После запуска сервера могут подключаться клиенты (рисунок 22).

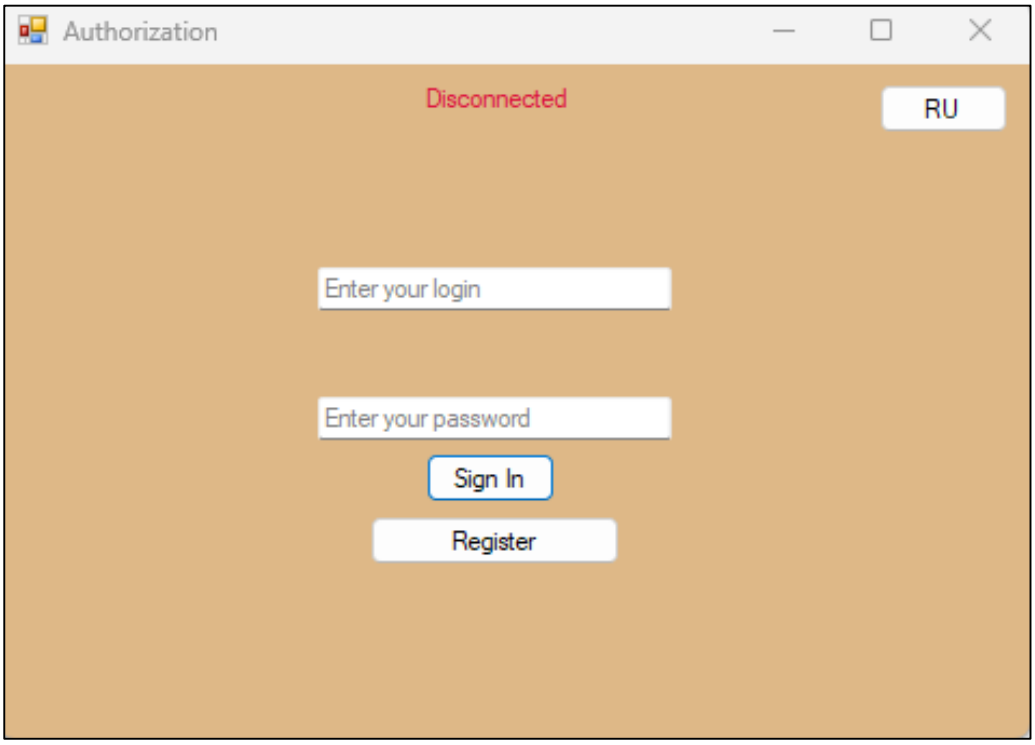


Рисунок 22 - Окно входа или регистрация нового пользователя

При необходимости можно поменять язык интерфейса (рисунки 23-24).

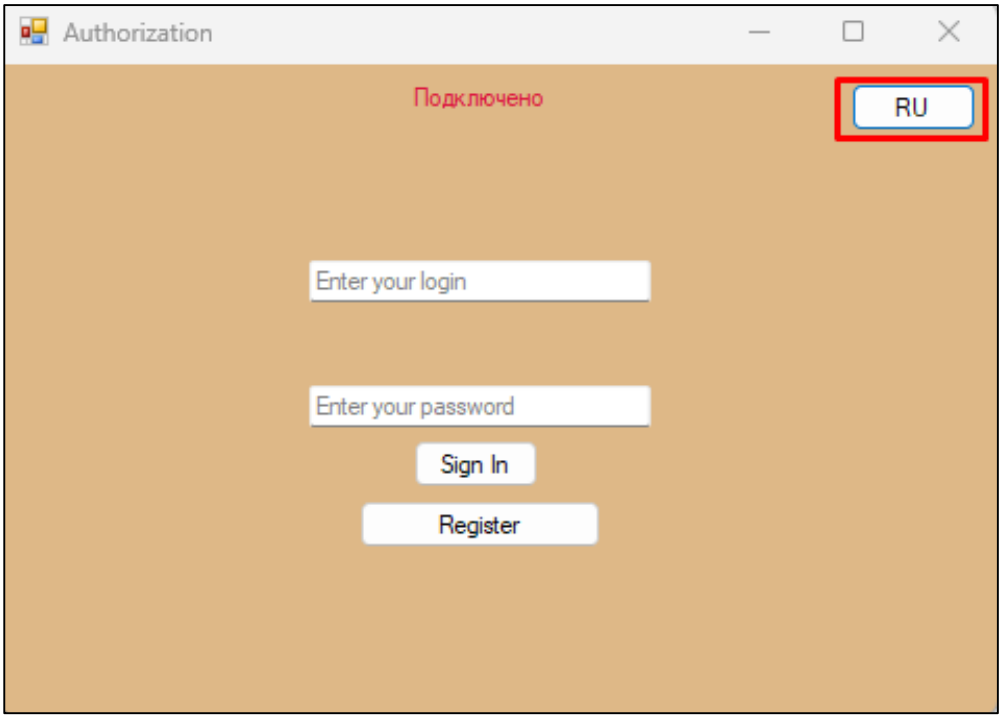


Рисунок 23 – Смена языка

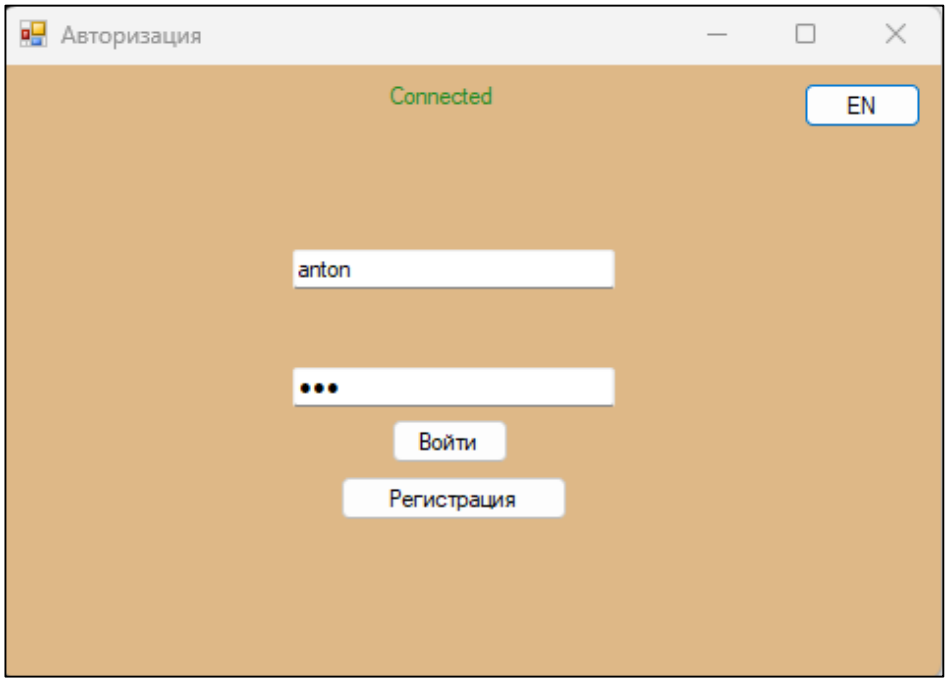


Рисунок 24 – Интерфейс на русском языке

Для регистрации пользователя необходимо ввести логин и пароль, после нажать на кнопку «Регистрация» (рисунок 25).

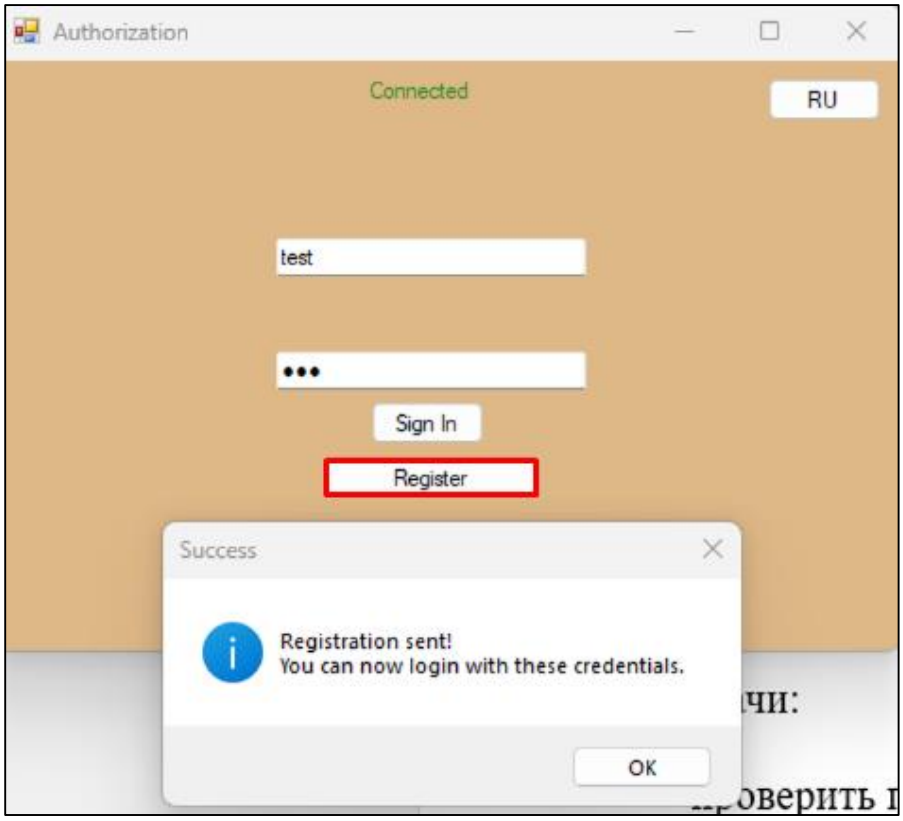


Рисунок 25 – Регистрация пользователя

Для входа необходимо ввести логин и пароль зарегистрированного пользователя, после нажать на кнопку «Войти» (рисунок 26).

Рисунок 26 – Авторизация пользователя

При авторизации или любых действий со стороны клинета ведётся лоигрование на сервере (рисунки 27-28).

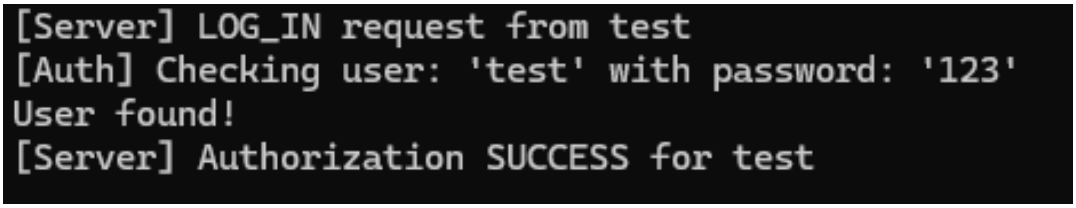


Рисунок 27 – Вход существующего пользователя (логи с сервера)

Рисунок 28 – Обработка ошибочных данных авторизации

1.6.3 Тестирование главного меню

Главное меню представлено на рисунке 29.

Рисунок 29 – Главное меню

Для того чтобы отобразить всех зарегистрированных пользователей необходимо нажать на кнопку «Обновить» (рисунок 30).

Рисунок 30 – Главное меню со всеми зарегистрированными пользователями

Для того чтобы найти какого-то пользователя по логину и написать ему, нужно нажать на поле поиска и ввести логин (рисунки 31 – 32).

Рисунок 31 – Отображение поле поиска пользователя

Рисунок 32 – Поиск пользователя по логину

Если необходимо выйти в меню авторизации, то нужно нажать на кнопку «Выйти» (рисунок 33).

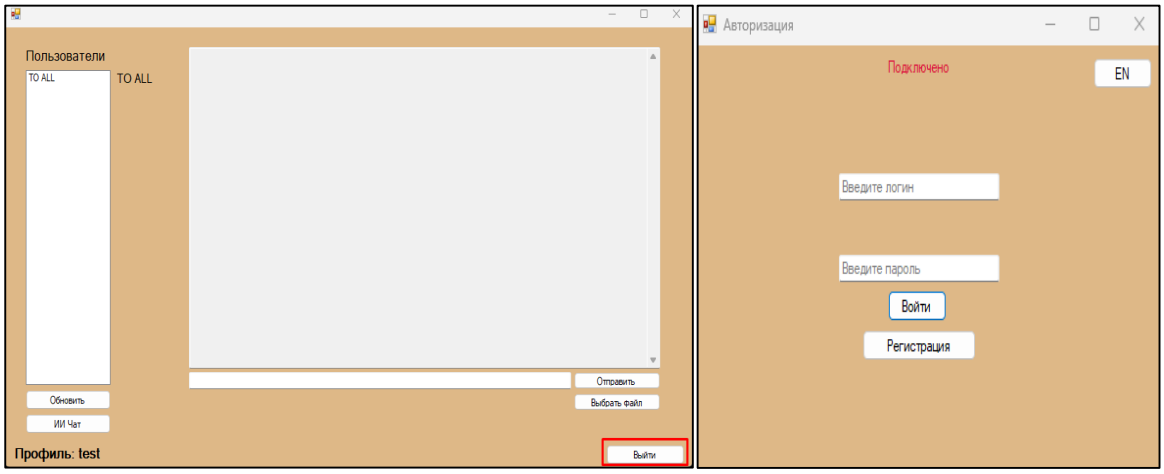


Рисунок 33 – Выход в главное меню

1.6.4 Тестирование обмена сообщениями

Обмен текстовыми сообщениями между двумя клиентами, получение сообщений в режиме реального времени (рисунок 34).

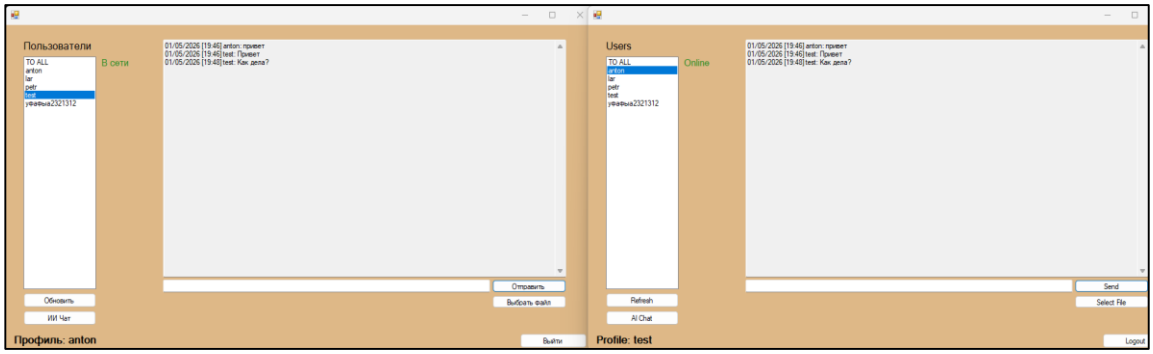


Рисунок 34 – Обмен сообщениями между пользователями

1.6.5 Тестирование передачи файлов и пересылки сообщений

Для отправки файловых сообщений необходимо нажать на кнопку «Выбрать файл», после выбрать нужный файл и нажать на кнопку открыть, после на кнопку «Отправить» (рисунок 35).

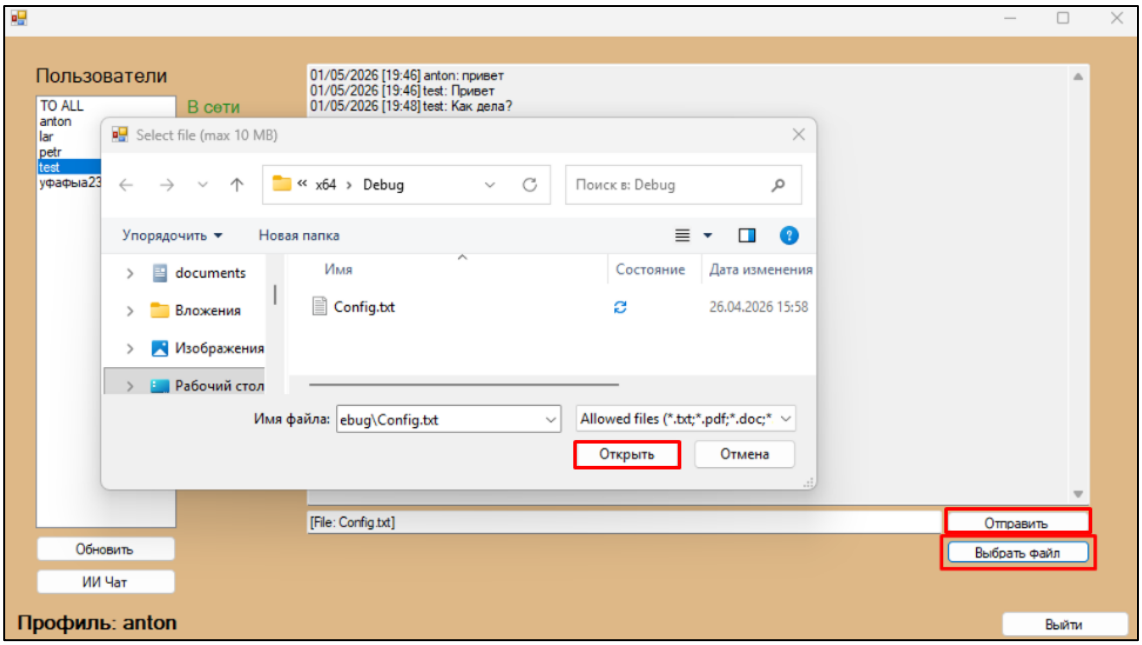


Рисунок 35 - Отправка файловых сообщений

У получателя файл открывается и сохраняется в «Загрузки» (рисунок 36).

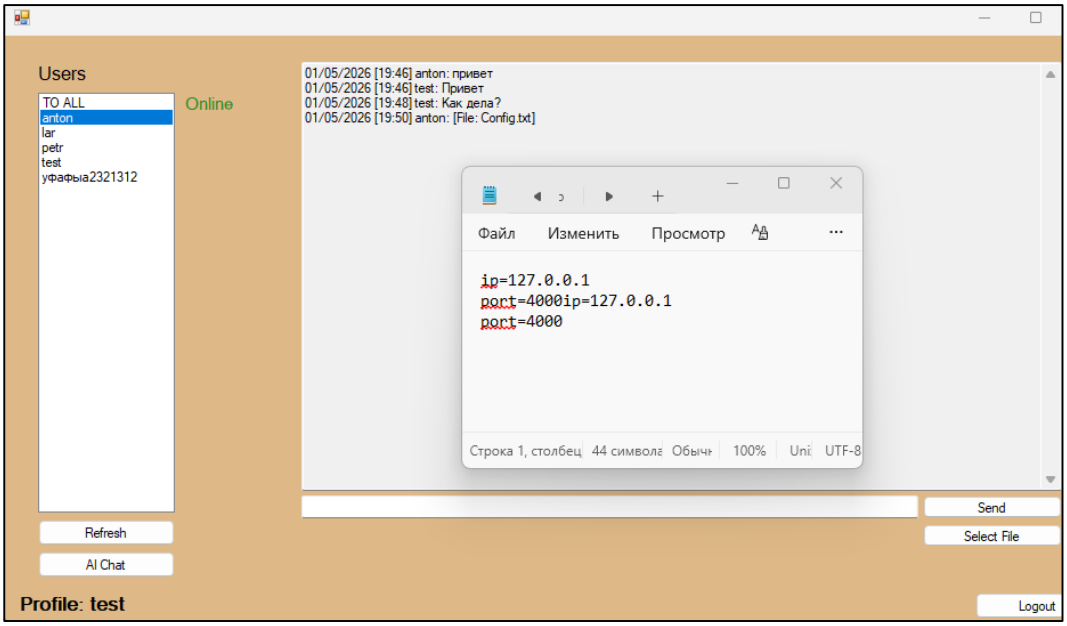


Рисунок 36 – Открытие файлового сообщения

Для пересылки сообщений нужно выделить полностью строку, которую необходимо переслать, после ПКМ по этой строке и нажать на кнопку «Переслать» (рисунок 37).

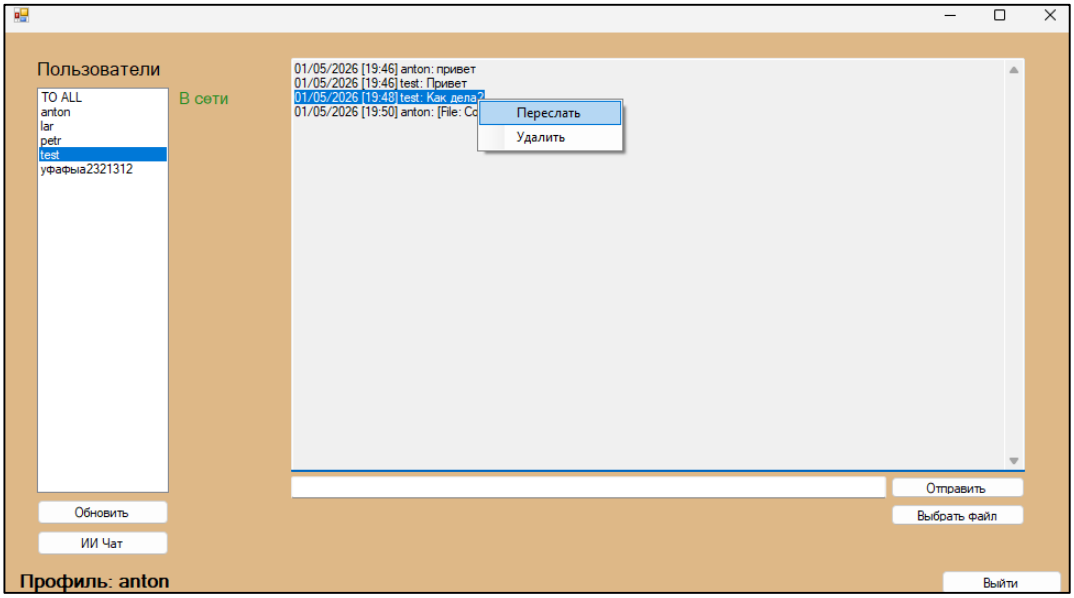


Рисунок 37 – Пересылка сообщений

В отрывшемся окне нужно выбрать кому хотите переслать сообщение, после нажать на «Переслать» (рисунок 38).

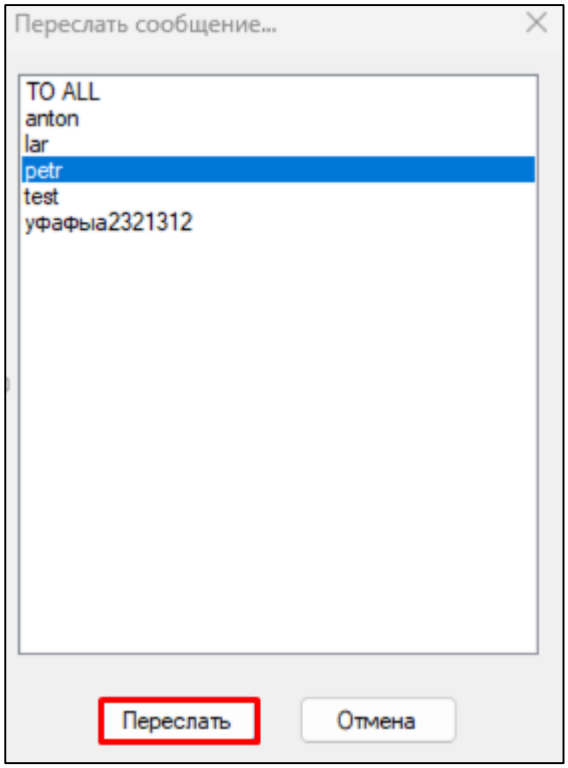


Рисунок 38 - кому можно переслать

После нажатия на кнопку переслать появляется окно с уведомлением о успешной отправки пересланного сообщения (рисунок 39).

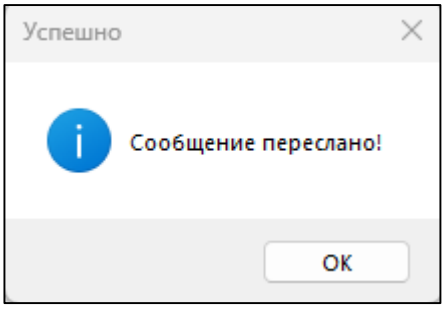


Рисунок 39 – Уведомление о успешной пересылке

На рисунке 40 предстaлено получение пересланного сообщения.

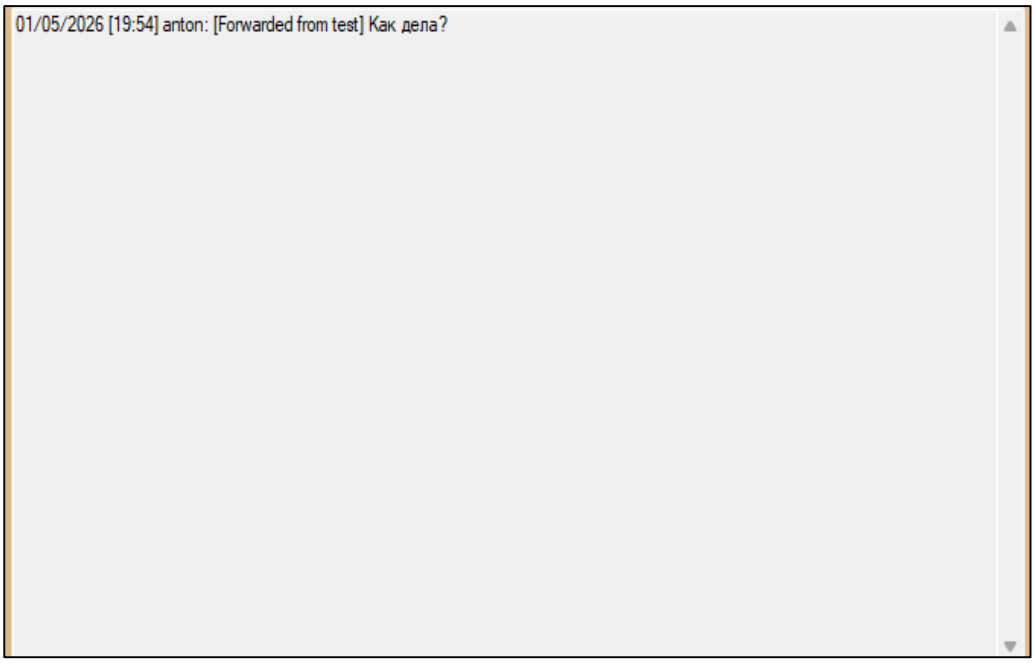


Рисунок 40 – полученное пересланное сообщение

Для удаления сообщения нужно также выделить всю строку, которую необходимо удалить, после ПКМ по этой строке и нажать на «Удалить» (рисунки 41 – 42).

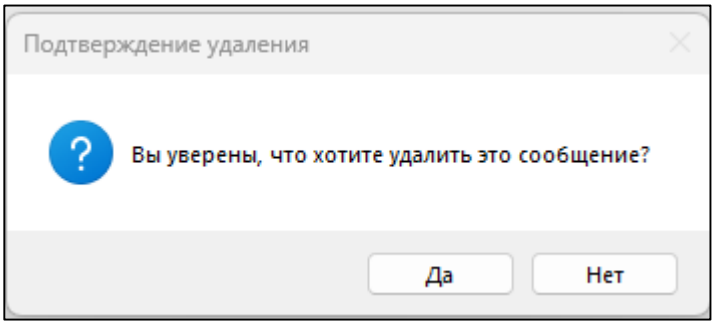


Рисунок 41 – Уведомление об удалении сообщения

Рисунок 42 – После удаления сообщения

1.6.6 Тестирование AI- ассистента

Для того чтобы зайти в AI чат необходимо нажать на кнопку «ИИ Чат» (рисунок 43).

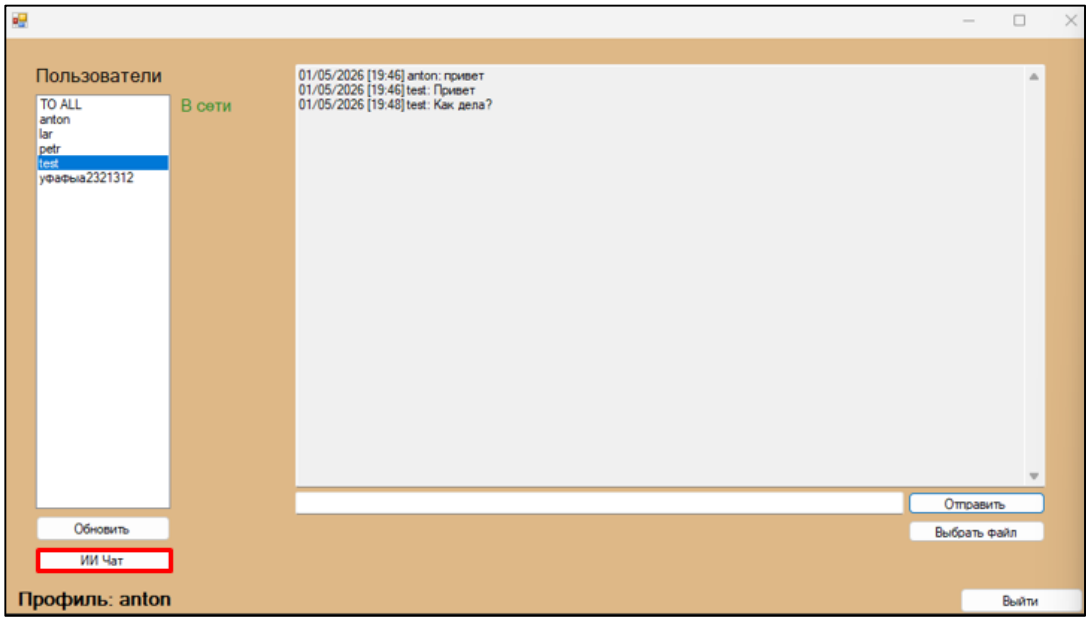


Рисунок 43 – Переход в ИИ Чат

Для отправки запроса необходимо ввести, что вы хотите в строку и нажать «Отправить» (рисунок 44). Ответ от ИИ представлен на рисунке 45.

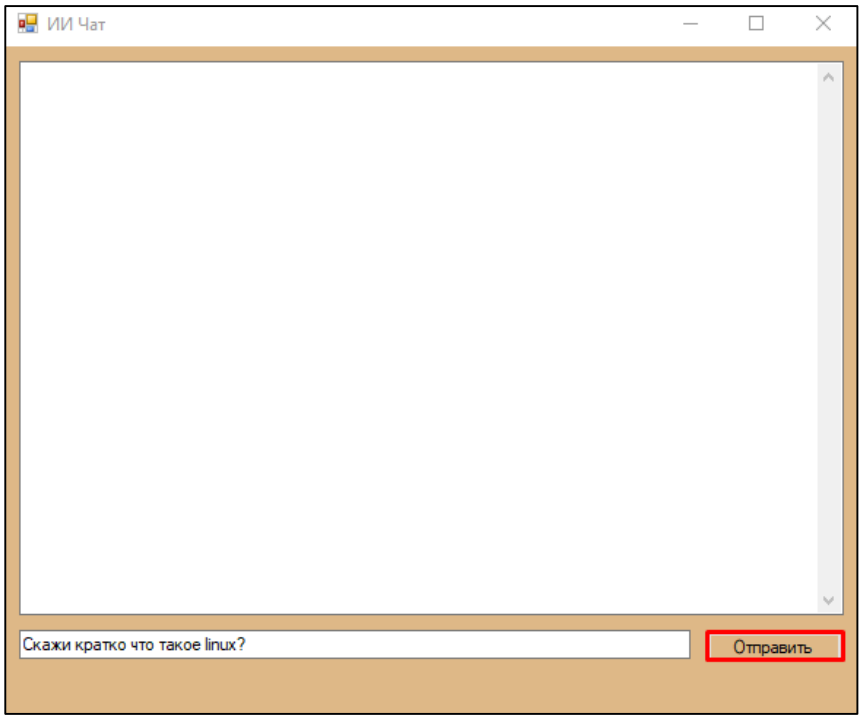


Рисунок 44 – Отправка запроса

На рисунке 45 представлен ответ от ИИ.

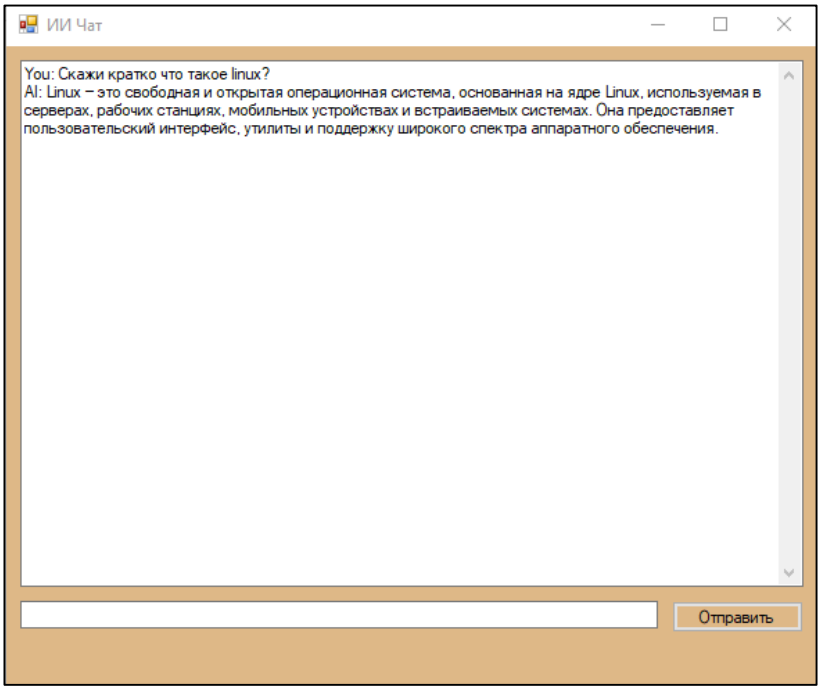


Рисунок 45 - Корректное отображение ответа в клиенте

2 Безопасность жизнедеятельности

2.1 Условия труда при разработке ПО

Процесс разработки программного обеспечения выполняется за персональным компьютером и классифицируется как интеллектуальный труд. В ходе такой работы человек подвергается ряду специфических нагрузок, среди которых преобладают интенсивная зрительная и высокая психоэмоциональная. Кроме того, специфика деятельности подразумевает длительное нахождение в статической сидячей позе, что создает дополнительную нагрузку на опорно-двигательный аппарат.

2.2 Анализ опасных и вредных факторов

К основным вредным факторам, возникающим в процессе разработки, относится длительная и непрерывная фокусировка на экране монитора, что в совокупности с высокими умственными нагрузками часто приводит к общему переутомлению. Дополнительную угрозу здоровью разработчика может представлять неудобная эргономика рабочего места, провоцирующая нарушения осанки и кровообращения. Помимо этого, присутствуют и опасные физические факторы, включающие риски поражения электрическим током при работе с оборудованием, а также потенциальную вероятность возгорания компьютерной техники из-за короткого замыкания или перегрева.

2.3 Требования к организации рабочего места

Для эффективного снижения профессиональных рисков и сохранения здоровья необходимо грамотно организовать рабочее пространство. Крайне важно использовать кресло с возможностью индивидуальной регулировки и подобрать правильную высоту рабочего стола. Монитор компьютера следует располагать на безопасном расстоянии от 50 до 70 сантиметров от глаз пользователя, а в самом помещении требуется обеспечить достаточное и равномерное освещение,

исключающее появление отвлекающих бликов на экране. Для снятия накопленного статического и зрительного напряжения настоятельно рекомендуется делать регулярные перерывы в работе и выполнять небольшую физическую разминку.

2.4 Электробезопасность и пожарная безопасность

При работе с вычислительной техникой требуется строгое соблюдение актуальных норм электро- и пожаробезопасности. В процессе эксплуатации оборудования необходимо использовать только полностью исправные соединительные кабели и блоки питания, категорически избегая чрезмерной перегрузки сетевых фильтров и розеток. Для предотвращения опасного перегрева аппаратных компонентов запрещается перекрывать или блокировать вентиляционные отверстия системных блоков и мониторов. Наконец, помимо общего соблюдения правил технической эксплуатации электроприборов, в рабочем помещении всегда должен быть обеспечен быстрый и беспрепятственный доступ к первичным средствам пожаротушения.

Заключение

В ходе работы была разработана распределенная интеллектуальная система под ОС Windows, включающая сервер, консольный и графический клиент. Реализованы ключевые функции: авторизация, обмен сообщениями, передача файлов, хранение истории и ИИ-чат.

Поставленная цель достигнута: создана работоспособная система, демонстрирующая полный цикл взаимодействия клиента и сервера. Разработанное решение может использоваться как основа для дальнейшего развития и внедрения дополнительных механизмов безопасности, масштабирования и мониторинга.

Список использованных источников

1. Технология odbc [Электронный ресурс] / Режим доступа: https://studfile.net/preview/11459579/page:30/, свободный. (дата обращения 28.04.2026).

2. Разработка консольных приложений с помощью Microsoft Visual Studio 2017: учебное пособие / Ю.В. Тракимус – Новосибирск: Изд-во НГТУ, 2018. – 3 с.

3. Реализация API. API-интерфейсы для самых маленьких [Электронный ресурс] / Режим доступа: https://habr.com/ru/articles/891816/, свободный. (дата обращения 29.04.2026).

4. Снейдер, Й. Эффективное программирование TCP/IP : практическое пособие : [16+] / Й. Снейдер ; пер. с англ. А. А. Слинкина. – 2-е изд. – Москва : ДМК Пресс, 2022. – 321 с. : ил., табл., схем. – (Для программистов). [Электронный ресурс] / Режим доступа: https://biblioclub.ru/index.php?page=book&id=705673, (дата обращения 02.05.2026).

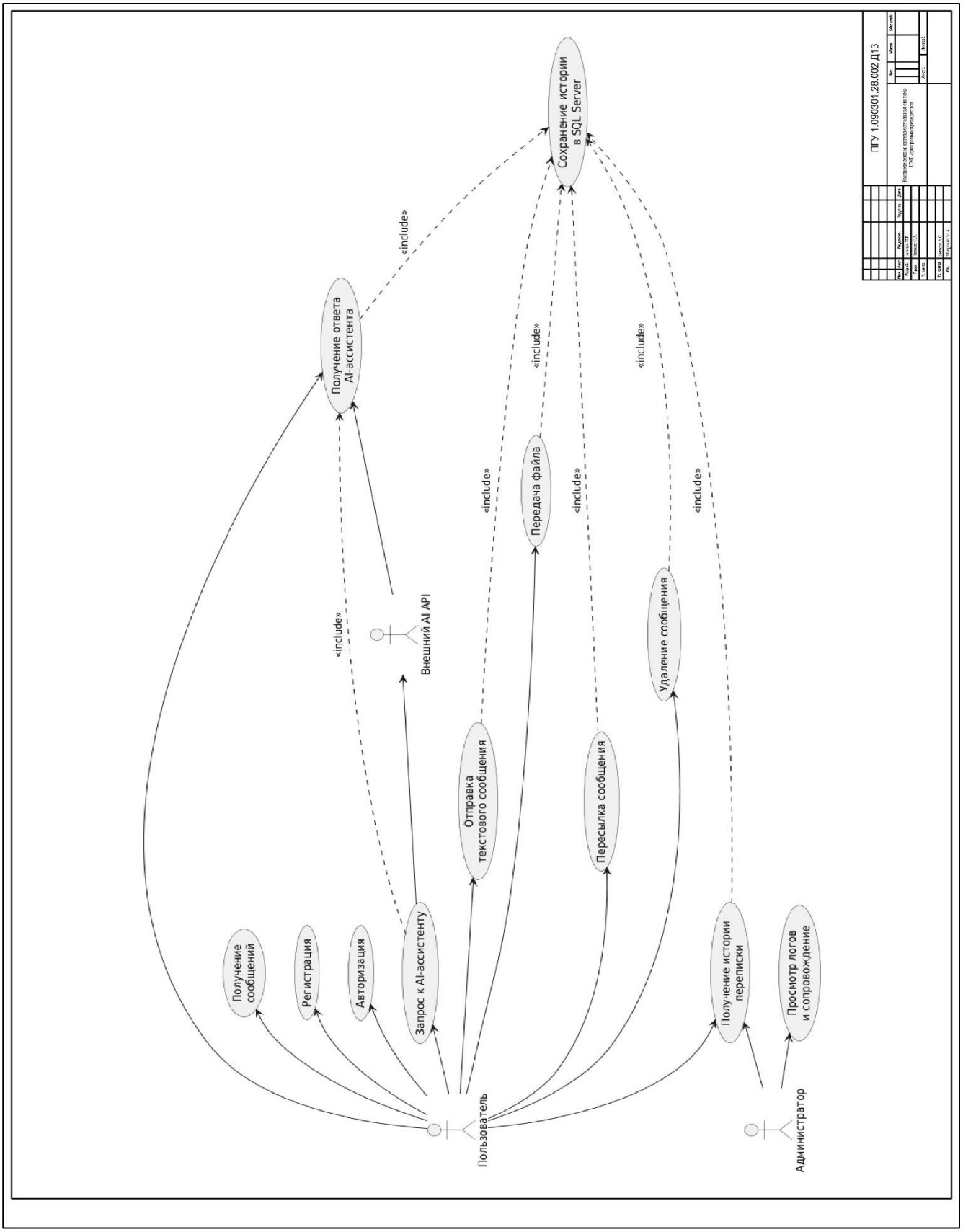
Приложение А

(обязательное)

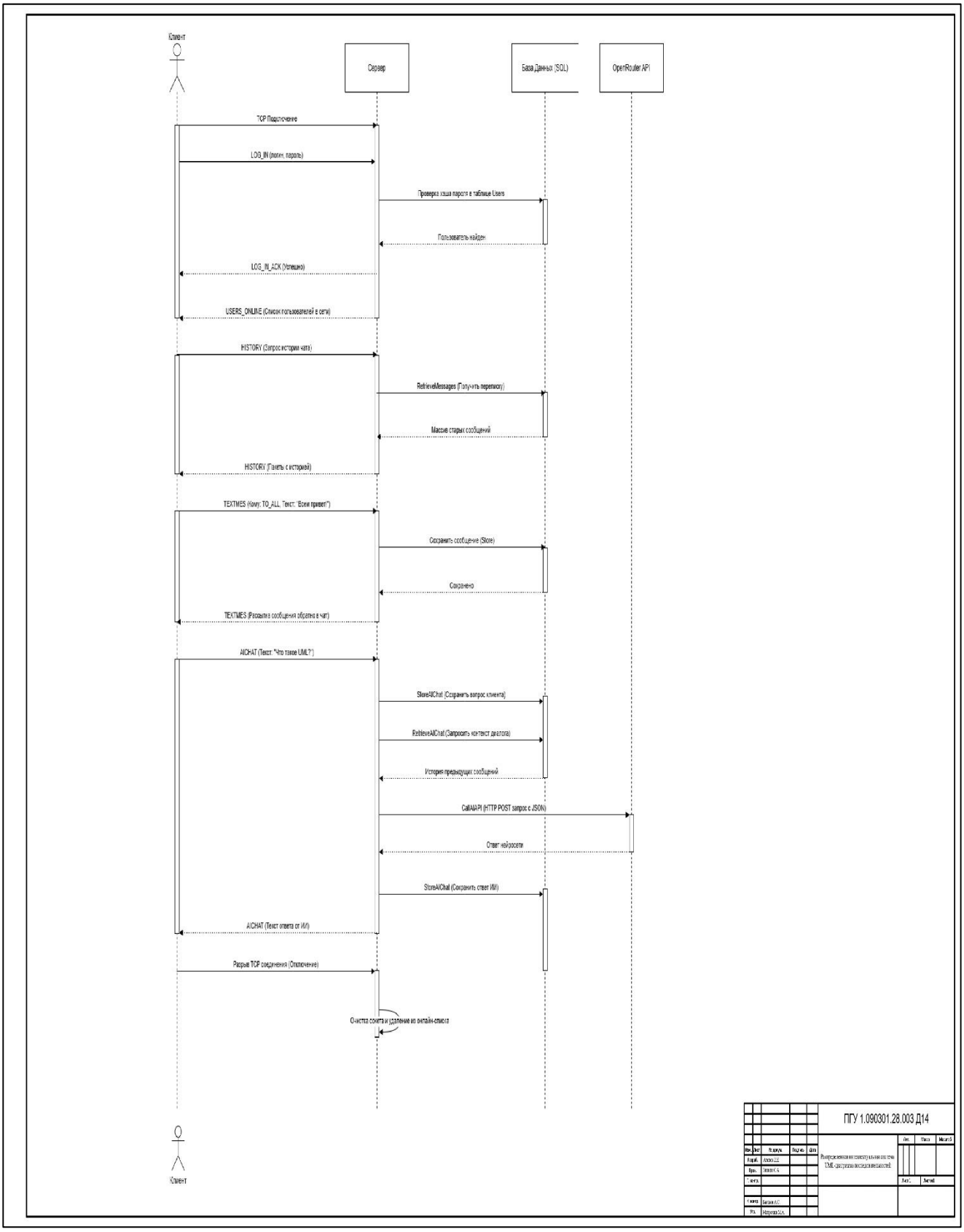
Распределенная интеллектуальная система

UML-диаграммы

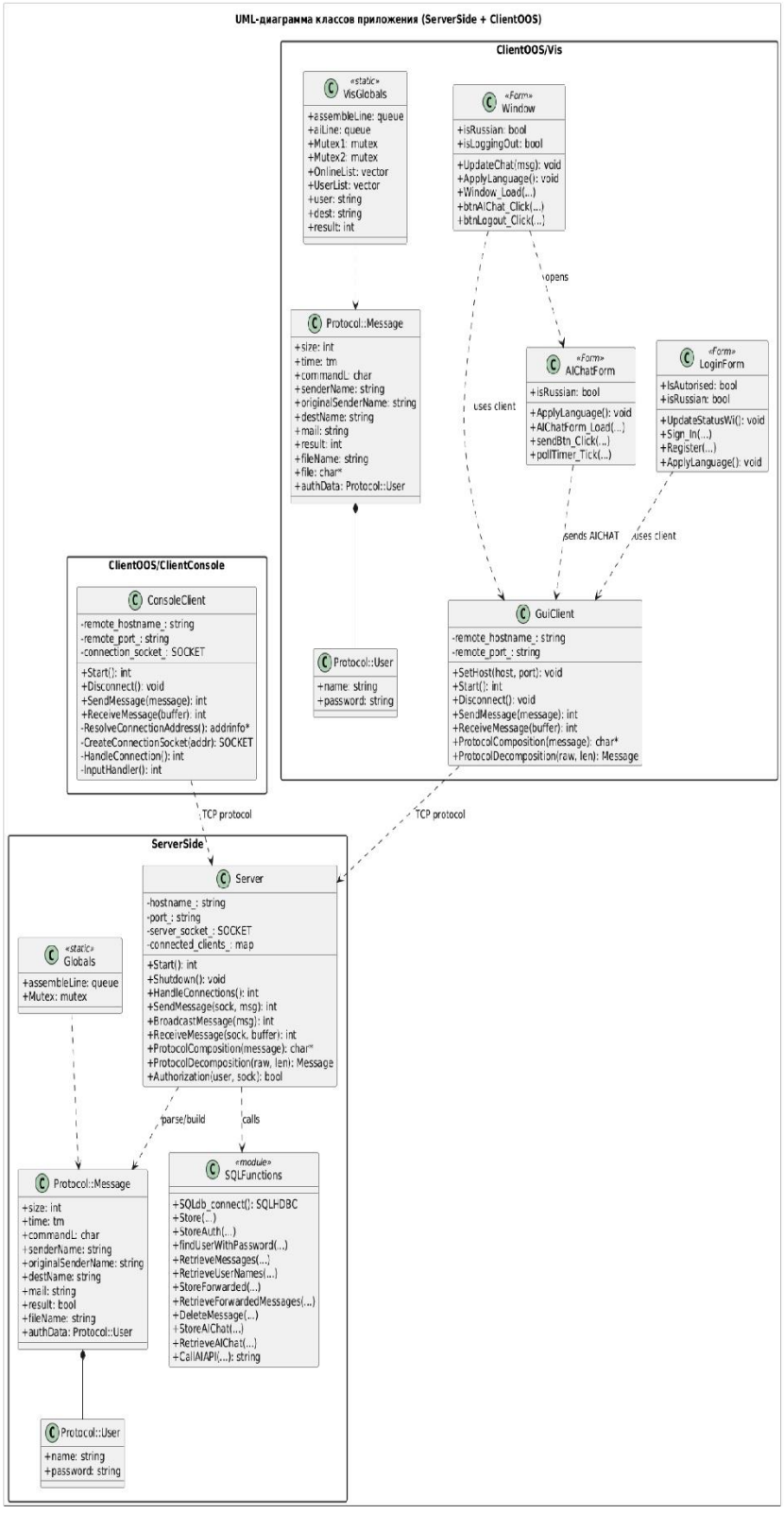
Приложение А.1 – UML-диаграмма прецедентов



Приложение А.2 – UML-диаграмма последовательностей

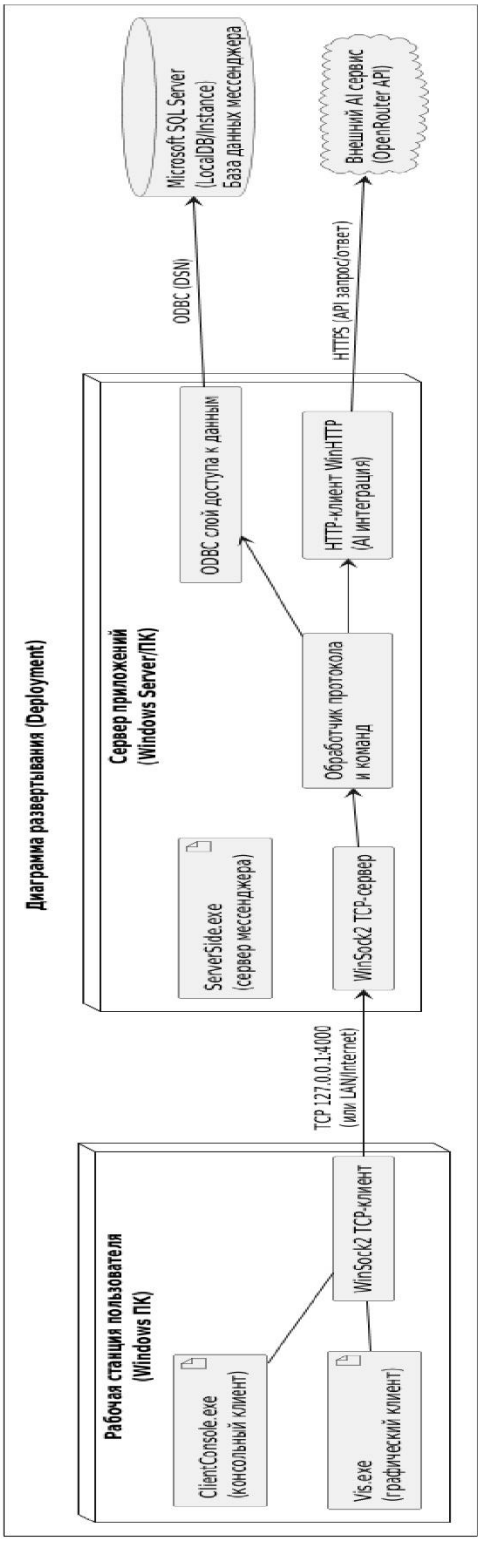


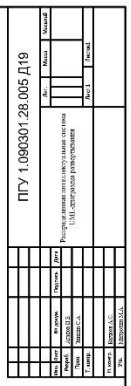
Приложение А.3 – UML-диаграмма классов



ПГУ 1.090301.28.004 Д11

Приложение А.4 – UML-диаграмма развертывания





Приложение Б

(обязательное)

Распределенная интеллектуальная система

Листинг программы

Листинг сервера

Файл Server.cpp

```
#include "Server.h"
#ifndef SERV
#define SERV
addrinfo* Server::GetServerLocalAddress() {
    addrinfo hints, * bind_address;
    memset(&hints, 0x00, sizeof(hints));
    hints.ai_family = AF_INET;
    hints.ai_socktype = SOCK_STREAM;

    std::cerr << "[Debug] Resolving server hostname.\n"s;
    if (getaddrinfo(hostname_.data(), port_.data(), &hints,
&bind_address) != 0) {
        std::cerr << "[Error] Failed to resolve server's local
address: getaddrinfo(): "s <<
std::system_category().message(GET_SOCKET_ERRNO()) << std::endl;
        return nullptr;
    }
    return bind_address;
}
SOCKET Server::CreateServerSocket(addrinfo* bind_address) {
    if (!bind_address) {
        std::cerr << "[Error] CreateServerSocket(): bind_address is
NULL."s << std::endl;
        return -1;
    }

    std::cerr << "[Debug] Creating server socket object.\n"s;
    SOCKET server_socket = socket(bind_address->ai_family,
bind_address->ai_socktype, bind_address->ai_protocol);
    if (!IS_VALID_SOCKET(server_socket)) {
```

```
        std::cerr << "[Error] Failed to create server socket:
socket(): "s << std::system_category().message(GET_SOCKET_ERRNO())
<< std::endl;
        return -1;
    }

    std::cerr << "[Debug] Binding socket to the resolved address."s
<< std::endl;
    if (bind(server_socket, bind_address->ai_addr, bind_address-
>ai_addrlen) == -1) {
        std::cerr << "[Error] Failed to bind server socket to
address "s << hostname_ << ":"s << port_ << " : bind(): "s <<
GET_SOCKET_ERRNO() << std::endl;
        return -1;
    }
    if (ConfigureServerSocket(server_socket) == -1) {
        return -1;
    }
    return server_socket;

}
int Server::ConfigureServerSocket(SOCKET serv_socket) {

#ifdef _WIN32
    char yes = 1;
#else
    int yes = 1;
#endif
    std::cerr << "[Debug] Setting SO_REUSEADDR socket option to the
server socket.\n"s;
    if (setsockopt(serv_socket, SOL_SOCKET, SO_REUSEADDR, &yes,
sizeof(yes)) == -1) {
```

```
        std::cerr << "[Error] Failed to set socket options:
setsockopt(): "s <<
std::system_category().message(GET_SOCKET_ERRNO()) << std::endl;
        return -1;
    }

    std::cerr << "[Debug] Activating server listenning mode.\n"s;
    if (listen(serv_socket, BACKLOG) == -1) {
        std::cerr << "[Error] Failed to activate socket listenner:
listen(): "s << std::system_category().message(GET_SOCKET_ERRNO())
<< std::endl;
        return -1;
    }
    std::cout << "[Info] Server is listenning for incoming
connections at "s << hostname_ << ":"s << port_ << '\n';
    return 0;
}
int Server::AcceptConnection()  {
    sockaddr_storage conn_addr;
    socklen_t conn_len = sizeof(conn_addr);

    SOCKET new_conn = accept(server_socket_,
reinterpret_cast<sockaddr*>(&conn_addr), &conn_len);
    ConnectionInfo new_conn_info = GetConnectionInfo(&conn_addr);
    if (!IS_VALID_SOCKET(new_conn)) {
        std::cerr << "[Error] Failed to accept new connection from
"s << new_conn_info.ToString() << " : accept(): "s <<
std::system_category().message(GET_SOCKET_ERRNO()) << std::endl;
        return -1;
    }

    std::cout << "[Info] New connection from "s <<
new_conn_info.ToString() << '\n';
```

```
    if (new_conn > max_socket_) {
        max_socket_ = new_conn;
    }
    FD_SET(new_conn, &sock_polling_set_);
    connected_clients_[new_conn] = new_conn_info;
    return 0;
}

void Server::DisconnectClient(SOCKET sockfd) {
    const ConnectionInfo& conn_info = connected_clients_.at(sockfd);
    std::cerr << "[Info] Client "s << conn_info.ToString() << " has
been disconnected.\n"s;

    CLOSE_SOCKET(sockfd);
    connected_clients_.erase(sockfd);
    FD_CLR(sockfd, &sock_polling_set_);
    Protocol::Message msg;
    msg.commandL = USERS_ONLINE;
    BroadcastMessage(std::string(ProtocolComposition(msg)));
}
int Server::HandleConnections()  {
    char* temp = new char[1024];
    int k;
    int ll = 0;
    while (true) {
        fd_set polling_set_copy = sock_polling_set_;

        if (select(max_socket_ + 1, &polling_set_copy, NULL, NULL,
NULL) < 0) {
            std::cerr << "[Error] Failed to fetch data on the server
socket: select(): "s <<
std::system_category().message(GET_SOCKET_ERRNO()) << std::endl;
            return -1;
        }
```

```
        for (SOCKET sock = 1; sock <= max_socket_ + 1; ++sock) {
            if (FD_ISSET(sock, &polling_set_copy)) {
                if (sock == server_socket_) {
                    if (AcceptConnection() == -1) {
                        return -1;
                    }

                }
                else {
                    char msg_buffer[MAX_DATA_BUFFER_SIZE];
                    memset(msg_buffer, 0x00, MAX_DATA_BUFFER_SIZE);

                    int recv_bytes = ReceiveMessage(sock,
msg_buffer);

                    if (recv_bytes <= 0) {
                        DisconnectClient(sock);
                        continue;
                    }

                   Protocol::Message mes =
ProtocolDecomposition(msg_buffer, recv_bytes);
                   switch (mes.commandL) {
                   case LOG_IN:
                       std::cout << "[Server] LOG_IN request from "
<< mes.authData.name << std::endl;
                       if (Authorization(mes.authData, sock)) {
                           std::cout << "[Server] Authorization
SUCCESS for " << mes.authData.name << std::endl;
                           Protocol::Message msg;
                           msg.commandL = USERS_ONLINE;
                           BroadcastMessage(std::string(ProtocolComp
osition(msg)));
```

```
                        Protocol::Message msg1;
                        msg1.commandL = USERS_OFLINE;
                        Server::SendMessage(sock,
ProtocolComposition(msg1));
                    }
                    break;

                case TEXTMES:
                    if (mes.destName == "TO_ALL") {
                        BroadcastMessage(std::string(ProtocolComp
osition(mes)));

                    }
                    for (const auto& i : connected_clients_) {
                        if (mes.destName == i.second.user.name) {

                            Server::SendMessage(i.first,
msg_buffer);
                        }

                    }
                    Store(dbc, mes);
                    break;
                case HISTORY:
                    std::cerr << "[Server] HISTORY request from "
<< mes.senderName << " to " << mes.destName << std::endl;
                    History = RetrieveMessages(dbc,
mes.senderName, mes.destName);
                    std::cerr << "[Server] HISTORY found " <<
History.size() << " messages" << std::endl;
                    for (auto& i : History) {
                        if (i.commandL != FORWARD) {
                            i.commandL = HISTORY;
                        }
```

```
                            Server::SendMessage(sock,
ProtocolComposition(i));
                        }
                        History.clear();
                        break;
                    case REGISTER:
                        StoreAuth(dbc, mes.authData);
                        break;
                    case FILE:
                    {
                        int fileSize = stoi(mes.mail);
                        char* fileBuf = new char[fileSize + 1];
                        memset(fileBuf, 0, fileSize + 1);
                        int received = 0;
                        while (received < fileSize) {
                            int r = recv(sock, fileBuf + received,
fileSize - received, 0);
                            if (r <= 0) break;
                            received += r;
                        }
                        std::cout << "[Server FILE] Received " <<
received << " bytes, filename=" << mes.fileName << std::endl;
                        for (const auto& i : connected_clients_) {
                            if (mes.destName == i.second.user.name) {
                                Server::SendMessage(i.first,
std::string(msg_buffer));
                                send(i.first, fileBuf, received,
NULL);
                                std::cout << "[Server FILE] Forwarded
to " << mes.destName << std::endl;
                                break;
                            }
                        }
                        delete[] fileBuf;
```

```
                    break;
                }
                case FORWARD:
                    if
(connected_clients_[sock].user.name.empty()) {
                        break;
                    }
                    if (mes.destName == "TO_ALL") {
                        BroadcastMessage(std::string(ProtocolComp
osition(mes)));
                        StoreForwarded(dbc, mes);
                    } else {
                        bool found = false;
                        for (const auto& i : connected_clients_)
{
                            if (mes.destName ==
i.second.user.name) {
                                Server::SendMessage(i.first,
std::string(ProtocolComposition(mes)));
                                found = true;
                                break;
                            }
                        }
                        Server::SendMessage(sock,
std::string(ProtocolComposition(mes)));
                        StoreForwarded(dbc, mes);
                    }
                    break;
                case DELETE_MSG:
                    std::cerr << "[Server] DELETE_MSG request
from " << mes.senderName << std::endl;
                    DeleteMessage(dbc, mes);
                    break;
                case AICHAT:
```

```
                    {
                        std::cerr << "[Server] AICHAT from " <<
mes.senderName << ": " << mes.mail << std::endl;
                        if (mes.mail == "__HISTORY__") {
                            auto history = RetrieveAIChat(dbc,
mes.senderName);
                            for (auto& entry : history) {
                                Protocol::Message resp;
                                resp.commandL = AICHAT;
                                resp.senderName = entry.first;
                                resp.mail = entry.second;
                                time_t now2 = time(NULL);
                                localtime_s(&resp.time, &now2);
                                Server::SendMessage(sock,
std::string(ProtocolComposition(resp)));
                            }
                        } else {
                            StoreAIChat(dbc, mes.senderName, "user",
mes.mail);
                            std::string aiReply =
CallAIAPI(mes.mail);
                            if (aiReply.size() > 800) aiReply =
aiReply.substr(0, 800) + "...";
                            StoreAIChat(dbc, mes.senderName, "ai",
aiReply);
                            Protocol::Message resp;
                            resp.commandL = AICHAT;
                            resp.senderName = "ai";
                            resp.mail = aiReply;
                            time_t now2 = time(NULL);
                            localtime_s(&resp.time, &now2);
                            Server::SendMessage(sock,
std::string(ProtocolComposition(resp)));
                        }
```

```
                    break;
                }
                default:

                    break;
                }


                 std::string msg_str(msg_buffer);

            }
        }
    }
    return 0;
}
bool Server::Authorization(Protocol::User auth,SOCKET sock) {
    if (findUserWithPassword(dbc, auth.name, auth.password)) {
        connected_clients_.at(sock).user = auth;
        Protocol::Message mes;
        mes.commandL = LOG_IN_ACK;
        mes.result = true;
        Server::SendMessage(sock,
std::string(ProtocolComposition(mes)));
        return true;
    }
    else {
        Protocol::Message mes;
        mes.commandL = LOG_IN_ACK;
        mes.result = false;
        Server::SendMessage(sock,
std::string(ProtocolComposition(mes)));
        return false;
    }
```

```
}
int Server::SendMessage(SOCKET receipient_socketfd, const
std::string& message) {
    std::string assembled_msg = message;
    PrependMessageLength(assembled_msg);

    int total_bytes = assembled_msg.size();
    int sent_bytes = 0;
    int sent_n;
    const ConnectionInfo& conn_info =
connected_clients_.at(receipient_socketfd);
    while (sent_bytes < total_bytes) {
        sent_n = send(receipient_socketfd, assembled_msg.data() +
sent_bytes, total_bytes - sent_bytes, 0);
        if (sent_n == -1) {
            std::cerr << "[Error] Failed to send data to "s <<
conn_info.ToString() << " : send(): "s <<
std::system_category().message(GET_SOCKET_ERRNO()) << std::endl;
            return sent_n;
        }
        sent_bytes += sent_n;
    }
    return sent_bytes;
}
int Server::BroadcastMessage(const std::string& message) {
    std::string packet = message;
    for (const auto i : connected_clients_) {
        if (SendMessage(i.first, packet) == -1) {
            DisconnectClient(i.first);
            return -1;
        }
    }
    return 0;
```

```
}

int Server::ReceiveMessage(SOCKET sender_socketfd, char*
writable_buffer) {
    if (!writable_buffer) {
        std::cerr << "[Error] ReceiveMessage(): writable_buffer is
NULL."s << std::endl;
        return -1;
    }
    const ConnectionInfo& conn_info =
connected_clients_.at(sender_socketfd);

    char message_size_str[5];

    memset(message_size_str, 0x00, sizeof(message_size_str));
    message_size_str[4] = '\0';

    int recv_bytes = recv(sender_socketfd, message_size_str,
sizeof(message_size_str) - 1, 0);
    if (recv_bytes <= 0) {
        if (recv_bytes < 0) {
            std::cerr << "[Error] Failed to receive message length
from "s << conn_info.ToString() << " : recv(): "s <<
std::system_category().message(GET_SOCKET_ERRNO()) << std::endl;
        }
        return recv_bytes;
    }
    for (const char c : std::string(message_size_str)) {
        if (!std::isdigit(c)) {
            std::cerr << "[Warning] Client "s <<
conn_info.ToString() << " : Communication protocol violaition. (char
'"s << c << "')\n"s;
            return -1;
        }
```

```
    }
    int packet_length = std::atoi(message_size_str);
    recv_bytes = recv(sender_socketfd, writable_buffer,
packet_length, 0);
    if (recv_bytes <= 0) {
        if (recv_bytes < 0) {
            std::cerr << "[Error] Failed to receive message from "s
<< conn_info.ToString() << " : recv(): "s <<
std::system_category().message(GET_SOCKET_ERRNO()) << std::endl;
        }
    }

    Protocol::Message regular =
ProtocolDecomposition(writable_buffer, packet_length);
    return recv_bytes;
}
int Server::Start() {

    addrinfo* server_addr_struct = GetServerLocalAddress();
    if (!server_addr_struct) {
        return -1;
    }

    server_socket_ = CreateServerSocket(server_addr_struct);
    if (!IS_VALID_SOCKET(server_socket_)) {
        return -1;
    }
    freeaddrinfo(server_addr_struct);

    FD_ZERO(&sock_polling_set_);
    FD_SET(server_socket_, &sock_polling_set_);
    max_socket_ = server_socket_;
    mytime = time(NULL);
    dbc = SQLdb_connect();
```

```
    return HandleConnections();
}

void Server::Shutdown() {
    std::cerr << "[Info] Shutting down the server...\n"s;
    CLOSE_SOCKET(server_socket_);
    std::cerr << "[Info] Server is shut down." << std::endl;
}
char* Server::ProtocolComposition(Protocol::Message message)
{
    if (message.commandL == TEXTMES || message.commandL ==
DELETE_MSG ||
        message.commandL == LOG_IN_ACK || message.commandL ==
USERS_ONLINE ||
        message.commandL == USERS_OFLINE || message.commandL ==
REGISTER ||
        message.commandL == AICHAT) {
        mytime = time(NULL);
        localtime_s(&message.time, &mytime);
    }
    int size = 0;
    std::string buf;
    char* mes = {0};
    int msg_len;
    int users = 0;
    switch (message.commandL)
    {
    case TEXTMES:
        msg_len = 100 + message.senderName.size() +
message.destName.size() + message.mail.size();
        mes = new char[msg_len];

        sprintf_s(mes, msg_len,"%2d%2d%4d%2d%2d%2d%2d%s/%s/%s/",
message.time.tm_mday, message.time.tm_mon , message.time.tm_year ,
```

```
message.time.tm_hour, message.time.tm_min, message.time.tm_sec,
message.commandL, message.senderName.c_str(),
message.destName.c_str(), message.mail.c_str());
        break;
    case USERS_ONLINE:
        for (auto& item : connected_clients_) {
            if (item.second.success) {
                size += item.second.user.name.size();
                buf.append(item.second.user.name);
                buf.append("/");
                size++;
                users++;
            }
        }
        msg_len = 100 + size;
        mes = new char[msg_len];
        sprintf_s(mes, msg_len, "%2d%2d%4d%2d%2d%2d%2d%2d%s",
message.time.tm_mday, message.time.tm_mon , message.time.tm_year ,
message.time.tm_hour, message.time.tm_min, message.time.tm_sec,
message.commandL, users, buf.c_str());
        break;
    case USERS_OFLINE:
        UsersAll = RetrieveUserNames(dbc);
        for (auto& item : UsersAll) {
                size += item.size();
                buf.append(item);
                buf.append("/");
                size++;
                users++;

        }
        msg_len = 100 + size;
        mes = new char[msg_len];
```

```
        sprintf_s(mes, msg_len, "%2d%2d%4d%2d%2d%2d%2d%2d%s",
message.time.tm_mday, message.time.tm_mon , message.time.tm_year ,
message.time.tm_hour, message.time.tm_min, message.time.tm_sec,
message.commandL, users, buf.c_str());
        break;

    case LOG_IN_ACK:

        msg_len = 100;
        mes = new char[msg_len];
        if (message.result) { users = 1; }
        else { users = 0; }
        sprintf_s(mes, msg_len, "%2d%2d%4d%2d%2d%2d%2d%1d",
message.time.tm_mday, message.time.tm_mon , message.time.tm_year ,
message.time.tm_hour, message.time.tm_min, message.time.tm_sec,
message.commandL, users);
        break;
    case HISTORY:
        message.senderName.erase(std::remove(message.senderName.begi
n(), message.senderName.end(), ' '), message.senderName.end());
        message.destName.erase(std::remove(message.destName.begin(),
message.destName.end(), ' '), message.destName.end());
        msg_len = 100 + message.senderName.size() +
message.destName.size() + message.mail.size();
        mes = new char[msg_len];

        sprintf_s(mes, msg_len, "%2d%2d%4d%2d%2d%2d%2d%s/%s/%s/",
message.time.tm_mday, message.time.tm_mon, message.time.tm_year ,
message.time.tm_hour, message.time.tm_min, message.time.tm_sec,
message.commandL, message.senderName.c_str(),
message.destName.c_str(), message.mail.c_str());

        break;
    case FORWARD:
```

```
        if (message.senderName.empty()) message.senderName =
"unknown";
        if (message.originalSenderName.empty())
message.originalSenderName = "unknown";
        if (message.destName.empty()) message.destName = "unknown";
        if (message.mail.empty()) message.mail = "";
        msg_len = 100 + message.senderName.size()
                    + message.originalSenderName.size()
                    + message.destName.size()
                    + message.mail.size();
        mes = new char[msg_len];
        memset(mes, 0, msg_len);
        std::cerr << "[Server FORWARD] msg_len=" << msg_len
                  << " time: " << message.time.tm_mday << "/" <<
message.time.tm_mon << "/" << message.time.tm_year
                  << " " << message.time.tm_hour << ":" <<
message.time.tm_min << ":" << message.time.tm_sec
                  << " cmd=" << (int)message.commandL
                  << " sender=" << message.senderName << " dest=" <<
message.destName << std::endl;

        sprintf_s(mes, msg_len,
            "%2d%2d%4d%2d%2d%2d%2d%s/%s/%s/%s/",
            message.time.tm_mday, message.time.tm_mon,
message.time.tm_year,
            message.time.tm_hour, message.time.tm_min,
message.time.tm_sec,
            message.commandL,
            message.senderName.c_str(),
            message.originalSenderName.c_str(),
            message.destName.c_str(),
            message.mail.c_str());
        std::cerr << "[Server FORWARD] sprintf_s completed
successfully" << std::endl;
```

```
        break;
    case AICHAT:
        msg_len = 50 + message.senderName.size() +
message.mail.size();
        mes = new char[msg_len];
        memset(mes, 0, msg_len);
        sprintf_s(mes, msg_len, "%2d%2d%4d%2d%2d%2d%2d%s/%s/",
            message.time.tm_mday, message.time.tm_mon,
message.time.tm_year,
            message.time.tm_hour, message.time.tm_min,
message.time.tm_sec,
            message.commandL,
            message.senderName.c_str(),
            message.mail.c_str());
        break;
    default:
        break;
    }
    return mes;
}
Protocol::Message Server::ProtocolDecomposition(const char* message,
int msg_len)
{
    Protocol::Message assembled;
    assembled.size = msg_len;
    char buf1[255] = { 0 };
    char buf2[255] = { 0 };
    char* buf3;
    int sizeSend;
    int sizeDest;
    int j = 0;
   std::string strbuf;
    int hosts = 0;
    assembled.time.tm_mday = NumberDec(message, 0, 1);
```

```
    assembled.time.tm_mon = NumberDec(message, 2, 3);
    assembled.time.tm_year = NumberDec4(message, 4, 7);
    assembled.time.tm_hour = NumberDec(message, 8, 9);
    assembled.time.tm_min = NumberDec(message, 10, 11);
    assembled.time.tm_sec = NumberDec(message, 12, 13);
    assembled.commandL = NumberDec(message, 14, 15);
    switch (assembled.commandL)
    {
    case HISTORY:
    case TEXTMES:
        for (int i = 0; i < 255; i++) {
            sizeSend = i;
            if (message[16 + i] == '/') {
                break;
            }
            buf1[i] = message[16 + i];

        }
        sizeSend++;
        assembled.senderName = std::string(buf1);
        for (int i = 0; i < 255; i++) {
            sizeDest = i;
            if (message[16 + sizeSend + i] == '/') {
                break;
            }
            buf2[i] = message[16 + sizeSend + i];
        }
        assembled.destName = std::string(buf2);
        sizeDest++;

        buf3 = new char[msg_len - (16 + sizeSend + sizeDest)];
        for (int i = 0; i < msg_len - (16 + sizeSend + sizeDest);
i++) {
            buf3[i] = 0;
```

```
        }
        for (j = 0;16 + sizeSend + sizeDest + j < msg_len;j++) {
            if (message[16 + sizeSend + sizeDest + j] == '/') {
                break;
            }
            buf3[j] = message[16 + sizeSend + sizeDest + j];

        }

        assembled.mail = std::string(buf3);
        break;
    case USERS_ONLINE:
        hosts = NumberDec(message, 16, 17);
        for (int i = 0; i < hosts; i++) {

            for (j = 0;message[j] != '/';j++) {
                buf1[j] = message[18 + j];
            }
            j++;
            strbuf.assign(buf1);
        }
        break;
    case LOG_IN:
        for (int i = 0; i < 255; i++) {
            sizeSend = i;
            if (message[16 + i] == '/') {
                break;
            }
            buf1[i] = message[16 + i];
        }
        sizeSend++;
        assembled.authData.name = std::string(buf1);
        for (int i = 0; i <255; i++) {
            if (message[16 + sizeSend+ i] == '/') {
```

```
                break;
            }
            buf2[i] = message[16 + sizeSend+ i];
        }
        assembled.authData.password = std::string(buf2);
        break;
    case REGISTER:
        for (int i = 0; i < 255; i++) {
            sizeSend = i;
            if (message[16 + i] == '/') {
                break;
            }
            buf1[i] = message[16 + i];
        }
        sizeSend++;
        assembled.authData.name = std::string(buf1);
        for (int i = 0; i < 255; i++) {
            if (message[16 + sizeSend + i] == '/') {
                break;
            }
            buf2[i] = message[16 + sizeSend + i];
        }
        assembled.authData.password = std::string(buf2);
        break;
    case FILE:
        buf3 = new char[1025];
        for (int i = 0; i < 255; i++) {
            sizeSend = i;
            if (message[16 + i] == '/') {
                break;
            }
            buf1[i] = message[16 + i];

        }
```

```
        sizeSend++;
        assembled.senderName = std::string(buf1);
        for (int i = 0; i < 255; i++) {
            sizeDest = i;
            if (message[16 + sizeSend + i] == '/') {
                break;
            }
            buf2[i] = message[16 + sizeSend + i];
        }
        assembled.destName = std::string(buf2);
        sizeDest++;
        j = sizeDest + sizeSend;
        memset(&buf1[0], 0, sizeof(buf1));
        for (int i = 0; i < 60; i++) {
            sizeSend = i;
            if (message[16 + i + j] == '/') {
                break;
            }
            buf1[i] = message[16 + j + i];

        }
        assembled.fileName = std::string(buf1);
        sizeSend++;
        j += sizeSend;
        j += 16;
        memset(&buf1[0], 0, sizeof(buf1));
        for (int i = 0; i < assembled.size - j; i++) {
            buf1[i] = message[i + j];
        }
        assembled.mail = std::string(buf1);
        break;
    case FORWARD:
    {
        char bufOrig[255] = { 0 };
```

```
        for (int i = 0; i < 255; i++) {
            sizeSend = i;
            if (message[16 + i] == '/') break;
            buf1[i] = message[16 + i];
        }
        sizeSend++;
        assembled.senderName = std::string(buf1);
        int sizeOrig = 0;
        for (int i = 0; i < 255; i++) {
            sizeOrig = i;
            if (message[16 + sizeSend + i] == '/') break;
            bufOrig[i] = message[16 + sizeSend + i];
        }
        sizeOrig++;
        assembled.originalSenderName = std::string(bufOrig);
        for (int i = 0; i < 255; i++) {
            sizeDest = i;
            if (message[16 + sizeSend + sizeOrig + i] == '/') break;
            buf2[i] = message[16 + sizeSend + sizeOrig + i];
        }
        assembled.destName = std::string(buf2);
        sizeDest++;
        int offset = 16 + sizeSend + sizeOrig + sizeDest;
        buf3 = new char[msg_len - offset + 1];
        memset(buf3, 0, msg_len - offset + 1);
        for (j = 0; offset + j < msg_len; j++) {
            if (message[offset + j] == '/') break;
            buf3[j] = message[offset + j];
        }
        assembled.mail = std::string(buf3);
        delete[] buf3;
        break;
    }
    case DELETE_MSG:
```

```
        for (int i = 0; i < 255; i++) {
            sizeSend = i;
            if (message[16 + i] == '/') break;
            buf1[i] = message[16 + i];
        }
        sizeSend++;
        assembled.senderName = std::string(buf1);

        for (int i = 0; i < 255; i++) {
            sizeDest = i;
            if (message[16 + sizeSend + i] == '/') break;
            buf2[i] = message[16 + sizeSend + i];
        }
        assembled.destName = std::string(buf2);
        sizeDest++;

        buf3 = new char[msg_len - (16 + sizeSend + sizeDest) + 1];
        memset(buf3, 0, msg_len - (16 + sizeSend + sizeDest) + 1);
        for (j = 0; 16 + sizeSend + sizeDest + j < msg_len; j++) {
            if (message[16 + sizeSend + sizeDest + j] == '/') break;
            buf3[j] = message[16 + sizeSend + sizeDest + j];
        }
        assembled.mail = std::string(buf3);
        delete[] buf3;
        break;
    case AICHAT:
        for (int i = 0; i < 255; i++) {
            sizeSend = i;
            if (message[16 + i] == '/') break;
            buf1[i] = message[16 + i];
        }
        sizeSend++;
        assembled.senderName = std::string(buf1);
        buf3 = new char[msg_len - (16 + sizeSend) + 1];
```

```
            memset(buf3, 0, msg_len - (16 + sizeSend) + 1);
            for (j = 0; 16 + sizeSend + j < msg_len; j++) {
                if (message[16 + sizeSend + j] == '/') break;
                buf3[j] = message[16 + sizeSend + j];
            }
            assembled.mail = std::string(buf3);
            delete[] buf3;
            break;
        default:
            std::cout << "undefined protocol" << std::endl;
            break;
        }
        return assembled;
}
int Server::NumberDec(const char* mes, int start, int end) {
        char buf[3] = {0};
        int j = 0;
        for (int i = start; i < end + 1; i++) {
            buf[j] = mes[i];
            j++;
        }
        buf[j] = '\0';
        return std::stoi(buf);
}
int Server::NumberDec4(const char* mes, int start, int end) {
        char buf[5] = {0};
        int j = 0;
        for (int i = start; i < end + 1; i++) {
            buf[j] = mes[i];
            j++;
        }
        buf[j] = '\0';
        return std::stoi(buf);
}
```

```
#endif
```

Файл ServerConfig.cpp

```
#include "Server.h"

int main() {

#ifdef _WIN32
    WSADATA d;
    if (WSAStartup(MAKEWORD(2, 2), &d)) {
        std::cerr << "Failed to initialize WinSockAPI: " <<
std::system_category().message(GET_SOCKET_ERRNO()) << std::endl;
        return 1;
    }
#endif
    SetConsoleCP(1251);
    SetConsoleOutputCP(1251);
    Server server("127.0.0.1", "4000");
    if (server.Start() == -1) {
        return 1;
    }
}
```

Файл SQL.cpp

```
#include "Common.h"
#include <winhttp.h>
#pragma comment(lib, "winhttp.lib")
SQLHDBC SQLdb_connect() {
    SQLHENV env;
    SQLHDBC dbc;
    SQLRETURN ret;
    ret = SQLAllocHandle(SQL_HANDLE_ENV, SQL_NULL_HANDLE, &env);
    if (ret != SQL_SUCCESS && ret != SQL_SUCCESS_WITH_INFO) {
        std::cerr << "Failed to allocate environment handle." <<
std::endl;
        return NULL;
    }
    ret = SQLSetEnvAttr(env, SQL_ATTR_ODBC_VERSION,
(void*)SQL_OV_ODBC3, 0);
    if (ret != SQL_SUCCESS && ret != SQL_SUCCESS_WITH_INFO) {
        std::cerr << "Failed to set ODBC version." << std::endl;
        SQLFreeHandle(SQL_HANDLE_ENV, env);
        return NULL;
    }
```

```
    ret = SQLAllocHandle(SQL_HANDLE_DBC, env, &dbc);
    if (ret != SQL_SUCCESS && ret != SQL_SUCCESS_WITH_INFO) {
        std::cerr << "Failed to allocate connection handle." <<
std::endl;
        SQLFreeHandle(SQL_HANDLE_ENV, env);
        return NULL;
    }
    SQLCHAR dsn[] = "Kurs";

    ret = SQLConnectA(dbc, dsn, SQL_NTS, (SQLCHAR*)"", SQL_NTS,
(SQLCHAR*)"", SQL_NTS);
    if (ret == SQL_SUCCESS || ret == SQL_SUCCESS_WITH_INFO) {
        std::cout << "Connected to database successfully!" <<
std::endl;
        return dbc;
    }
    else {
        std::cerr << "Failed to connect to database." << std::endl;
        SQLFreeHandle(SQL_HANDLE_DBC, dbc);
        SQLFreeHandle(SQL_HANDLE_ENV, env);
        return NULL;
    }

}
void Store(SQLHDBC hdbc, const Protocol::Message& message) {
    SQLHSTMT hstmt;
    SQLRETURN retcode;
    std::string sqlQuery = R"(
        INSERT INTO Message (size, time, commandL, senderName,
destName, mail)
        VALUES (?, ?, ?, ?, ?, ?)
    )";
    retcode = SQLAllocHandle(SQL_HANDLE_STMT, hdbc, &hstmt);
    if (retcode != SQL_SUCCESS && retcode != SQL_SUCCESS_WITH_INFO)
{
        std::cerr << "[illegible] SQLHANDLE." <<
std::endl;
        return;
    }
    retcode = SQLPrepareA(hstmt, (SQLCHAR*)sqlQuery.c_str(),
SQL_NTS);
    if (retcode != SQL_SUCCESS && retcode != SQL_SUCCESS_WITH_INFO)
{
        std::cerr << "[illegible] SQL-[illegible]" << std::endl;
        SQLFreeHandle(SQL_HANDLE_STMT, hstmt);
        return;
    }
```

```
    SQLBindParameter(hstmt, 1, SQL_PARAM_INPUT, SQL_C_LONG,
SQL_INTEGER, 0, 0, (SQLPOINTER)&message.size, 0, nullptr);
    char timeStr[20];
    std::strftime(timeStr, sizeof(timeStr), "%Y-%m-%d %H:%M:%S",
&message.time);
    SQLBindParameter(hstmt, 2, SQL_PARAM_INPUT, SQL_C_CHAR,
SQL_VARCHAR, 20, 0, timeStr, 0, nullptr);
    int cmdVal = (unsigned char)message.commandL;
    char cmdStr[4] = {0};
    sprintf_s(cmdStr, 4, "%d", cmdVal);
    SQLLEN cmdLen = SQL_NTS;
    SQLBindParameter(hstmt, 3, SQL_PARAM_INPUT, SQL_C_CHAR,
SQL_CHAR, 3, 0, (SQLPOINTER)cmdStr, 0, &cmdLen);
    SQLBindParameter(hstmt, 4, SQL_PARAM_INPUT, SQL_C_CHAR,
SQL_VARCHAR, message.senderName.size(), 0,
(SQLCHAR*)message.senderName.c_str(), 0, nullptr);
    SQLBindParameter(hstmt, 5, SQL_PARAM_INPUT, SQL_C_CHAR,
SQL_VARCHAR, message.destName.size(), 0,
(SQLCHAR*)message.destName.c_str(), 0, nullptr);
    SQLBindParameter(hstmt, 6, SQL_PARAM_INPUT, SQL_C_CHAR,
SQL_VARCHAR, message.mail.size(), 0, (SQLCHAR*)message.mail.c_str(),
0, nullptr);
    retcode = SQLExecute(hstmt);
    if (retcode != SQL_SUCCESS && retcode != SQL_SUCCESS_WITH_INFO)
{
        std::cerr << "������ ��� ��������SQL-������" << std::endl;
    }
    else {
        std::cout << "������ ������ ������ ? ��� ������" <<
std::endl;
    }
    SQLFreeHandle(SQL_HANDLE_STMT, hstmt);
}
void StoreAuth(SQLHDBC hdbc, Protocol::User user) {
    SQLHSTMT hstmt;
    SQLRETURN retcode;
    std::string sqlQuery =
        "INSERT INTO Users (username,password) "
        "VALUES (?,?);";
    retcode = SQLAllocHandle(SQL_HANDLE_STMT, hdbc, &hstmt);
    if (retcode != SQL_SUCCESS && retcode != SQL_SUCCESS_WITH_INFO)
{
        std::cerr << "������ ��� ������ ������SQLHANDLE." <<
std::endl;
        return;
    }
    SQLLEN nameLen = SQL_NTS;
```

```
    SQLLEN passwordLen = SQL_NTS;
    retcode = SQLPrepareA(hstmt, (SQLCHAR*)sqlQuery.c_str(),
SQL_NTS);
    if (retcode != SQL_SUCCESS && retcode != SQL_SUCCESS_WITH_INFO)
{
        std::cerr << "������ ��� ��������SQL-������" << std::endl;
        SQLFreeHandle(SQL_HANDLE_STMT, hstmt);
        return;
    }
    SQLBindParameter(hstmt, 1, SQL_PARAM_INPUT, SQL_C_CHAR,
SQL_CHAR, 255, 0, (SQLCHAR*)user.name.c_str(), 0,&nameLen);
    SQLBindParameter(hstmt, 2, SQL_PARAM_INPUT, SQL_C_CHAR,
SQL_CHAR, 255, 0, (SQLCHAR*)user.password.c_str(), 0, &passwordLen);
    SQLCHAR sqlState[SQL_SQLSTATE_SIZE + 1];
    SQLCHAR message[SQL_MAX_MESSAGE_LENGTH];
    SQLINTEGER nativeError;
    SQLSMALLINT textLength;
    retcode = SQLExecute(hstmt);
    if (retcode != SQL_SUCCESS && retcode != SQL_SUCCESS_WITH_INFO)
{
        std::cerr << "������ ��� ��������SQL-������" << std::endl;

        retcode = SQLGetDiagRecA(SQL_HANDLE_STMT, hstmt, 1,
sqlState, &nativeError, message, sizeof(message), &textLength);
        std::cerr << "SQL State: " << sqlState << ", Message: " <<
message << "native " << retcode << std::endl;
    }
    else {
        std::cout << "������ ������ ������ ! ���! ������" <<
std::endl;
    }
    SQLFreeHandle(SQL_HANDLE_STMT, hstmt);
};
bool findUserWithPassword(SQLHDBC dbc, const std::string& username,
const std::string& password) {
    SQLHSTMT stmt;
    SQLRETURN ret;
    SQLCHAR query[] = "SELECT * FROM Users WHERE username = ? AND
password = ?";
    SQLCHAR sqlState[SQL_SQLSTATE_SIZE + 1];
    SQLCHAR message[SQL_MAX_MESSAGE_LENGTH];
    SQLINTEGER nativeError;
    SQLSMALLINT textLength;
    std::cout << "[Auth] Checking user: '" << username << "' with
password: '" << password << "'" << std::endl;
    ret = SQLAllocHandle(SQL_HANDLE_STMT, dbc, &stmt);
    if (ret != SQL_SUCCESS && ret != SQL_SUCCESS_WITH_INFO) {
```

```
        std::cerr << "Failed to allocate statement handle." <<
std::endl;
        return false;
    }
    ret = SQLPrepareA(stmt, query, SQL_NTS);
    if (ret != SQL_SUCCESS && ret != SQL_SUCCESS_WITH_INFO) {
        std::cerr << "Failed to prepare SQL statement." <<
std::endl;
        SQLFreeHandle(SQL_HANDLE_STMT, stmt);
        return false;
    }
    ret = SQLBindParameter(stmt, 1, SQL_PARAM_INPUT, SQL_C_CHAR,
SQL_CHAR, username.size(), 0, (SQLCHAR*)username.c_str(), 0, NULL);
    if (ret != SQL_SUCCESS && ret != SQL_SUCCESS_WITH_INFO) {
        std::cerr << "Failed to bind username parameter." <<
std::endl;
        SQLFreeHandle(SQL_HANDLE_STMT, stmt);
        return false;
    }
    ret = SQLBindParameter(stmt, 2, SQL_PARAM_INPUT, SQL_C_CHAR,
SQL_CHAR, password.size(), 0, (SQLCHAR*)password.c_str(), 0, NULL);
    if (ret != SQL_SUCCESS && ret != SQL_SUCCESS_WITH_INFO) {
        std::cerr << "Failed to bind password parameter." <<
std::endl;
        SQLFreeHandle(SQL_HANDLE_STMT, stmt);
        return false;
    }
    ret = SQLExecute(stmt);
    if (ret != SQL_SUCCESS && ret != SQL_SUCCESS_WITH_INFO) {
        std::cerr << "Failed to execute SQL query." << std::endl;
        SQLFreeHandle(SQL_HANDLE_STMT, stmt);
        return false;
    }
    ret = SQLFetch(stmt);
    if (ret == SQL_SUCCESS || ret == SQL_SUCCESS_WITH_INFO) {
        std::cout << "User found!" << std::endl;
        SQLFreeHandle(SQL_HANDLE_STMT, stmt);
        return true;
    }
    else {
        std::cout << "User not found." << std::endl;
        SQLFreeHandle(SQL_HANDLE_STMT, stmt);
        return false;
    }
}
std::vector<Protocol::Message> RetrieveMessages(SQLHDBC hdbc, const
std::string& parameter, const std::string& parameter2) {
    SQLHSTMT hstmt = nullptr;
```

```
    SQLRETURN retcode;
    const char* query;
    std::vector<Protocol::Message> messages;
    if (parameter == parameter2) {
        query = R"(
        SELECT size, time, commandL, senderName, destName, mail
        FROM Message
        WHERE senderName = ? AND destName = ? OR destName = 'TO_ALL'
    )";
    }
    else {
        query = R"(
        SELECT size, time, commandL, senderName, destName, mail
        FROM Message
        WHERE (senderName = ? OR senderName = ?)
        AND (destName = ? OR destName = ? OR destName = 'TO_ALL')
        AND senderName != destName;
    )";
    }
    retcode = SQLAllocHandle(SQL_HANDLE_STMT, hdbc, &hstmt);
    if (retcode != SQL_SUCCESS && retcode != SQL_SUCCESS_WITH_INFO)
{
        std::cerr << "�� ������ ������ ������ ������" <<
std::endl;
        return messages;
    }
    SQLBindParameter(hstmt, 1, SQL_PARAM_INPUT, SQL_C_CHAR,
SQL_VARCHAR, parameter.length(), 0,
        (SQLPOINTER)parameter.c_str(), parameter.length(), nullptr);
    SQLBindParameter(hstmt, 2, SQL_PARAM_INPUT, SQL_C_CHAR,
SQL_VARCHAR, parameter2.length(), 0,
        (SQLPOINTER)parameter2.c_str(), parameter2.length(),
nullptr);
    SQLBindParameter(hstmt, 3, SQL_PARAM_INPUT, SQL_C_CHAR,
SQL_VARCHAR, parameter.length(), 0,
        (SQLPOINTER)parameter.c_str(), parameter.length(), nullptr);
    SQLBindParameter(hstmt, 4, SQL_PARAM_INPUT, SQL_C_CHAR,
SQL_VARCHAR, parameter2.length(), 0,
        (SQLPOINTER)parameter2.c_str(), parameter2.length(),
nullptr);
    retcode = SQLExecDirectA(hstmt, (SQLCHAR*)query, SQL_NTS);
    if (retcode != SQL_SUCCESS && retcode != SQL_SUCCESS_WITH_INFO)
{
        std::cerr << "������ ��� ������SQL ������" << std::endl;
        SQLCHAR sqlState[6], messageText[256];
        SQLINTEGER nativeError;
        SQLSMALLINT textLength;
```

```
        SQLGetDiagRecA(SQL_HANDLE_STMT, hstmt, 1, sqlState,
&nativeError, messageText, sizeof(messageText), &textLength);
        std::cerr << "SQL State: " << sqlState << ", Message: " <<
messageText << std::endl;
        SQLFreeHandle(SQL_HANDLE_STMT, hstmt);
        return messages;
    }
    while (SQLFetch(hstmt) == SQL_SUCCESS) {
        Protocol::Message message;
        char timeBuffer[20] = {0};
        char commandBuf[4]  = {0};
        char senderNameBuffer[256] = {0};
        char destNameBuffer[256]   = {0};
        char mailBuffer[1024]      = {0};
        SQLGetData(hstmt, 1, SQL_C_LONG,
&message.size,    0,                        nullptr);
        SQLGetData(hstmt, 2, SQL_C_CHAR,
timeBuffer,       sizeof(timeBuffer),       nullptr);
        SQLGetData(hstmt, 3, SQL_C_CHAR,
commandBuf,       sizeof(commandBuf),       nullptr);
        SQLGetData(hstmt, 4, SQL_C_CHAR, senderNameBuffer,
sizeof(senderNameBuffer), nullptr);
        SQLGetData(hstmt, 5, SQL_C_CHAR,
destNameBuffer,   sizeof(destNameBuffer),   nullptr);
        SQLGetData(hstmt, 6, SQL_C_CHAR,
mailBuffer,       sizeof(mailBuffer),       nullptr);
        std::istringstream timeStream(timeBuffer);
        timeStream >> std::get_time(&message.time, "%Y-%m-%d
%H:%M:%S");
        if (timeStream.fail()) {
            time_t now = time(NULL);
            localtime_s(&message.time, &now);
        }
        message.commandL = (char)std::atoi(commandBuf);
        message.senderName = std::string(senderNameBuffer);
        message.destName   = std::string(destNameBuffer);
        message.mail       = std::string(mailBuffer);

        messages.push_back(message);
    }
    SQLFreeHandle(SQL_HANDLE_STMT, hstmt);
    std::vector<Protocol::Message> forwardedMessages =
RetrieveForwardedMessages(hdbc, parameter, parameter2);
    messages.insert(messages.end(), forwardedMessages.begin(),
forwardedMessages.end());
    return messages;
}
std::vector<std::string> RetrieveUserNames(SQLHDBC hdbc) {
```

```
    SQLHSTMT hstmt = nullptr;
    SQLRETURN retcode;
    std::vector<std::string> userNames;
    const char* query = R"(
        SELECT DISTINCT username FROM Users
    )";
    retcode = SQLAllocHandle(SQL_HANDLE_STMT, hdbc, &hstmt);
    if (retcode != SQL_SUCCESS && retcode != SQL_SUCCESS_WITH_INFO)
{
        std::cerr << "[illegible]" <<
std::endl;
        return userNames;
    }
    retcode = SQLExecDirectA(hstmt, (SQLCHAR*)query, SQL_NTS);
    if (retcode != SQL_SUCCESS && retcode != SQL_SUCCESS_WITH_INFO)
{
        std::cerr << "[illegible]SQL [illegible]" << std::endl;
        SQLCHAR sqlState[6], messageText[256];
        SQLINTEGER nativeError;
        SQLSMALLINT textLength;
        SQLGetDiagRecA(SQL_HANDLE_STMT, hstmt, 1, sqlState,
&nativeError, messageText, sizeof(messageText), &textLength);
        std::cerr << "SQL State: " << sqlState << ", Message: " <<
messageText << std::endl;

        SQLFreeHandle(SQL_HANDLE_STMT, hstmt);
        return userNames;
    }
    char nameBuffer[256];
    while (SQLFetch(hstmt) == SQL_SUCCESS) {
        SQLGetData(hstmt, 1, SQL_C_CHAR, nameBuffer,
sizeof(nameBuffer), nullptr);
        userNames.emplace_back(nameBuffer);
    }
    SQLFreeHandle(SQL_HANDLE_STMT, hstmt);
    return userNames;
}
void StoreForwarded(SQLHDBC hdbc, const Protocol::Message& message)
{
    SQLHSTMT hstmt;
    SQLRETURN retcode;
    char senderBuf[256] = {0};
    char origSenderBuf[256] = {0};
    char destBuf[256] = {0};
    char mailBuf[4096] = {0};
    strncpy_s(senderBuf, message.senderName.c_str(), 255);
```

```
    strncpy_s(origSenderBuf, message.originalSenderName.c_str(),
255);
    strncpy_s(destBuf, message.destName.c_str(), 255);
    strncpy_s(mailBuf, message.mail.c_str(), 4095);
    SQL_TIMESTAMP_STRUCT ts = {0};
    ts.year     = message.time.tm_year + 1900;
    ts.month    = message.time.tm_mon + 1;
    ts.day      = message.time.tm_mday;
    ts.hour     = message.time.tm_hour;
    ts.minute   = message.time.tm_min;
    ts.second   = message.time.tm_sec;
    ts.fraction = 0;
    std::cerr << "[StoreForwarded] time=" << ts.year << "-" <<
ts.month << "-" << ts.day
              << " " << ts.hour << ":" << ts.minute << std::endl;

    std::string sqlQuery = R"(
        INSERT INTO ForwardedMessage (size, time, commandL,
senderName, originalSenderName, destName, mail)
        VALUES (?, ?, ?, ?, ?, ?, ?)
    )";
    retcode = SQLAllocHandle(SQL_HANDLE_STMT, hdbc, &hstmt);
    if (retcode != SQL_SUCCESS && retcode != SQL_SUCCESS_WITH_INFO)
{
        std::cerr << "StoreForwarded: failed to allocate statement
handle." << std::endl;
        return;
    }
    retcode = SQLPrepareA(hstmt, (SQLCHAR*)sqlQuery.c_str(),
SQL_NTS);
    if (retcode != SQL_SUCCESS && retcode != SQL_SUCCESS_WITH_INFO)
{
        std::cerr << "StoreForwarded: failed to prepare SQL
statement." << std::endl;
        SQLFreeHandle(SQL_HANDLE_STMT, hstmt);
        return;
    }
    SQLLEN sizeLen = 0;
    SQLLEN timeLen = sizeof(SQL_TIMESTAMP_STRUCT);
    SQLLEN commandLen = 1;
    SQLLEN senderLen = SQL_NTS;
    SQLLEN origSenderLen = SQL_NTS;
    SQLLEN destLen = SQL_NTS;
    SQLLEN mailLen = SQL_NTS;
    SQLBindParameter(hstmt, 1, SQL_PARAM_INPUT, SQL_C_LONG,
SQL_INTEGER, 0, 0,
        (SQLPOINTER)&message.size, 0, &sizeLen);
```

```
    SQLBindParameter(hstmt, 2, SQL_PARAM_INPUT,
SQL_C_TYPE_TIMESTAMP, SQL_TYPE_TIMESTAMP,
        19, 0, (SQLPOINTER)&ts, sizeof(ts), &timeLen);
    SQLBindParameter(hstmt, 3, SQL_PARAM_INPUT, SQL_C_CHAR,
SQL_CHAR, 1, 0,
        (SQLPOINTER)&message.commandL, 1, &commandLen);
    SQLBindParameter(hstmt, 4, SQL_PARAM_INPUT, SQL_C_CHAR,
SQL_VARCHAR,
        255, 0, (SQLPOINTER)senderBuf, 0, &senderLen);
    SQLBindParameter(hstmt, 5, SQL_PARAM_INPUT, SQL_C_CHAR,
SQL_VARCHAR,
        255, 0, (SQLPOINTER)origSenderBuf, 0, &origSenderLen);
    SQLBindParameter(hstmt, 6, SQL_PARAM_INPUT, SQL_C_CHAR,
SQL_VARCHAR,
        255, 0, (SQLPOINTER)destBuf, 0, &destLen);
    SQLBindParameter(hstmt, 7, SQL_PARAM_INPUT, SQL_C_CHAR,
SQL_VARCHAR,
        4096, 0, (SQLPOINTER)mailBuf, 0, &mailLen);
    retcode = SQLExecute(hstmt);
    if (retcode != SQL_SUCCESS && retcode != SQL_SUCCESS_WITH_INFO)
{
        std::cerr << "StoreForwarded: failed to execute INSERT." <<
std::endl;
        SQLCHAR sqlState[6], errorMsg[256];
        SQLINTEGER nativeError;
        SQLSMALLINT textLength;
        SQLGetDiagRecA(SQL_HANDLE_STMT, hstmt, 1, sqlState,
&nativeError, errorMsg, sizeof(errorMsg), &textLength);
        std::cerr << "SQL State: " << sqlState << ", Error: " <<
errorMsg << std::endl;
    }
    SQLFreeHandle(SQL_HANDLE_STMT, hstmt);
}
std::vector<Protocol::Message> RetrieveForwardedMessages(SQLHDBC
hdbc,
    const std::string& parameter, const std::string& parameter2)
{
    SQLHSTMT hstmt = nullptr;
    SQLRETURN retcode;
    std::vector<Protocol::Message> messages;
    const char* query;

    std::cerr << "[RetrieveForwardedMessages] parameter='" <<
parameter
              << "' parameter2='" << parameter2 << "'" << std::endl;
    query = R"(
        SELECT size, time, commandL, senderName, originalSenderName,
destName, mail
```

```
        FROM ForwardedMessage
        WHERE ((senderName = ? AND destName = ?)
           OR (senderName = ? AND destName = ?)
           OR destName = 'TO_ALL'
           OR destName = 'TOALL')
    )";
    retcode = SQLAllocHandle(SQL_HANDLE_STMT, hdbc, &hstmt);
    if (retcode != SQL_SUCCESS && retcode != SQL_SUCCESS_WITH_INFO)
{
        std::cerr << "RetrieveForwardedMessages: failed to allocate
statement handle." << std::endl;
        return messages;
    }
    retcode = SQLPrepareA(hstmt, (SQLCHAR*)query, SQL_NTS);
    if (retcode != SQL_SUCCESS && retcode != SQL_SUCCESS_WITH_INFO)
{
        std::cerr << "RetrieveForwardedMessages: failed to prepare
statement." << std::endl;
        SQLFreeHandle(SQL_HANDLE_STMT, hstmt);
        return messages;
    }
    SQLLEN len1 = SQL_NTS;
    SQLLEN len2 = SQL_NTS;
    SQLLEN len3 = SQL_NTS;
    SQLLEN len4 = SQL_NTS;
    SQLBindParameter(hstmt, 1, SQL_PARAM_INPUT, SQL_C_CHAR,
SQL_VARCHAR,
        255, 0, (SQLPOINTER)parameter.c_str(), 0, &len1);
    SQLBindParameter(hstmt, 2, SQL_PARAM_INPUT, SQL_C_CHAR,
SQL_VARCHAR,
        255, 0, (SQLPOINTER)parameter2.c_str(), 0, &len2);
    SQLBindParameter(hstmt, 3, SQL_PARAM_INPUT, SQL_C_CHAR,
SQL_VARCHAR,
        255, 0, (SQLPOINTER)parameter2.c_str(), 0, &len3);
    SQLBindParameter(hstmt, 4, SQL_PARAM_INPUT, SQL_C_CHAR,
SQL_VARCHAR,
        255, 0, (SQLPOINTER)parameter.c_str(), 0, &len4);
    retcode = SQLExecute(hstmt);
    if (retcode != SQL_SUCCESS && retcode != SQL_SUCCESS_WITH_INFO)
{
        SQLCHAR sqlState[6], messageText[256];
        SQLINTEGER nativeError;
        SQLSMALLINT textLength;
        SQLGetDiagRecA(SQL_HANDLE_STMT, hstmt, 1, sqlState,
&nativeError,
            messageText, sizeof(messageText), &textLength);
        std::cerr << "RetrieveForwardedMessages SQL error: " <<
sqlState
```

```
                  << " - " << messageText << std::endl;
        SQLFreeHandle(SQL_HANDLE_STMT, hstmt);
        return messages;
    }
    std::cerr << "[RetrieveForwardedMessages] SQL query executed
successfully" << std::endl;
    int fetchCount = 0;
    SQLRETURN fetchResult;
    while ((fetchResult = SQLFetch(hstmt)) == SQL_SUCCESS) {
        fetchCount++;
        std::cerr << "[RetrieveForwardedMessages] Fetching row " <<
fetchCount << std::endl;
        Protocol::Message message;
        SQL_TIMESTAMP_STRUCT ts2   = {0};
        char commandBuf[4]         = {0};
        char senderBuf[512]        = {0};
        char origSenderBuf[512]    = {0};
        char destBuf[512]          = {0};
        char mailBuf[8192]         = {0};
        SQLGetData(hstmt, 1, SQL_C_LONG,           &message.size,
0,          nullptr);
        SQLGetData(hstmt, 2,
SQL_C_TYPE_TIMESTAMP,  &ts2,          sizeof(ts2), nullptr);
        SQLGetData(hstmt, 3,
SQL_C_CHAR,            commandBuf,    sizeof(commandBuf),    nullptr
);
        SQLGetData(hstmt, 4,
SQL_C_CHAR,            senderBuf,     sizeof(senderBuf),     nullptr
);
        SQLGetData(hstmt, 5, SQL_C_CHAR,            origSenderBuf,
sizeof(origSenderBuf), nullptr);
        SQLGetData(hstmt, 6,
SQL_C_CHAR,            destBuf,       sizeof(destBuf),       nullptr
);
        SQLGetData(hstmt, 7,
SQL_C_CHAR,            mailBuf,       sizeof(mailBuf),       nullptr
);
        message.time.tm_year = ts2.year - 1900;
        message.time.tm_mon  = ts2.month - 1;
        message.time.tm_mday = ts2.day;
        message.time.tm_hour = ts2.hour;
        message.time.tm_min  = ts2.minute;
        message.time.tm_sec  = ts2.second;
        std::cerr << "[RetrieveForwardedMessages] ts2: year=" <<
ts2.year << " mon=" << ts2.month
                  << " day=" << ts2.day << " hour=" << ts2.hour << "
min=" << ts2.minute << std::endl;
        message.commandL           = commandBuf[0];
```

```
        message.senderName         = std::string(senderBuf);
        message.originalSenderName = std::string(origSenderBuf);
        message.destName           = std::string(destBuf);
        message.mail               = std::string(mailBuf);
        messages.push_back(message);
    }
    if (fetchResult != SQL_SUCCESS && fetchResult != SQL_NO_DATA) {
        SQLCHAR sqlState[6], messageText[256];
        SQLINTEGER nativeError;
        SQLSMALLINT textLength;
        SQLGetDiagRecA(SQL_HANDLE_STMT, hstmt, 1, sqlState,
&nativeError,
            messageText, sizeof(messageText), &textLength);
        std::cerr << "[RetrieveForwardedMessages] SQLFetch error: "
<< sqlState
                  << " - " << messageText << std::endl;
    }
    std::cerr << "[RetrieveForwardedMessages] Found " <<
messages.size() << " messages" << std::endl;
    SQLFreeHandle(SQL_HANDLE_STMT, hstmt);
    return messages;
}
void DeleteMessage(SQLHDBC hdbc, const Protocol::Message& message) {
    SQLHSTMT hstmt;
    SQLRETURN retcode;
    int year  = message.time.tm_year + 1900;
    int month = message.time.tm_mon + 1;
    int day   = message.time.tm_mday;
    int hour  = message.time.tm_hour;
    int min   = message.time.tm_min;
    std::cerr << "[DeleteMessage] sender='" << message.senderName
              << "' dest='" << message.destName
              << "' mail='" << message.mail
              << "' time=" << year << "-" << month << "-" << day
              << " " << hour << ":" << min << std::endl;
    std::cerr << "[DeleteMessage] Raw tm values: year=" <<
message.time.tm_year
              << " mon=" << message.time.tm_mon << " day=" <<
message.time.tm_mday
              << " hour=" << message.time.tm_hour << " min=" <<
message.time.tm_min << std::endl;
    char sqlMsg[1024];
    char sqlFwd[1024];
    sprintf_s(sqlMsg, 1024,
        "DELETE TOP(1) FROM Message WHERE senderName='%s' AND
mail='%s' AND (destName='%s' OR destName='TO_ALL' OR
destName='TOALL')",
```

```
        message.senderName.c_str(), message.mail.c_str(),
message.destName.c_str());
    sprintf_s(sqlFwd, 1024,
        "DELETE TOP(1) FROM ForwardedMessage WHERE senderName='%s'
AND mail='%s' AND (destName='%s' OR destName='TO_ALL' OR
destName='TOALL')",
        message.senderName.c_str(), message.mail.c_str(),
message.destName.c_str());
    std::cerr << "[DeleteMessage] SQL1: " << sqlMsg << std::endl;
    retcode = SQLAllocHandle(SQL_HANDLE_STMT, hdbc, &hstmt);
    if (retcode == SQL_SUCCESS || retcode == SQL_SUCCESS_WITH_INFO)
{
        retcode = SQLExecDirectA(hstmt, (SQLCHAR*)sqlMsg, SQL_NTS);
        if (retcode == SQL_SUCCESS || retcode ==
SQL_SUCCESS_WITH_INFO) {
            SQLLEN rowCount = 0;
            SQLRowCount(hstmt, &rowCount);
            std::cerr << "[DeleteMessage] Deleted " << rowCount << "
rows from Message" << std::endl;
        } else {
            SQLCHAR sqlState[6] = {0};
            SQLINTEGER nativeError;
            SQLSMALLINT textLength;
            SQLGetDiagRecA(SQL_HANDLE_STMT, hstmt, 1, sqlState,
&nativeError, nullptr, 0, &textLength);
            std::cerr << "[DeleteMessage] Message error: " <<
(char*)sqlState << std::endl;
        }
        SQLFreeHandle(SQL_HANDLE_STMT, hstmt);
    }
    std::cerr << "[DeleteMessage] SQL2: " << sqlFwd << std::endl;
    retcode = SQLAllocHandle(SQL_HANDLE_STMT, hdbc, &hstmt);
    if (retcode == SQL_SUCCESS || retcode == SQL_SUCCESS_WITH_INFO)
{
        retcode = SQLExecDirectA(hstmt, (SQLCHAR*)sqlFwd, SQL_NTS);
        if (retcode == SQL_SUCCESS || retcode ==
SQL_SUCCESS_WITH_INFO) {
            SQLLEN rowCount = 0;
            SQLRowCount(hstmt, &rowCount);
            std::cerr << "[DeleteMessage] Deleted " << rowCount << "
rows from ForwardedMessage" << std::endl;
        } else {
            SQLCHAR sqlState[6] = {0};
            SQLINTEGER nativeError;
            SQLSMALLINT textLength;
            SQLGetDiagRecA(SQL_HANDLE_STMT, hstmt, 1, sqlState,
&nativeError, nullptr, 0, &textLength);
```

```
            std::cerr << "[DeleteMessage] ForwardedMessage error: "
<< (char*)sqlState << std::endl;
        }
        SQLFreeHandle(SQL_HANDLE_STMT, hstmt);
    }
}
void StoreAIChat(SQLHDBC hdbc, const std::string& username, const
std::string& role, const std::string& message) {
    SQLHSTMT hstmt;
    const char* sql = "INSERT INTO AIChat (username, role, message)
VALUES (?, ?, ?)";
    SQLAllocHandle(SQL_HANDLE_STMT, hdbc, &hstmt);
    SQLPrepareA(hstmt, (SQLCHAR*)sql, SQL_NTS);
    SQLLEN l1 = SQL_NTS, l2 = SQL_NTS, l3 = SQL_NTS;
    SQLBindParameter(hstmt, 1, SQL_PARAM_INPUT, SQL_C_CHAR,
SQL_VARCHAR, 255, 0, (SQLPOINTER)username.c_str(), 0, &l1);
    SQLBindParameter(hstmt, 2, SQL_PARAM_INPUT, SQL_C_CHAR,
SQL_VARCHAR, 10,  0, (SQLPOINTER)role.c_str(),     0, &l2);
    SQLBindParameter(hstmt, 3, SQL_PARAM_INPUT, SQL_C_CHAR,
SQL_VARCHAR, 4000,0, (SQLPOINTER)message.c_str(),  0, &l3);
    SQLExecute(hstmt);
    SQLFreeHandle(SQL_HANDLE_STMT, hstmt);
}
std::vector<std::pair<std::string,std::string>>
RetrieveAIChat(SQLHDBC hdbc, const std::string& username) {
    std::vector<std::pair<std::string,std::string>> history;
    SQLHSTMT hstmt;
    char sql[512];
    sprintf_s(sql, 512,
        "SELECT role, message FROM AIChat WHERE username='%s' ORDER
BY time ASC",
        username.c_str());
    SQLAllocHandle(SQL_HANDLE_STMT, hdbc, &hstmt);
    SQLRETURN ret = SQLExecDirectA(hstmt, (SQLCHAR*)sql, SQL_NTS);
    if (ret == SQL_SUCCESS || ret == SQL_SUCCESS_WITH_INFO) {
        char roleBuf[16] = {0};
        char msgBuf[4096] = {0};
        while (SQLFetch(hstmt) == SQL_SUCCESS) {
            memset(roleBuf, 0, sizeof(roleBuf));
            memset(msgBuf, 0, sizeof(msgBuf));
            SQLGetData(hstmt, 1, SQL_C_CHAR, roleBuf,
sizeof(roleBuf), nullptr);
            SQLGetData(hstmt, 2, SQL_C_CHAR,
msgBuf,  sizeof(msgBuf),  nullptr);
            history.push_back({std::string(roleBuf),
std::string(msgBuf)});
        }
    }
```

```
    SQLFreeHandle(SQL_HANDLE_STMT, hstmt);
    return history;
}
std::string CallAIAPI(const std::string& userMessage) {
    int wlen = MultiByteToWideChar(1251, 0, userMessage.c_str(), -1,
NULL, 0);
    std::wstring wstr(wlen, 0);
    MultiByteToWideChar(1251, 0, userMessage.c_str(), -1, &wstr[0],
wlen);
    int ulen = WideCharToMultiByte(CP_UTF8, 0, wstr.c_str(), -1,
NULL, 0, NULL, NULL);
    std::string utf8msg(ulen, 0);
    WideCharToMultiByte(CP_UTF8, 0, wstr.c_str(), -1, &utf8msg[0],
ulen, NULL, NULL);
    if (!utf8msg.empty() && utf8msg.back() == '\0')
utf8msg.pop_back();
    std::string escaped = utf8msg;
    std::string result = "";
    size_t pos = 0;
    while ((pos = escaped.find('"', pos)) != std::string::npos) {
        escaped.replace(pos, 1, "\\\"");
        pos += 2;
    }
    std::string body = "{\"model\":\"nvidia/nemotron-3-super-120b-
a12b:free\",\"messages\":[{\"role\":\"system\",\"content\":\"You are
a helpful assistant. Do not use emoji in your
responses.\"},{\"role\":\"user\",\"content\":\"" + escaped +
"\"}]}";
    HINTERNET hSession = WinHttpOpen(L"ServerSide/1.0",
        WINHTTP_ACCESS_TYPE_DEFAULT_PROXY, WINHTTP_NO_PROXY_NAME,
WINHTTP_NO_PROXY_BYPASS, 0);
    if (!hSession) { std::cerr << "[AI] WinHttpOpen failed" <<
std::endl; return "Error: no session"; }
    HINTERNET hConnect = WinHttpConnect(hSession, L"openrouter.ai",
INTERNET_DEFAULT_HTTPS_PORT, 0);
    if (!hConnect) { WinHttpCloseHandle(hSession); return "Error: no
connect"; }
    HINTERNET hRequest = WinHttpOpenRequest(hConnect, L"POST",
L"/api/v1/chat/completions",
        NULL, WINHTTP_NO_REFERER, WINHTTP_DEFAULT_ACCEPT_TYPES,
WINHTTP_FLAG_SECURE);
    if (!hRequest) { WinHttpCloseHandle(hConnect);
WinHttpCloseHandle(hSession); return "Error: no request"; }
    std::wstring headers = L"Content-Type:
application/json\r\nAuthorization: Bearer sk-or-v1-
a3e076e8abddb017050d48f32507064351c968687608e7b4eb9d5acdc66d7f41\r\n
HTTP-Referer: http://localhost\r\n";
    BOOL sent = WinHttpSendRequest(hRequest,
```

```
        headers.c_str(), (DWORD)headers.size(),
        (LPVOID)body.c_str(), (DWORD)body.size(),
(DWORD)body.size(), 0);
    if (!sent || !WinHttpReceiveResponse(hRequest, NULL)) {
        WinHttpCloseHandle(hRequest); WinHttpCloseHandle(hConnect);
WinHttpCloseHandle(hSession);
        return "Error: send failed";
    }
    std::string response;
    DWORD bytesAvail = 0;
    while (WinHttpQueryDataAvailable(hRequest, &bytesAvail) &&
bytesAvail > 0) {
        char* buf = new char[bytesAvail + 1];
        DWORD bytesRead = 0;
        WinHttpReadData(hRequest, buf, bytesAvail, &bytesRead);
        buf[bytesRead] = '\0';
        response += buf;
        delete[] buf;
    }
    WinHttpCloseHandle(hRequest);
    WinHttpCloseHandle(hConnect);
    WinHttpCloseHandle(hSession);
    std::cerr << "[AI] Response: " << response << std::endl;
    size_t contentPos = response.find("\"content\":");
    if (contentPos == std::string::npos) return "Error: no content
in response";
    contentPos += 10; (response[contentPos] == ' ' ||
response[contentPos] == '\n')) contentPos++;
    if (response[contentPos] == '"') contentPos++;
    std::string aiText;
    while (contentPos < response.size() && response[contentPos] !=
'"') {
        if (response[contentPos] == '\\' && contentPos + 1 <
response.size()) {
            contentPos++;
            if (response[contentPos] == 'n') aiText += '\n';
            else if (response[contentPos] == '"') aiText += '"';
            else aiText += response[contentPos];
        } else {
            aiText += response[contentPos];
        }
        contentPos++;
    }
    if (aiText.empty()) return "No response";
    int wlen2 = MultiByteToWideChar(CP_UTF8, 0, aiText.c_str(), -1,
NULL, 0);
    std::wstring wstr2(wlen2, 0);
```

```
    MultiByteToWideChar(CP_UTF8, 0, aiText.c_str(), -1, &wstr2[0],
wlen2);
    int clen = WideCharToMultiByte(1251, 0, wstr2.c_str(), -1, NULL,
0, "?", NULL);
    std::string cp1251(clen, 0);
    WideCharToMultiByte(1251, 0, wstr2.c_str(), -1, &cp1251[0],
clen, "?", NULL);
    if (!cp1251.empty() && cp1251.back() == '\0') cp1251.pop_back();
    return cp1251;
}
```

Листинг клиентского приложения «ClientOOS»

Файл Client.cpp

```
#include "Client.h"
int Client::Start() {
    std::cerr << "[Info] Starting the client.\n"s;

    // Get server address
    addrinfo* conn_addr = ResolveConnectionAddress();
    if (!conn_addr) {
        return -1;
    }

    // Create connection socket for the client
    connection_socket_ = CreateConnectionSocket(conn_addr);
    if (!IS_VALID_SOCKET(connection_socket_)) {
        return -1;
    }

    std::cerr << "[Info] Connecting to remote host ("s <<
remote_hostname_ << ":"s << remote_port_ << ")\n"s;
    if (connect(connection_socket_, conn_addr->ai_addr, conn_addr-
>ai_addrlen) == -1) {
        std::cerr << "[Error] Failed to connect to the remote host:
connect(): "s << std::system_category().message(GET_SOCKET_ERRNO())
<< std::endl;
        return -1;
    }
    std::cerr << "[Info] Successfully connected to "s <<
remote_hostname_ << ":"s << remote_port_ << std::endl;

    freeaddrinfo(conn_addr); // clean up ununsed resources

    return HandleConnection();
```

```
}

void Client::Disconnect() {
    std::cerr << "[Info] Disconnecting from the remote server.\n"s;
    CLOSE_SOCKET(connection_socket_);
    std::cerr << "[Info] Disconnected."s << std::endl;
}
addrinfo* Client::ResolveConnectionAddress()  {
    std::cerr << "[Debug] Resolving remote server address.\n"s;
    // Create helper and connection data structures for
getaddrinfo() call
    addrinfo hints, * connection_address;
    memset(&hints, 0x00, sizeof(hints));
    hints.ai_family = AF_INET; // use IPv4 for connection
    hints.ai_socktype = SOCK_STREAM; // use TCP

    if (getaddrinfo(remote_hostname_.data(), remote_port_.data(),
&hints, &connection_address) == -1) {
        std::cerr << "[Error] Failed to resolve remote host address:
getaddrinfo(): "s <<
std::system_category().message(GET_SOCKET_ERRNO()) << std::endl;
        return nullptr;
    }

    return connection_address;
}

// @return `-1` on error, valid socket on success.
SOCKET Client::CreateConnectionSocket(addrinfo* conn_addr) {
    if (!conn_addr) {
        std::cerr << "[Error] CreateConnectionSocket(): conn_addr is
NULL."s << std::endl;
        return -1;
    }

    std::cerr << "[Debug] Creating new connection socket.\n"s;
    SOCKET new_conn_socket = socket(conn_addr->ai_family, conn_addr-
>ai_socktype, conn_addr->ai_protocol);
    if (!IS_VALID_SOCKET(new_conn_socket)) {
        std::cerr << "[Error] Failed to create a new connection
socket: socket(): "s <<
std::system_category().message(GET_SOCKET_ERRNO()) << std::endl;
        return -1;
    }
    return new_conn_socket;
}
int Client::SendMessage(const std::string& message) {
    std::string assembled_msg(message);
```

```
    PrependMessageLength(assembled_msg);
    int total_bytes = assembled_msg.size();
    int sent_bytes = 0;
    int sent_n;
    std::cerr << "[Debug] Sending message: '"s << assembled_msg <<
"'\n"s;
    while (total_bytes > sent_bytes) {
        sent_n = send(connection_socket_, assembled_msg.data() +
sent_bytes, total_bytes - sent_bytes, 0);
        if (sent_n == -1) {
            std::cerr << "[Error] Failed to send data to the remote
host: send(): "s <<
std::system_category().message(GET_SOCKET_ERRNO()) << std::endl;
            return sent_n;
        }
        sent_bytes += sent_n;
        std::cerr << "[Debug] Sending "s << sent_n << " bytes to the
remote host\n"s;
    }


    return sent_bytes;
}

int Client::ReceiveMessage(char* writable_buff) {

    // Receive packet length (first 4 bytes)
    char msg_len_str[5]; // 4 bytes + 1 byte for the null-
terminating character
    memset(msg_len_str, 0x00, sizeof(msg_len_str));
    msg_len_str[4] = '\0';

    int recv_bytes = recv(connection_socket_, msg_len_str,
sizeof(msg_len_str) - 1, 0);
    if (recv_bytes <= 0) { // either client disconnect or an error
        return recv_bytes;
    }
    std::cerr << "[Debug] Received "s << recv_bytes << " bytes
(packet length): '"s << msg_len_str << "\n"s;
    // Check if the message conforms to the protocol
    for (const char c : std::string(msg_len_str)) {
        if (!std::isdigit(c)) {
            std::cerr << "[Error] Failed to read data from the
remote host: invalid protocol format.\n"s;
            return -1;
        }
    }
```

```
    int packet_length = std::atoi(msg_len_str);
    recv_bytes = recv(connection_socket_, writable_buff,
packet_length, 0);
    if (recv_bytes <= 0) { // Check for errors
        return recv_bytes;
    }
    std::cerr << "[Debug] Received "s << recv_bytes << " bytes
(actual packet)\n"s << writable_buff << "\n"s;

    return recv_bytes;
}
void Client::PrintInputPrompt() const {
    std::cin.clear();
    std::cout << " >>> "s;
    std::cout.flush();
}

int Client::InputHandler() {
    while (true) {
        char msg_buff[MAX_DATA_BUFFER_SIZE];
        PrintInputPrompt();

        std::fgets(msg_buff, MAX_DATA_BUFFER_SIZE, stdin);
        std::string message_str(msg_buff);
        message_str.pop_back(); // fgets() adds \n char to the end
of the string
        if (SendMessage(message_str) == -1) {
            std::exit(1);
        }
        memset(msg_buff, 0x00, MAX_DATA_BUFFER_SIZE);
    }
}

int Client::HandleConnection()  {
    std::thread input_worker_thread(&Client::InputHandler, this); //
Create a new thread for reading user input
    input_worker_thread.detach();
    while (true) {
        char msg_buff[MAX_DATA_BUFFER_SIZE];
        memset(msg_buff, 0x00, sizeof(msg_buff));

        int recv_bytes = ReceiveMessage(msg_buff);
        if (recv_bytes <= 0) {
            if (recv_bytes == 0) {
                std::cerr << "[Info] Remote host has closed the
connection."s << std::endl;
                std::exit(1);
            }
```

```
            else {
                std::cerr << "[Error] Failed to receive data from
the remote host: recv(): "s <<
std::system_category().message(GET_SOCKET_ERRNO()) << std::endl;
                std::exit(1);
            }
        }
        std::cout << msg_buff << '\n';

        PrintInputPrompt();
    }
}
```

Файл ClientConsole.cpp

```
#include "Common.h"
#include "Client.h"
int main(int argc, char* argv[]) {
    if (argc != 3) {
        std::cerr << "[Usage] client <remote_address>
<remote_port>"s << std::endl;
        return 1;
    }
#ifdef _WIN32
    WSADATA d;
    if (WSAStartup(MAKEWORD(2, 2), &d)) {
        std::cerr << "[Error] Failed to initialize WinSockAPI: "s <<
std::system_category().message(GET_SOCKET_ERRNO()) << std::endl;
        return 1;
    }
#endif

    Client client(argv[1], argv[2]);
    if (client.Start() == -1) {
        return 1;
    }
}
```

Файл Chat.cpp

```
#include "Chat.h"
#include "Login.h"
#include "Client.h"
#include "Common.h"
#include <thread>
#include <iostream>
#include <format>
#include <functional>
```

```
#include <vector>
using namespace std;
using namespace Vis;
using namespace System;
using namespace System::Windows::Forms;
using namespace System::Runtime::InteropServices;
void SideSocketProcces(const string& a, const string& b);
int consts = 0;
Client clie;
std::ifstream inputFile;
[STAThread]
int WINAPI WinMain
(
    HINSTANCE hInstance, //
    HINSTANCE hPrevInstance,//
    LPSTR lpCmdLine, //
    int nCmdShow  //
){
#ifdef _WIN32
    WSADATA d;
    if (WSAStartup(MAKEWORD(2, 2), &d)) {
        std::cerr << "[Error] Failed to initialize WinSockAPI: "s <<
std::system_category().message(GET_SOCKET_ERRNO()) << std::endl;
        return 1;
    }
#endif
    // Установка кодировки для правильного отображения русского
текста
    SetConsoleCP(1251);
    SetConsoleOutputCP(1251);
    setlocale(LC_ALL, "Russian");

    Application::EnableVisualStyles();
    Application::SetCompatibleTextRenderingDefault(false);
    Window^ win = gcnew Window;
    win->winPtr = &win;
    LoginForm^ auth = gcnew LoginForm;
    auth->win = &win;

    auth->Show();
    string ip;
    string port;
    bool configRead = ReadConnectionInfo("Config.txt", ip, port);

    if (!configRead || ip.empty() || port.empty()) {
        ip = "127.0.0.1";
        port = "4000";
    }
```

```
    thread  socke(SideSocketProcces, ip, port);
    socke.detach();

    // Даём время на подключение
    Sleep(2000);




    Application::Run(win);

}


void SideSocketProcces(const string &host, const string &port) {
    clie.SetHost(host, port);
    clie.Start();
}

System::Void Window::button2_Click(System::Object^ sender,
System::EventArgs^ e) {


    char* msg_buff =
(char*)(void*)Marshal::StringToHGlobalAnsi(textBox2->Text);
    std::string message_str(msg_buff);

    // If a file is selected, use filename as the message text
    if (fileName != nullptr && fileName != " ") {
        std::string file =
string((char*)(void*)Marshal::StringToHGlobalAnsi(fileName));
        std::size_t found = file.find_last_of("/\\");
        message_str = "[File: " + file.substr(found + 1) + "]";
    }

    if (message_str.empty() || message_str == " ") return;
    Protocol::Message mes;
    time_t mytime = time(NULL);
    mes.time = *localtime(&mytime);
    mes.commandL = TEXTMES;
    mes.destName = Globals::dest;
    mes.senderName = Globals::user;
    mes.mail = message_str;
    //std::cerr << "[Info] MAIL SODERJ."s << mes.mail << std::endl;
    Globals::Mutex1.lock();
    Globals::assembleLine.push(mes);
    Globals::Mutex1.unlock();
```

```
    char* somemes = clie.ProtocolComposition(mes);
    if (clie.SendMessage(somemes) == -1) {
        std::exit(1);
    }
    textBox2->Clear();
    if (fileName != " " &&  fileName!=nullptr) {

        std::string file=
string((char*)(void*)Marshal::StringToHGlobalAnsi(fileName));

        // Read entire file into memory
        std::ifstream f(file, std::ios::binary | std::ios::ate);
        if (!f.is_open()) {
            std::cerr << "Failed to open file: " << file <<
std::endl;
            fileName = nullptr;
            return;
        }
        std::streamsize fileSize = f.tellg();
        f.seekg(0, std::ios::beg);
        std::vector<char> fileData(fileSize);
        if (!f.read(fileData.data(), fileSize)) {
            std::cerr << "Failed to read file" << std::endl;
            fileName = nullptr;
            return;
        }
        f.close();

        std::size_t found = file.find_last_of("/\\");
        std::string fname = file.substr(found + 1);

        // Send one FILE packet with size, then raw bytes
        Protocol::Message fmes;
        time_t mytime2 = time(NULL);
        fmes.time = *localtime(&mytime2);
        fmes.commandL = FILE;
        fmes.mail = std::to_string(fileSize);
        fmes.fileName = fname;
        fmes.destName = Globals::dest;
        fmes.senderName = Globals::user;
        clie.SendMessage(clie.ProtocolComposition(fmes));
        // Send raw bytes
        int sent = 0;
        while (sent < (int)fileSize) {
            int r = send(clie.connection_socket_, fileData.data() +
sent, (int)fileSize - sent, 0);
            if (r <= 0) break;
            sent += r;
```

```
        }
        std::cerr << "[Client] Sent file: " << fname << " (" <<
fileSize << " bytes)" << std::endl;
        fileName = nullptr;
        textBox2->Clear();
    }
}
System::Void Window::UpdateChat(Protocol::Message assembled) {

    char buf[300];

    if (assembled.commandL == FORWARD) {
        sprintf_s(buf, 300,
            "%02d/%02d/%02d [%02d:%02d] %s: [Forwarded from %s]
%s\r\n",
            assembled.time.tm_mday, assembled.time.tm_mon + 1,
assembled.time.tm_year + 1900,
            assembled.time.tm_hour, assembled.time.tm_min,
            assembled.senderName.c_str(),
            assembled.originalSenderName.c_str(),
            assembled.mail.c_str());
    } else {
        sprintf_s(buf, 300,
            "%02d/%02d/%02d [%02d:%02d] %s: %s\r\n",
            assembled.time.tm_mday, assembled.time.tm_mon + 1,
assembled.time.tm_year + 1900,
            assembled.time.tm_hour, assembled.time.tm_min,
            assembled.senderName.c_str(),
            assembled.mail.c_str());
    }

    System::String^ sys = gcnew String(buf);
    AppendTextSafe(sys);

}
System::Void LoginForm::LoginForm_FormClosing(System::Object^
sender, System::Windows::Forms::FormClosingEventArgs^ e) {
    if (!this->IsAutorised){
        (*this->win)->Close();
}

}
System::Void Window::Window_FormClosing(System::Object^ sender,
System::Windows::Forms::FormClosingEventArgs^ e) {
    if (isLoggingOut) {
        e->Cancel = true;
        isLoggingOut = false;
        return;
```

```
    }
    clie.Disconnect();
}

System::Void
Window::listBox1_SelectionChangeCommitted(System::Object^ sender,
System::EventArgs^ e) {

    bool fl = false;
    string g =
string((char*)(void*)Marshal::StringToHGlobalAnsi(listBox1-
>SelectedItem->ToString()));

    g.erase(std::remove(g.begin(), g.end(), ' '), g.end());
    for (auto& i : Globals::OnlineList) {
        if (i == g) {
            fl = true;
        }
    }
    if (listBox1->SelectedIndex == 0) {
        // TO ALL - hide online/offline status
        label2->Text = "";
        Globals::dest = "TO_ALL";
    } else {
        if (fl) {
            label2->Text = isRussian ? gcnew String("\xC2
\xF1\xE5\xF2\xE8") : L"Online";
            label2->ForeColor = System::Drawing::Color::ForestGreen;
        } else {
            label2->Text = isRussian ? gcnew String("\xCD\xE5 \xE2
\xF1\xE5\xF2\xE8") : L"Offline";
            label2->ForeColor = System::Drawing::Color::Crimson;
        }
        Globals::dest = g;
    }
    textBox1->Clear();
    Protocol::Message mes;
    mes.commandL = HISTORY;
    mes.destName = Globals::dest;
    mes.senderName = Globals::user;
    time_t mytime = time(NULL);
    mes.time = *localtime(&mytime);
    mes.mail = "/";
    clie.SendMessage(clie.ProtocolComposition(mes));
}
```

```
System::Void LoginForm::Sign_In(System::Object^ sender,
System::EventArgs^ e) {
    bool ru = isRussian;
    String^ loginPlaceholder = ru ? gcnew
String("\xC2\xE2\xE5\xE4\xE8\xF2\xE5 \xEB\xEE\xE3\xE8\xED") :
L"Enter your login";
    String^ passPlaceholder  = ru ? gcnew
String("\xC2\xE2\xE5\xE4\xE8\xF2\xE5 \xEF\xE0\xF0\xEE\xEB\xFC") :
L"Enter your password";

    if (textBox1->Text == loginPlaceholder || textBox2->Text ==
passPlaceholder ||
        String::IsNullOrWhiteSpace(textBox1->Text) ||
String::IsNullOrWhiteSpace(textBox2->Text)) {
        MessageBox::Show(
            ru ? gcnew String("\xC7\xE0\xEF\xEE\xEB\xED\xE8\xF2\xE5
\xEB\xEE\xE3\xE8\xED \xE8 \xEF\xE0\xF0\xEE\xEB\xFC") : L"Please fill
in login and password",
            ru ? gcnew String("\xCE\xF8\xE8\xE1\xEA\xE0") :
L"Error",
            MessageBoxButtons::OK, MessageBoxIcon::Warning);
        return;
    }

    int connectionAttempts = 0;
    while (clie.GetStatus() != 1 && connectionAttempts < 50) {
        Sleep(100);
        connectionAttempts++;
        UpdateStatusWi();
    }

    if (clie.GetStatus() == 1) {
        Protocol::User us;
        us.name =
string((char*)(void*)Marshal::StringToHGlobalAnsi(textBox1->Text));
        us.password =
string((char*)(void*)Marshal::StringToHGlobalAnsi(textBox2->Text));
        Protocol::Message mes;
        mes.authData = us;
        mes.commandL = LOG_IN;
        Globals::result = 0;
        clie.SendMessage(clie.ProtocolComposition(mes));
        for (int i = 0; i < 10; i++) {
            if (Globals::result == 1) {
                Globals::user = us.name;
                this->IsAutorised = true;
                (*this->win)->Opacity = 1;
```

```
                (*this->win)->labelProfile->Text = (isRussian ?
gcnew String("\xCF\xF0\xEE\xF4\xE8\xEB\xFC: ") : L"Profile: ") +
gcnew String(us.name.c_str());
                (*this->win)->isRussian = isRussian;
                (*this->win)->ApplyLanguage();
                this->Close();
                (*this->win)->ShowInTaskbar = true;
                if (!(*this->win)->backgroundWorker1->IsBusy)
                    (*this->win)->backgroundWorker1-
>RunWorkerAsync();
                break;
            }
            else { cout << "wrong auth data" << endl; }
            Sleep(100);
        }
        if (Globals::result != 1) {
            MessageBox::Show(
                ru ? gcnew String("\xCD\xE5\xE2\xE5\xF0\xED\xFB\xE9
\xEB\xEE\xE3\xE8\xED \xE8\xEB\xE8 \xEF\xE0\xF0\xEE\xEB\xFC") :
L"Wrong login or password",
                ru ? gcnew String("\xCE\xF8\xE8\xE1\xEA\xE0
\xE0\xE2\xF2\xEE\xF0\xE8\xE7\xE0\xF6\xE8\xE8") : L"Authentication
Error",
                MessageBoxButtons::OK, MessageBoxIcon::Error);
        }
    }
    else {
        MessageBox::Show(
            ru ? gcnew String("\xCD\xE5 \xF3\xE4\xE0\xEB\xEE\xF1\xFC
\xEF\xEE\xE4\xEA\xEB\xFE\xF7\xE8\xF2\xFC\xF1\xFF \xEA
\xF1\xE5\xF0\xE2\xE5\xF0\xF3.") : L"Cannot connect to server.\nCheck
that server is running and Config.txt is correct.",
            ru ? gcnew String("\xCE\xF8\xE8\xE1\xEA\xE0
\xEF\xEE\xE4\xEA\xEB\xFE\xF7\xE5\xED\xE8\xFF") : L"Connection
Error",
            MessageBoxButtons::OK, MessageBoxIcon::Error);
    }
}
System::Void LoginForm::Register(System::Object^ sender,
System::EventArgs^ e) {
    bool ru = isRussian;
    String^ loginPlaceholder = ru ? gcnew
String("\xC2\xE2\xE5\xE4\xE8\xF2\xE5 \xEB\xEE\xE3\xE8\xED") :
L"Enter your login";
    String^ passPlaceholder  = ru ? gcnew
String("\xC2\xE2\xE5\xE4\xE8\xF2\xE5 \xEF\xE0\xF0\xEE\xEB\xFC") :
L"Enter your password";
```

```
    if (textBox1->Text == loginPlaceholder || textBox2->Text ==
passPlaceholder ||
        String::IsNullOrWhiteSpace(textBox1->Text) ||
String::IsNullOrWhiteSpace(textBox2->Text)) {
        MessageBox::Show(
            ru ? gcnew String("\xC7\xE0\xEF\xEE\xEB\xED\xE8\xF2\xE5
\xEB\xEE\xE3\xE8\xED \xE8 \xEF\xE0\xF0\xEE\xEB\xFC") : L"Please fill
in login and password",
            ru ? gcnew String("\xCE\xF8\xE8\xE1\xEA\xE0") :
L"Error",
            MessageBoxButtons::OK, MessageBoxIcon::Warning);
        return;
    }

    int connectionAttempts = 0;
    while (clie.GetStatus() != 1 && connectionAttempts < 50) {
        Sleep(100);
        connectionAttempts++;
        UpdateStatusWi();
    }

    if (clie.GetStatus() == 1) {
        Protocol::User us;
        us.name =
string((char*)(void*)Marshal::StringToHGlobalAnsi(textBox1->Text));
        us.password =
string((char*)(void*)Marshal::StringToHGlobalAnsi(textBox2->Text));
        Protocol::Message mes;
        mes.authData = us;
        mes.commandL = REGISTER;
        clie.SendMessage(clie.ProtocolComposition(mes));
        MessageBox::Show(
            ru ? gcnew
String("\xD0\xE5\xE3\xE8\xF1\xF2\xF0\xE0\xF6\xE8\xFF
\xEE\xF2\xEF\xF0\xE0\xE2\xEB\xE5\xED\xE0!") : L"Registration
sent!\nYou can now login with these credentials.",
            ru ? gcnew String("\xD3\xF1\xEF\xE5\xF8\xED\xEE") :
L"Success",
            MessageBoxButtons::OK, MessageBoxIcon::Information);
    }
    else {
        MessageBox::Show(
            ru ? gcnew String("\xCD\xE5 \xF3\xE4\xE0\xEB\xEE\xF1\xFC
\xEF\xEE\xE4\xEA\xEB\xFE\xF7\xE8\xF2\xFC\xF1\xFF \xEA
\xF1\xE5\xF0\xE2\xE5\xF0\xF3.") : L"Cannot connect to server.\nCheck
that server is running and Config.txt is correct.",
```

```
            ru ? gcnew String("\xCE\xF8\xE8\xE1\xEA\xE0
\xEF\xEE\xE4\xEA\xEB\xFE\xF7\xE5\xED\xE8\xFF") : L"Connection
Error",
            MessageBoxButtons::OK, MessageBoxIcon::Error);
    }
}
System::Void LoginForm::UpdateStatusWi() {
    if (clie.GetStatus() == 1) {
        UpdateLabel(isRussian ? gcnew
String("\xCF\xEE\xE4\xEA\xEB\xFE\xF7\xE5\xED\xEE") : L"Connected");
    }
    else {
        UpdateLabel(isRussian ? gcnew
String("\xCE\xF2\xEA\xEB\xFE\xF7\xE5\xED\xEE") : L"Disconnected");
    }
}
System::Void LoginForm::Recon(System::Object^ sender,
System::EventArgs^ e) {
    if (clie.GetStatus() == 0) {
        clie.Start();
    }
}

System::Void Window::textBox1_MouseDown(System::Object^ sender,
System::Windows::Forms::MouseEventArgs^ e) {
    if (e->Button == System::Windows::Forms::MouseButtons::Right) {
        // Показать контекстное меню только если есть выделенный
текст
        if (textBox1->SelectionLength > 0) {
            contextMenuStrip1->Show(textBox1, e->Location);
        }
    }
}

System::Void Window::forwardMenuItem_Click(System::Object^ sender,
System::EventArgs^ e) {
    String^ lineStr;

    // Проверяем есть ли выделенный текст
    if (!String::IsNullOrEmpty(textBox1->SelectedText)) {
        lineStr = textBox1->SelectedText;

        // Если выделено несколько строк, берем только первую
        int newlinePos = lineStr->IndexOf('\n');
        if (newlinePos > 0) {
            lineStr = lineStr->Substring(0, newlinePos);
        }
    } else {
```

```
        // Если нет выделения, ищем строку под курсором
        int selectionStart = textBox1->SelectionStart;
        String^ fullText = textBox1->Text;

        if (String::IsNullOrEmpty(fullText)) {
            MessageBox::Show("No messages to forward", "Empty Chat",
                             MessageBoxButtons::OK,
MessageBoxIcon::Warning);
            return;
        }

        // Найти начало и конец строки
        int lineStart = selectionStart;
        while (lineStart > 0 && fullText[lineStart - 1] != '\n') {
            lineStart--;
        }

        int lineEnd = selectionStart;
        while (lineEnd < fullText->Length && fullText[lineEnd] !=
'\n' && fullText[lineEnd] != '\r') {
            lineEnd++;
        }

        if (lineEnd <= lineStart) {
            MessageBox::Show(
                isRussian ? gcnew
String("\xCF\xEE\xE6\xE0\xEB\xF3\xE9\xF1\xF2\xE0,
\xE2\xFB\xE1\xE5\xF0\xE8\xF2\xE5 \xF1\xF2\xF0\xEE\xEA\xF3
\xF1\xEE\xEE\xE1\xF9\xE5\xED\xE8\xFF") : L"Please select or click on
a message line",
                isRussian ? gcnew String("\xCD\xE5\xF2
\xE2\xFB\xE1\xEE\xF0\xE0") : L"No Selection",
                MessageBoxButtons::OK, MessageBoxIcon::Warning);
            return;
        }

        lineStr = fullText->Substring(lineStart, lineEnd -
lineStart);
    }

    std::string line =
string((char*)(void*)Marshal::StringToHGlobalAnsi(lineStr));

    // Убираем управляющие символы
    while (!line.empty() && (line.front() == ' ' || line.front() ==
'\t' || line.front() == '\r' || line.front() == '\n')) {
        line.erase(line.begin());
    }
```

```
    while (!line.empty() && (line.back() == ' ' || line.back() ==
'\t' || line.back() == '\n' || line.back() == '\r')) {
        line.pop_back();
    }

    if (line.empty()) {
        MessageBox::Show("Please select a message line with text",
"Empty Line",
                         MessageBoxButtons::OK,
MessageBoxIcon::Warning);
        return;
    }

    // Сначала найти закрывающую скобку времени "]"
    size_t bracketPos = line.find("]");
    if (bracketPos == std::string::npos) {
        MessageBox::Show("Selected text is not a message.\nPlease
select a line like:\n22/04/2026 [21:23] user: message",
                         "Invalid Format", MessageBoxButtons::OK,
MessageBoxIcon::Warning);
        return;
    }

    // Теперь искать двоеточие ПОСЛЕ скобки
    size_t colonPos = line.find(":", bracketPos);

    if (colonPos == std::string::npos) {
        MessageBox::Show("Selected text is not a message.\nPlease
select a line like:\n22/04/2026 [21:23] user: message",
                         "Invalid Format", MessageBoxButtons::OK,
MessageBoxIcon::Warning);
        return;
    }

    // Извлечь имя отправителя
    size_t senderStart = bracketPos + 1;
    while (senderStart < line.length() && line[senderStart] == ' ')
{
        senderStart++;
    }

    std::string senderPart = line.substr(senderStart, colonPos -
senderStart);

    // Извлечь сообщение
    size_t messageStart = colonPos + 1;
    while (messageStart < line.length() && line[messageStart] == '
') {
```

```
        messageStart++;
    }
    std::string messagePart = line.substr(messageStart);

    std::string originalSenderName;
    std::string mail;

    // Проверить пересланное сообщение
    if (messagePart.find("[Forwarded from ") == 0) {
        size_t endBracket = messagePart.find("]");
        if (endBracket != std::string::npos) {
            originalSenderName = messagePart.substr(16, endBracket -
16);
            size_t mailStart = endBracket + 1;
            while (mailStart < messagePart.length() &&
messagePart[mailStart] == ' ') {
                mailStart++;
            }
            mail = messagePart.substr(mailStart);
        } else {
            originalSenderName = senderPart;
            mail = messagePart;
        }
    } else {
        originalSenderName = senderPart;
        mail = messagePart;
    }

    // Создать диалог выбора пользователя
    Form^ forwardDialog = gcnew Form();
    forwardDialog->Text = isRussian ? gcnew
String("\xCF\xE5\xF0\xE5\xF1\xEB\xE0\xF2\xFC
\xF1\xEE\xEE\xE1\xF9\xE5\xED\xE8\xE5...") : L"Forward message
to...";
    forwardDialog->Size = System::Drawing::Size(300, 400);
    forwardDialog->StartPosition = FormStartPosition::CenterParent;
    forwardDialog->FormBorderStyle =
System::Windows::Forms::FormBorderStyle::FixedDialog;
    forwardDialog->MaximizeBox = false;
    forwardDialog->MinimizeBox = false;

    ListBox^ userListBox = gcnew ListBox();
    userListBox->Location = System::Drawing::Point(10, 10);
    userListBox->Size = System::Drawing::Size(260, 300);

    Globals::Mutex2.lock();
    userListBox->Items->Add("TO ALL");
    for (const auto& user : Globals::UserList) {
```

```
        if (user != Globals::user) {
            userListBox->Items->Add(gcnew String(user.c_str()));
        }
    }
    Globals::Mutex2.unlock();

    Button^ okButton = gcnew Button();
    okButton->Text = isRussian ? gcnew
String("\xCF\xE5\xF0\xE5\xF1\xEB\xE0\xF2\xFC") : L"Forward";
    okButton->Location = System::Drawing::Point(50, 320);
    okButton->Size = System::Drawing::Size(80, 25);
    okButton->DialogResult =
System::Windows::Forms::DialogResult::OK;

    Button^ cancelButton = gcnew Button();
    cancelButton->Text = isRussian ? gcnew
String("\xCE\xF2\xEC\xE5\xED\xE0") : L"Cancel";
    cancelButton->Location = System::Drawing::Point(150, 320);
    cancelButton->Size = System::Drawing::Size(80, 25);
    cancelButton->DialogResult =
System::Windows::Forms::DialogResult::Cancel;

    forwardDialog->Controls->Add(userListBox);
    forwardDialog->Controls->Add(okButton);
    forwardDialog->Controls->Add(cancelButton);
    forwardDialog->AcceptButton = okButton;
    forwardDialog->CancelButton = cancelButton;

    if (forwardDialog->ShowDialog() ==
System::Windows::Forms::DialogResult::OK &&
        userListBox->SelectedItem != nullptr) {

        std::string selectedUser =
string((char*)(void*)Marshal::StringToHGlobalAnsi(
            userListBox->SelectedItem->ToString()));

        selectedUser.erase(std::remove(selectedUser.begin(),
selectedUser.end(), ' '), selectedUser.end());

        Protocol::Message mes;
        time_t mytime = time(NULL);
        mes.time = *localtime(&mytime);
        mes.commandL = FORWARD;
        mes.senderName = Globals::user;
        mes.originalSenderName = originalSenderName;
        mes.destName = selectedUser;
        mes.mail = mail;
```

```
        char* forwardPacket = clie.ProtocolComposition(mes);
        if (clie.SendMessage(forwardPacket) == -1) {
            MessageBox::Show(
                isRussian ? gcnew String("\xCE\xF8\xE8\xE1\xEA\xE0
\xEF\xE5\xF0\xE5\xF1\xEB\xEB\xEA\xE8") : L"Failed to send forward
message",
                isRussian ? gcnew String("\xCE\xF8\xE8\xE1\xEA\xE0")
: L"Error",
                MessageBoxButtons::OK, MessageBoxIcon::Error);
        } else {
            MessageBox::Show(
                isRussian ? gcnew
String("\xD1\xEE\xEE\xE1\xF9\xE5\xED\xE8\xE5
\xEF\xE5\xF0\xE5\xF1\xEB\xE0\xED\xEE!") : L"Message forwarded
successfully!",
                isRussian ? gcnew
String("\xD3\xF1\xEF\xE5\xF8\xED\xEE") : L"Success",
                MessageBoxButtons::OK, MessageBoxIcon::Information);
        }
    }

    delete forwardDialog;
}

System::Void Window::deleteMenuItem_Click(System::Object^ sender,
System::EventArgs^ e) {
    // Берём всю строку где находится курсор (независимо от
выделения)
    String^ fullText = textBox1->Text;
    int cursorPos = textBox1->SelectionStart;

    // Находим начало строки
    int lineStart = cursorPos;
    while (lineStart > 0 && fullText[lineStart - 1] != '\n')
lineStart--;

    // Находим конец строки
    int lineEnd = cursorPos;
    while (lineEnd < fullText->Length && fullText[lineEnd] != '\r'
&& fullText[lineEnd] != '\n') lineEnd++;

    if (lineStart >= lineEnd) {
        MessageBox::Show(
            isRussian ? gcnew
String("\xCF\xEE\xE6\xE0\xEB\xF3\xE9\xF1\xF2\xE0,
\xEA\xEB\xE8\xEA\xED\xE8\xF2\xE5 \xED\xE0 \xF1\xF2\xF0\xEE\xEA\xF3
\xF1\xEE\xEE\xE1\xF9\xE5\xED\xE8\xFF") : L"Please click on a message
line to delete",
```

```
            isRussian ? gcnew String("\xCD\xE5\xF2
\xE2\xFB\xE1\xEE\xF0\xE0") : L"No Selection",
            MessageBoxButtons::OK, MessageBoxIcon::Warning);
        return;
    }

    // Подтверждение удаления
    System::Windows::Forms::DialogResult result = MessageBox::Show(
        isRussian ? gcnew String("\xC2\xFB
\xF3\xE2\xE5\xF0\xE5\xED\xFB, \xF7\xF2\xEE \xF5\xEE\xF2\xE8\xF2\xE5
\xF3\xE4\xE0\xEB\xE8\xF2\xFC \xFD\xF2\xEE
\xF1\xEE\xEE\xE1\xF9\xE5\xED\xE8\xE5?") : L"Are you sure you want to
delete this message?",
        isRussian ? gcnew
String("\xCF\xEE\xE4\xF2\xE2\xE5\xF0\xE6\xE4\xE5\xED\xE8\xE5
\xF3\xE4\xE0\xEB\xE5\xED\xE8\xFF") : L"Confirm Delete",
        MessageBoxButtons::YesNo,
        MessageBoxIcon::Question);

    if (result == System::Windows::Forms::DialogResult::Yes) {
        // Парсим строку чтобы извлечь данные
        // Формат: DD/MM/YYYY [HH:MM] sender: message

        String^ lineTextManaged = fullText->Substring(lineStart,
lineEnd - lineStart)->Trim();
        std::string lineStr =
string((char*)(void*)Marshal::StringToHGlobalAnsi(lineTextManaged));
        std::cerr << "[Delete] Parsing line: '" << lineStr << "'" <<
std::endl;

        // Извлекаем дату/время и отправителя
        size_t bracketPos = lineStr.find('[');
        size_t closeBracketPos = lineStr.find(']', bracketPos);
        size_t colonPos = lineStr.find(':', closeBracketPos);

        if (bracketPos == std::string::npos || colonPos ==
std::string::npos) {
            MessageBox::Show("Cannot parse message format", "Error",
                             MessageBoxButtons::OK,
MessageBoxIcon::Error);
            return;
        }

        // Извлекаем дату (DD/MM/YY)
        std::string dateStr = lineStr.substr(0, bracketPos - 1);
        // Извлекаем время [HH:MM]
        std::string timeStr = lineStr.substr(bracketPos + 1,
closeBracketPos - bracketPos - 1);
```

```
        // Извлекаем отправителя
        std::string senderStr = lineStr.substr(closeBracketPos + 2,
colonPos - closeBracketPos - 2);

        // Извлекаем сообщение
        std::string messageText = lineStr.substr(colonPos + 2);

        // Если пересланное сообщение - убираем префикс [Forwarded
from X]
        // В БД хранится только оригинальный текст без префикса
        if (messageText.size() > 16 && messageText.substr(0, 16) ==
"[Forwarded from ") {
            size_t fwdClose = messageText.find(']');
            if (fwdClose != std::string::npos && fwdClose + 2 <=
messageText.size()) {
                messageText = messageText.substr(fwdClose + 2);
            }
        }

        // Создаем DELETE сообщение
        Protocol::Message delMsg;
        delMsg.commandL = DELETE_MSG;
        delMsg.senderName = senderStr;
        delMsg.destName = Globals::dest;
        delMsg.mail = messageText;

        // Парсим время
        std::istringstream dateStream(dateStr);
        std::istringstream timeStream(timeStr);
        int day, month, year, hour, minute;
        char slash, colon;
        dateStream >> day >> slash >> month >> slash >> year;
        timeStream >> hour >> colon >> minute;

        delMsg.time.tm_mday = day;
        delMsg.time.tm_mon = month - 1;   // display is 1-12, tm_mon
is 0-11
        delMsg.time.tm_year = year - 1900; // display is full year
e.g. 2026, tm_year is years since 1900
        delMsg.time.tm_hour = hour;
        delMsg.time.tm_min = minute;
        delMsg.time.tm_sec = 0;

        // Отправляем на сервер
        char* deletePacket = clie.ProtocolComposition(delMsg);
        if (clie.SendMessage(deletePacket) == -1) {
            MessageBox::Show("Failed to send delete request",
"Error",
```

```
                                MessageBoxButtons::OK,
MessageBoxIcon::Error);
        } else {
            // Удаляем строку из UI (lineStart/lineEnd уже известны
сверху)
            int removeEnd = lineEnd;
            // Пропускаем \r\n
            if (removeEnd < fullText->Length && fullText[removeEnd]
== '\r') removeEnd++;
            if (removeEnd < fullText->Length && fullText[removeEnd]
== '\n') removeEnd++;

            String^ newText = fullText->Remove(lineStart, removeEnd
- lineStart);
            textBox1->Text = newText;
            textBox1->SelectionStart = lineStart;
            textBox1->SelectionLength = 0;
        }
    }
}


System::Void Window::btnLogout_Click(System::Object^ sender,
System::EventArgs^ e) {
    isLoggingOut = true;

    Globals::user = "";
    Globals::dest = "";
    Globals::OnlineList.clear();
    Globals::UserList.clear();
    textBox1->Clear();
    listBox1->Items->Clear();
    listBox1->Items->Add("TO ALL");

    LoginForm^ auth = gcnew LoginForm;
    auth->win = winPtr;
    auth->isRussian = isRussian;
    auth->Show();

    this->Opacity = 0;
    this->ShowInTaskbar = false;
    labelProfile->Text = isRussian ? gcnew
String("\xCF\xF0\xEE\xF4\xE8\xEB\xFC: ") : L"Profile: ";
}

System::Void Window::btnAIChat_Click(System::Object^ sender,
System::EventArgs^ e) {
    AIChatForm^ aiForm = gcnew AIChatForm();
```

```
    aiForm->isRussian = isRussian;
    aiForm->ApplyLanguage();
    aiForm->Show();
}
```

Файл Client.cpp

```
#include "Client.h"
int Client::Start() {
    std::cerr << "[Info] Starting the client.\n"s;
    // Get server address
    addrinfo* conn_addr = ResolveConnectionAddress();
    if (!conn_addr) {
        std::cerr << "[Error] Failed to resolve address\n"s;
        return -1;
    }

    // Create connection socket for the client
    connection_socket_ = CreateConnectionSocket(conn_addr);
    if (!IS_VALID_SOCKET(connection_socket_)) {
        std::cerr << "[Error] Failed to create socket\n"s;
        freeaddrinfo(conn_addr);
        return -1;
    }

    std::cerr << "[Info] Connecting to remote host ("s <<
remote_hostname_ << ":"s << remote_port_ << ")\n"s;
    if (connect(connection_socket_, conn_addr->ai_addr, conn_addr-
>ai_addrlen) == -1) {
        std::cerr << "[Error] Failed to connect to the remote host:
connect(): "s << std::system_category().message(GET_SOCKET_ERRNO())
<< std::endl;
        freeaddrinfo(conn_addr);
        return -1;
    }
    std::cerr << "[Info] Successfully connected to "s <<
remote_hostname_ << ":"s << remote_port_ << std::endl;
    status = 1;
    freeaddrinfo(conn_addr); // clean up ununsed resources

    // Запускаем HandleConnection в отдельном потоке чтобы не
блокировать
    std::thread receiver(&Client::HandleConnection, this);
    receiver.detach();

    return 0; // Возвращаем успех сразу после подключения
}
```

```
void Client::Disconnect() {
    std::cerr << "[Info] Disconnecting from the remote server.\n"s;
    CLOSE_SOCKET(connection_socket_);
    status = 0;
    std::cerr << "[Info] Disconnected."s << std::endl;

};
addrinfo* Client::ResolveConnectionAddress()  {
    std::cerr << "[Debug] Resolving remote server address.\n"s;
    // Create helper and connection data structures for
getaddrinfo() call
    addrinfo hints, * connection_address;
    memset(&hints, 0x00, sizeof(hints));
    hints.ai_family = AF_INET; // use IPv4 for connection
    hints.ai_socktype = SOCK_STREAM; // use TCP

    if (getaddrinfo(remote_hostname_.data(), remote_port_.data(),
&hints, &connection_address) == -1) {
        std::cerr << "[Error] Failed to resolve remote host address:
getaddrinfo(): "s <<
std::system_category().message(GET_SOCKET_ERRNO()) << std::endl;
        return nullptr;
    }

    return connection_address;
}

Protocol::Message Client::ProtocolDecomposition(const char*
message,int msg_len)
{

    Protocol::Message assembled;
    assembled.size = msg_len;
    std::ofstream file;
    std::ifstream test;
    char buf1[255] = { 0 };
    char buf2[255] = { 0 };
    char* buf3;
    int sizeSend;
    int sizeDest;
    int k = 0;
    int j = 0;
    string strbuf;
    std::string filepath;
    int hosts = 0;
    assembled.time.tm_mday = NumberDec(message, 0, 1);
    assembled.time.tm_mon = NumberDec(message, 2, 3);
    assembled.time.tm_year = NumberDec4(message, 4, 7);
```

```
    assembled.time.tm_hour = NumberDec(message, 8, 9);
    assembled.time.tm_min = NumberDec(message, 10, 11);
    assembled.time.tm_sec = NumberDec(message, 12, 13);
    assembled.commandL = NumberDec(message, 14, 15);
    switch (assembled.commandL)
    {
    case HISTORY:
    case TEXTMES:
        for (int i = 0; i < 10; i++) {
            sizeSend = i;
            if (message[16 + i] == '/') {
                break;
            }
            buf1[i] = message[16 + i];

        }
        sizeSend++;
        assembled.senderName = std::string(buf1);
        for (int i = 0; i < 10; i++) {
            sizeDest = i;
            if (message[16 + sizeSend + i] == '/') {
                break;
            }
            buf2[i] = message[16 + sizeSend + i];
        }
        assembled.destName = std::string(buf2);
        sizeDest++;

        buf3 = new char[msg_len - (16 + sizeSend + sizeDest)];
        for (int i = 0; i < msg_len - (16 + sizeSend + sizeDest);
i++) {
            buf3[i] = 0;
        }
        for (j = 0;16 + sizeSend + sizeDest + j < msg_len;j++) {
            if (message[16 + sizeSend + sizeDest + j] == '/') {
                break;
            }
            buf3[j] = message[16 + sizeSend + sizeDest + j];
        }

        assembled.mail = std::string(buf3);
        break;
    case USERS_ONLINE:
        hosts = NumberDec(message, 16, 17);
        Globals::Mutex2.lock();
        Globals::OnlineList.clear();
        for (int i = 0; i < hosts; i++) {
            memset(&buf1[0], 0, sizeof(buf1));
```

```
            strbuf.clear();
            for (int j = 0;message[k + 18] != '/';j++) {
                buf1[j] = message[18 + k];

                k++;
            }
            k++;

            strbuf.assign(buf1);

            std::cerr << "[Error] from client.cpp lock" << endl;
            Globals::OnlineList.push_back(strbuf);

        }
        Globals::Mutex2.unlock();
        break;
    case USERS_OFLINE:
        hosts = NumberDec(message, 16, 17);
        Globals::Mutex2.lock();
        Globals::UserList.clear();
        for (int i = 0; i < hosts; i++) {
            memset(&buf1[0], 0, sizeof(buf1));
            strbuf.clear();
            for (j = 0;message[k + 18] != '/';j++) {
                buf1[j] = message[18 + k];
                k++;
            }
            k++;

            strbuf.assign(buf1);
            std::cerr << "[Error] from client.cpp lock" << endl;
            Globals::UserList.push_back(strbuf);

        }
        Globals::Mutex2.unlock();
        break;
    case LOG_IN_ACK:
        buf1[0] = message[16];
        assembled.result = atoi(buf1);
        break;
    case FILE:
        buf3 = new char[1024];
        for (int i = 0; i < 10; i++) {
            sizeSend = i;
            if (message[16 + i] == '/') {
                break;
            }
            buf1[i] = message[16 + i];
```

```
}
sizeSend++;
assembled.senderName = std::string(buf1);
for (int i = 0; i < 10; i++) {
    sizeDest = i;
    if (message[16 + sizeSend + i] == '/') {
        break;
    }
    buf2[i] = message[16 + sizeSend + i];
}
assembled.destName = std::string(buf2);
sizeDest++;
j = sizeDest + sizeSend;
memset(&buf1[0], 0, sizeof(buf1));
for (int i = 0; i < 60; i++) {
    sizeSend = i;
    if (message[16 + i + j] == '/') {
        break;
    }
    buf1[i] = message[16 + j+ i];

}
assembled.fileName = std::string(buf1);
sizeSend++;
j += sizeSend;
j += 16;
memset(&buf1[0], 0, sizeof(buf1));
for (int i = 0; i < assembled.size-j; i++) {
     buf1[i]= message[i+j];
}

test.open(assembled.fileName);
if (test.is_open()) {
    test.close();
    file.open(assembled.fileName, ios::binary | ios::app);
}
else {
    file.open(assembled.fileName, ios::binary | ios::out);
}
memset(&buf3[0], 0, sizeof(buf3));
recv(connection_socket_,buf3 , stoi(string(buf1)), NULL);
for (int i = 0; i < stoi(string(buf1)); i++) {
    file << buf3[i];
}


file.close();
```

```
        break;
    case FORWARD:
    {
        // --- senderName ---
        for (int i = 0; i < 10; i++) {
            sizeSend = i;
            if (message[16 + i] == '/') break;
            buf1[i] = message[16 + i];
        }
        sizeSend++;
        assembled.senderName = std::string(buf1);

        // --- originalSenderName ---
        char bufOrig[255] = { 0 };
        int sizeOrig = 0;
        for (int i = 0; i < 255; i++) {
            sizeOrig = i;
            if (message[16 + sizeSend + i] == '/') break;
            bufOrig[i] = message[16 + sizeSend + i];
        }
        sizeOrig++;
        assembled.originalSenderName = std::string(bufOrig);

        // --- destName ---
        for (int i = 0; i < 10; i++) {
            sizeDest = i;
            if (message[16 + sizeSend + sizeOrig + i] == '/') break;
            buf2[i] = message[16 + sizeSend + sizeOrig + i];
        }
        assembled.destName = std::string(buf2);
        sizeDest++;

        // --- mail ---
        int offset = 16 + sizeSend + sizeOrig + sizeDest;
        buf3 = new char[msg_len - offset + 1];
        memset(buf3, 0, msg_len - offset + 1);
        for (j = 0; offset + j < msg_len; j++) {
            if (message[offset + j] == '/') break;
            buf3[j] = message[offset + j];
        }
        assembled.mail = std::string(buf3);
        delete[] buf3;
        break;
    }
    case AICHAT:
    {
        // senderName (role) / mail /
        for (int i = 0; i < 255; i++) {
```

```
            sizeSend = i;
            if (message[16 + i] == '/') break;
            buf1[i] = message[16 + i];
        }
        sizeSend++;
        assembled.senderName = std::string(buf1);
        buf3 = new char[msg_len - (16 + sizeSend) + 1];
        memset(buf3, 0, msg_len - (16 + sizeSend) + 1);
        for (j = 0; 16 + sizeSend + j < msg_len; j++) {
            if (message[16 + sizeSend + j] == '/') break;
            buf3[j] = message[16 + sizeSend + j];
        }
        assembled.mail = std::string(buf3);
        delete[] buf3;
        break;
    }
    default:
        cout << "undefined protocol" << endl;
        break;
    }
    return assembled;
}
int Client::NumberDec(const char* mes, int start, int end) {
    char* buf = new char[2];
    int j = 0;
    for (int i = start; i < end+1; i++) {
        buf[j] = mes[i];
        j++;
    }
    return stoi(buf);
}
int Client::NumberDec4(const char* mes, int start, int end) {

    char buf[4] = { 0 };
    int j = 0;
    for (int i = start; i < end + 1; i++) {
        buf[j] = mes[i];
        j++;
    }
    return stoi(buf);
}
char* Client::ProtocolComposition(Protocol::Message message)
{

    mytime = time(NULL);
    message.time = *localtime(&mytime);
    int mes_len;
    char* mes;
```

```
    switch (message.commandL)
    {
    case HISTORY:
    case TEXTMES:
        mes_len = 50 + message.senderName.size() +
message.destName.size() + message.mail.size();
        mes = new char[mes_len];
        cout << message.mail << endl;
        sprintf_s(mes, mes_len, "%2d%2d%4d%2d%2d%2d%2d%s/%s/%s/",
message.time.tm_mday, message.time.tm_mon, message.time.tm_year
,message.time.tm_hour, message.time.tm_min, message.time.tm_sec,
message.commandL, message.senderName.c_str(),
message.destName.c_str(), message.mail.c_str());
        cout << mes << endl;
        break;
    case LOG_IN:
        mes_len = 50 + message.authData.name.size() +
message.authData.password.size();
        mes = new char[mes_len];
        sprintf_s(mes, mes_len, "%2d%2d%4d%2d%2d%2d%2d%s/%s/",
message.time.tm_mday, message.time.tm_mon, message.time.tm_year
,message.time.tm_hour, message.time.tm_min, message.time.tm_sec,
message.commandL, message.authData.name.c_str(),
message.authData.password.c_str());
        break;
    case REGISTER:
        mes_len = 50 + message.authData.name.size() +
message.authData.password.size();
        mes = new char[mes_len];
        sprintf_s(mes, mes_len, "%2d%2d%4d%2d%2d%2d%2d%s/%s/",
message.time.tm_mday, message.time.tm_mon , message.time.tm_year
,message.time.tm_hour, message.time.tm_min, message.time.tm_sec,
message.commandL, message.authData.name.c_str(),
message.authData.password.c_str());
        break;
    case FILE:
        mes_len = 50 + message.senderName.size() +
message.destName.size() + message.fileName.size() +
message.mail.size();
        mes = new char[mes_len];
        sprintf_s(mes, mes_len, "%2d%2d%4d%2d%2d%2d%2d%s/%s/%s/%s",
message.time.tm_mday, message.time.tm_mon ,
message.time.tm_year,message.time.tm_hour, message.time.tm_min,
message.time.tm_sec, message.commandL, message.senderName.c_str(),
message.destName.c_str(),message.fileName.c_str(),message.mail.c_str
());
        break;
    case FORWARD:
```

```
        mes_len = 50 + message.senderName.size()
                     + message.originalSenderName.size()
                     + message.destName.size()
                     + message.mail.size()
                     + 10; // Увеличиваем запас для разделителей
        mes = new char[mes_len];

        // Debug: выводим размеры для проверки
        cout << "[DEBUG FORWARD] mes_len=" << mes_len
             << " sender=" << message.senderName.size()
             << " orig=" << message.originalSenderName.size()
             << " dest=" << message.destName.size()
             << " mail=" << message.mail.size() << endl;

        sprintf_s(mes, mes_len,
            "%2d%2d%4d%2d%2d%2d%2d%s/%s/%s/%s/",
            message.time.tm_mday, message.time.tm_mon,
message.time.tm_year,
            message.time.tm_hour, message.time.tm_min,
message.time.tm_sec,
            message.commandL,
            message.senderName.c_str(),
            message.originalSenderName.c_str(),
            message.destName.c_str(),
            message.mail.c_str());
        break;
    case DELETE_MSG:
        mes_len = 50 + message.senderName.size()
                     + message.destName.size()
                     + message.mail.size();
        mes = new char[mes_len];
        sprintf_s(mes, mes_len,
            "%2d%2d%4d%2d%2d%2d%2d%s/%s/%s/",
            message.time.tm_mday, message.time.tm_mon,
message.time.tm_year,
            message.time.tm_hour, message.time.tm_min,
message.time.tm_sec,
            message.commandL,
            message.senderName.c_str(),
            message.destName.c_str(),
            message.mail.c_str());
        break;
    case AICHAT:
        mes_len = 50 + message.senderName.size() +
message.mail.size();
        mes = new char[mes_len];
        memset(mes, 0, mes_len);
        sprintf_s(mes, mes_len,
```

```
            "%2d%2d%4d%2d%2d%2d%2d%s/%s/",
            message.time.tm_mday, message.time.tm_mon,
message.time.tm_year,
            message.time.tm_hour, message.time.tm_min,
message.time.tm_sec,
            message.commandL,
            message.senderName.c_str(),
            message.mail.c_str());
        break;
    default:
        break;
    }
    return mes;


}
// @return `-1` on error, valid socket on success....

SOCKET Client::CreateConnectionSocket(addrinfo* conn_addr) {
    if (!conn_addr) {
        std::cerr << "[Error] CreateConnectionSocket(): conn_addr is
NULL."s << std::endl;
        return -1;
    }

    std::cerr << "[Debug] Creating new connection socket.\n"s;
    SOCKET new_conn_socket = socket(conn_addr->ai_family, conn_addr-
>ai_socktype, conn_addr->ai_protocol);
    if (!IS_VALID_SOCKET(new_conn_socket)) {
        std::cerr << "[Error] Failed to create a new connection
socket: socket(): "s <<WSAGetLastError()<< std::endl;
        return -1;
    }
    return new_conn_socket;
}
int Client::SendMessage(const std::string& message) {
    std::string assembled_msg(message);
    PrependMessageLength(assembled_msg);
    int total_bytes = assembled_msg.size();
    int send_bytes = 0;
    int send_n;
    std::cerr << "[Debug] Sending message: '"s << assembled_msg <<
"'\n"s;
    while (total_bytes > send_bytes) {
        send_n = send(connection_socket_, assembled_msg.data() +
send_bytes, total_bytes - send_bytes, 0);
        if (send_n == -1) {
```

```
            std::cerr << "[Error] Failed to send data to the remote
host: send(): "s <<
std::system_category().message(GET_SOCKET_ERRNO()) << std::endl;
            return send_n;
        }
        send_bytes += send_n;
        std::cerr << "[Debug] Sending "s << send_n << " bytes to the
remote host\n"s;
    }


    return send_bytes;
}

int Client::ReceiveMessage(char* writable_buff) {

    // Receive packet length (first 4 bytes)
    char msg_len_str[5]; // 4 bytes + 1 byte for the null-
terminating character
    memset(msg_len_str, 0x00, sizeof(msg_len_str));
    msg_len_str[4] = '\0';

    int recv_bytes = recv(connection_socket_, msg_len_str,
sizeof(msg_len_str) - 1, 0);
    if (recv_bytes <= 0) { // either client disconnect or an error
        return recv_bytes;
    }
    std::cerr << "[Debug] Received "s << recv_bytes << " bytes
(packet length): '"s << msg_len_str << "\n"s;
    // Check if the message conforms to the protocol
    for (const char c : std::string(msg_len_str)) {
        if (!std::isdigit(c)) {
            std::cerr << "[Error] Failed to read data from the
remote host: invalid protocol format.\n"s;
            return -1;
        }
    }

    int packet_length = std::atoi(msg_len_str);
    recv_bytes = recv(connection_socket_, writable_buff,
packet_length, 0);
    if (recv_bytes <= 0) { // Check for errors
        return recv_bytes;
    }
    std::cerr << "[Debug] Received "s << recv_bytes << " bytes
(actual packet)\n"s << writable_buff << "\n"s;
    Protocol::Message regular =
ProtocolDecomposition(writable_buff,packet_length);
```

```
    switch (regular.commandL) {
    case HISTORY:
    case TEXTMES:
            std::cerr << "[Debug] ["s << regular.time.tm_hour << ":"
<< regular.time.tm_min << "] " << regular.mail << "\n"s;
            Globals::Mutex1.lock();
            Globals::assembleLine.push(regular);
            Globals::Mutex1.unlock();
            break;
    case USERS_ONLINE:
            std::cerr << "[Debug] ["s << regular.time.tm_hour << ":"
<< regular.time.tm_min << "] ";
            Globals::Mutex2.lock();

            for (string i : Globals::OnlineList) {
                std::cerr << i;
            }
            Globals::Mutex2.unlock();
            std::cerr << "\n"s;
            break;
    case LOG_IN_ACK:
        Globals::result = regular.result;
        std::cerr << "[Client] LOG_IN_ACK received, result = " <<
regular.result << endl;
        cout << "[Client] Globals::result set to " <<
Globals::result << endl;
        break;
    case FORWARD:
        std::cerr << "[Client FORWARD received] time: " <<
regular.time.tm_hour << ":" << regular.time.tm_min
                  << " from: " << regular.senderName
                  << " orig: " << regular.originalSenderName
                  << " mail: " << regular.mail << std::endl;
        Globals::Mutex1.lock();
        Globals::assembleLine.push(regular);
        Globals::Mutex1.unlock();
        break;
    case AICHAT:
        Globals::Mutex1.lock();
        Globals::aiLine.push(regular);
        Globals::Mutex1.unlock();
        break;
    case FILE:
    {
        // Receive raw file bytes after the header packet
        int fileSize = std::stoi(regular.mail);
        std::string savePath = std::string(getenv("USERPROFILE")) +
"\\Downloads\\" + regular.fileName;
```

```
        std::ofstream outFile(savePath, std::ios::binary);
        if (outFile.is_open()) {
            int received = 0;
            char fileBuf[CHUNK_SIZE + 1];
            while (received < fileSize) {
                int toRead = min(CHUNK_SIZE, fileSize - received);
                memset(fileBuf, 0, sizeof(fileBuf));
                int r = recv(connection_socket_, fileBuf, toRead,
0);
                if (r <= 0) break;
                outFile.write(fileBuf, r);
                received += r;
            }
            outFile.close();
            std::cerr << "[Client] File saved: " << savePath <<
std::endl;
            // Notify UI
            Protocol::Message notify;
            notify.commandL = TEXTMES;
            notify.senderName = regular.senderName;
            notify.destName = regular.destName;
            notify.mail = "[File received: " + regular.fileName +
"]";
            time_t now = time(NULL);
            localtime_s(&notify.time, &now);
            Globals::Mutex1.lock();
            Globals::assembleLine.push(notify);
            Globals::Mutex1.unlock();
        }
        break;
    }
    default:

        break;
    }

    return recv_bytes;
}


int Client::HandleConnection()  {

    while (true) {
        char msg_buff[MAX_DATA_BUFFER_SIZE];
        memset(msg_buff, 0x00, sizeof(msg_buff));
        memset(msg_buff, 0x00, sizeof(msg_buff));

        int recv_bytes = ReceiveMessage(msg_buff);
```

```
        if (recv_bytes <= 0) {
            if (recv_bytes == 0) {
                std::cerr << "[Info] Remote host has closed the
connection."s << std::endl;
                std::exit(1);
            }
            else {
                std::cerr << "[Error] Failed to receive data from
the remote host: recv(): "s <<
std::system_category().message(GET_SOCKET_ERRNO()) << std::endl;
                std::exit(1);
            }
        }
        std::cout << msg_buff << '\n';


    }
}
```

Файл Globals.cpp

```
#include "Common.h"
std::mutex Globals::Mutex1;
std::mutex Globals::Mutex2;
std::queue<Protocol::Message> Globals::assembleLine;
std::queue<Protocol::Message> Globals::aiLine;
std::vector<string> Globals::OnlineList;
std::vector<string> Globals::UserList;
std::string Globals::user="";
std::string Globals::dest = "";
int Globals::result = 3;
bool ReadConnectionInfo(const std::string& filename, std::string&
ip, std::string& port) {
    std::ifstream file(filename);
    if (!file.is_open())
    {
        std::cerr << "Error: Unable to open file " << filename <<
std::endl;
        return false;
    }

    std::string line;
    while (std::getline(file, line))
    {
        std::istringstream iss(line);
        std::string key, value;
        if (std::getline(iss, key, '=') && std::getline(iss, value))
        {
```

```
            if (key == "ip")
            {
                ip = value;
            }
            else if (key == "port")
            {
                port = value;
            }
        }
    }

    file.close();
    return true;
}
```

Файл AIChatForm.h

```
#pragma once
#include "Common.h"
#include "Client.h"
extern Client clie;
namespace Vis {
    using namespace System;
    using namespace System::Windows::Forms;
    using namespace System::Runtime::InteropServices;
    using namespace std;

    public ref class AIChatForm : public Form {
    public:
        TextBox^ historyBox;
        TextBox^ inputBox;
        Button^  sendBtn;
        System::Windows::Forms::Timer^ pollTimer;
        bool isRussian;

        AIChatForm() {
            isRussian = false;
            this->Text = L"AI Chat";
            this->Size = System::Drawing::Size(600, 500);
            this->StartPosition = FormStartPosition::CenterParent;
            this->BackColor = System::Drawing::Color::BurlyWood;

            historyBox = gcnew TextBox();
            historyBox->Multiline = true;
            historyBox->ReadOnly = true;
            historyBox->ScrollBars = ScrollBars::Vertical;
            historyBox->Location = System::Drawing::Point(10, 10);
            historyBox->Size = System::Drawing::Size(565, 380);
```

```
            historyBox->BackColor = System::Drawing::Color::White;

            inputBox = gcnew TextBox();
            inputBox->Location = System::Drawing::Point(10, 400);
            inputBox->Size = System::Drawing::Size(460, 23);

            sendBtn = gcnew Button();
            sendBtn->Text = L"Send";
            sendBtn->Location = System::Drawing::Point(480, 400);
            sendBtn->Size = System::Drawing::Size(95, 23);
            sendBtn->Click += gcnew System::EventHandler(this,
&AIChatForm::sendBtn_Click);

            pollTimer = gcnew System::Windows::Forms::Timer();
            pollTimer->Interval = 100;
            pollTimer->Tick += gcnew System::EventHandler(this,
&AIChatForm::pollTimer_Tick);

            this->Controls->Add(historyBox);
            this->Controls->Add(inputBox);
            this->Controls->Add(sendBtn);

            this->Load += gcnew System::EventHandler(this,
&AIChatForm::AIChatForm_Load);
            this->FormClosing += gcnew
System::Windows::Forms::FormClosingEventHandler(this,
&AIChatForm::AIChatForm_Closing);
        }

        void ApplyLanguage() {
            if (isRussian) {
                this->Text = gcnew String("\xC8\xC8 \xD7\xE0\xF2");
                sendBtn->Text = gcnew
String("\xCE\xF2\xEF\xF0\xE0\xE2\xE8\xF2\xFC");
            } else {
                this->Text = L"AI Chat";
                sendBtn->Text = L"Send";
            }
        }

    private:
        void AIChatForm_Load(System::Object^ sender,
System::EventArgs^ e) {
            pollTimer->Start();
            // Request history
            Protocol::Message req;
            req.commandL = AICHAT;
            req.senderName = Globals::user;
```

```
            req.mail = "__HISTORY__";
            time_t now = time(NULL);
            req.time = *localtime(&now);
            clie.SendMessage(clie.ProtocolComposition(req));
        }

        void AIChatForm_Closing(System::Object^ sender,
System::Windows::Forms::FormClosingEventArgs^ e) {
            pollTimer->Stop();
            // Drain aiLine on close
            Globals::Mutex1.lock();
            while (!Globals::aiLine.empty()) Globals::aiLine.pop();
            Globals::Mutex1.unlock();
        }

        void pollTimer_Tick(System::Object^ sender,
System::EventArgs^ e) {
            Globals::Mutex1.lock();
            bool gotAny = false;
            while (!Globals::aiLine.empty()) {
                Protocol::Message m = Globals::aiLine.front();
                Globals::aiLine.pop();
                std::string prefix = (m.senderName == "user") ?
"You: " : "AI: ";
                historyBox->AppendText(gcnew String((prefix + m.mail
+ "\r\n").c_str()));
                gotAny = true;
            }
            Globals::Mutex1.unlock();
            if (gotAny) {
                sendBtn->Enabled = true;
            }
        }

        void sendBtn_Click(System::Object^ sender,
System::EventArgs^ e) {
            if (String::IsNullOrWhiteSpace(inputBox->Text)) return;
            std::string userMsg =
string((char*)(void*)Marshal::StringToHGlobalAnsi(inputBox->Text));

            historyBox->AppendText("You: " + inputBox->Text +
"\r\n");
            inputBox->Clear();
            sendBtn->Enabled = false;

            Protocol::Message msg;
            msg.commandL = AICHAT;
            msg.senderName = Globals::user;
```

```
            msg.mail = userMsg;
            time_t t = time(NULL);
            msg.time = *localtime(&t);
            clie.SendMessage(clie.ProtocolComposition(msg));
        }
    };
}
```

Приложение В

(обязательное)

Распределенная интеллектуальная система

Презентация

Распределенная интеллектуальная система

Автор работы:

студент гр. 22ВВП2 Агапов Илья Евгеньевич

Научный руководитель:

д.т.н., профессор каф. ВТ Зинкин Сергей Александрович

Пенза, 2026

Цель работы и объекты

Цель работы – разработка клиент-серверной распределенной интеллектуальной системы под управлением ОС Windows с функциями обмена и пересылками сообщениями, передачи файлов и интеграции AI ассистента.

Объекты работы – консольный клиент, графический клиент, сервер, база данных, выполняющие следующие функции:

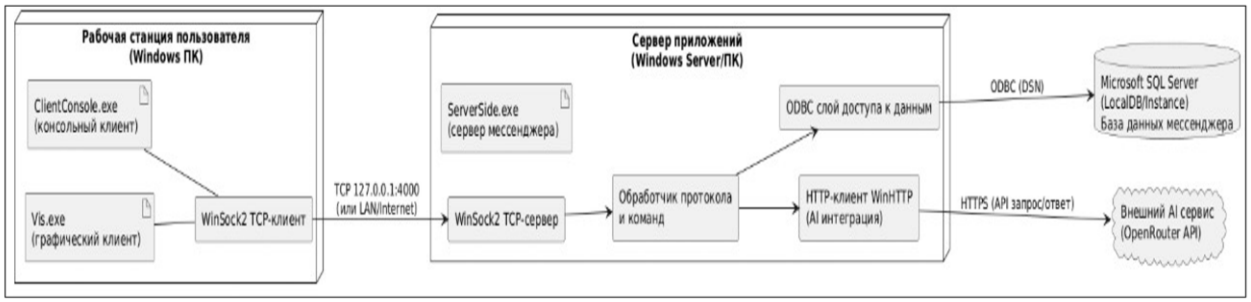
- Регистрацию и авторизацию пользователей
- Отправку и получение текстовых сообщений
- Передачу файлов
- работу AI-чата через внешний API
- Сохранение истории переписки

Сравнение аналогов

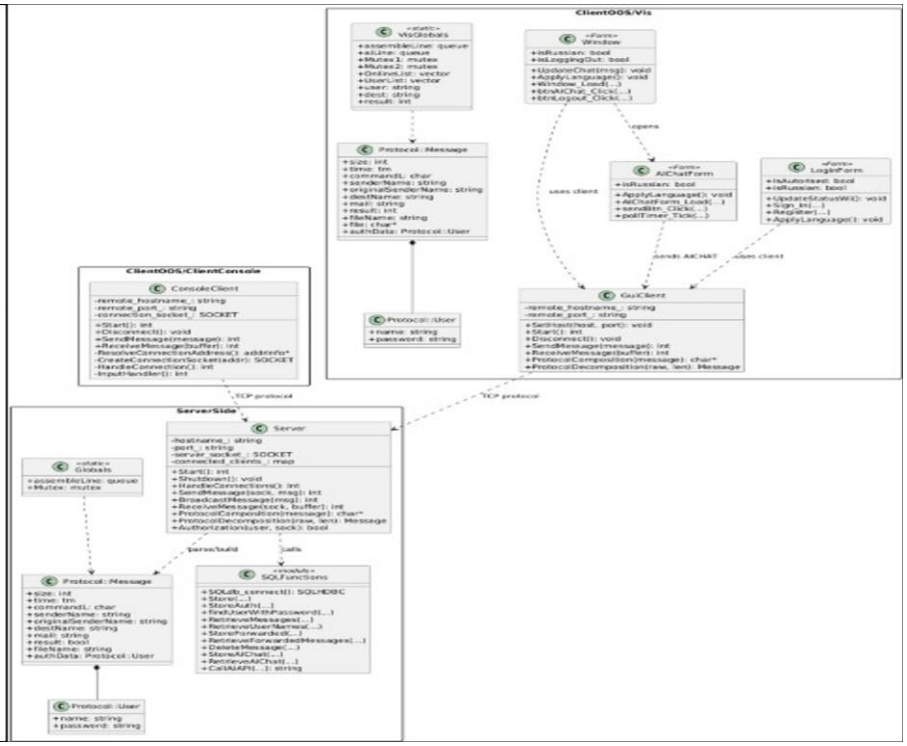
Название	преимущества	недостатки
Telegram	– высокая скорость доставки сообщений; – мульти-платформенность; – удобный интерфейс.	– отсутствие возможности развертывания на собственных серверах, что создает риски утечки коммерческой тайны; – не работает в Российской Федерации.
Вконтакте(VK)	– обширная пользовательская база в России и странах СНГ; – глубокая интеграция с экосистемой VK;	– серверная инфраструктура также расположена вне периметра организации;

Используемые технологии

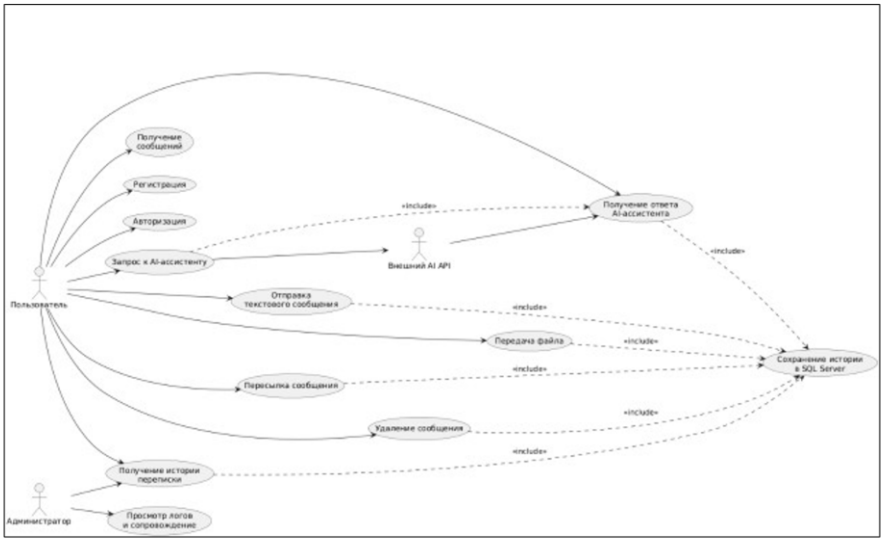
Структура приложения



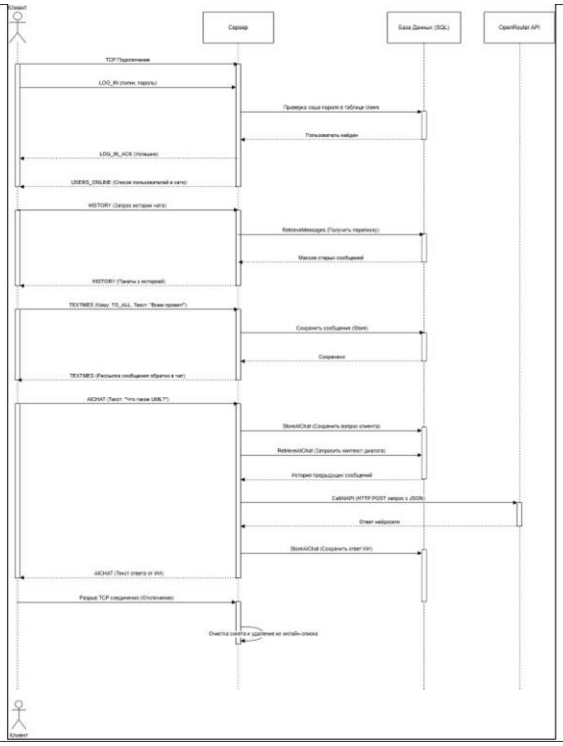
UML-диаграмма классов



UML-диаграмма вариантов использования



UML-диаграмма последовательности



Сущности БД

Users

Поле	Тип данных	Описание
ID	Int	Идентификатор пользователя
username	Varchar	Логин пользователя
password	Varchar	Пароль пользователя

Сущности БД

Message

Поле	Тип данных	Описание
ID	Int	Идентификатор сообщения
size	Int	Размер сообщения
time	Varchar	Время отправки
commandl	Char	Тип команды
senderName	Varchar	Имя отправителя
destName	Varchar	Имя получателя
mail	Varchar	Текст сообщения

Сущности БД

ForwardedMessage

Поле	Тип данных	Описание
ID	Int	Идентификатор сообщения
size	Int	Размер сообщения
time	Varchar	Время отправки
commandl	Char	Тип команды
senderName	Varchar	Имя отправителя
originalSenderName	Varchar	Имя оригинального отправителя
destName	Varchar	Имя получателя
mail	Varchar	Текст сообщения

Сущности БД

AIChat

Поле	Тип данных	Описание
ID	Int	Идентификатор пользователя
username	Varchar	Имя пользователя
role	Varchar	Роль сообщения
message	Varchar	Текст сообщения
time	Datetime	Дата отправки

Заключение

В ходе работы была разработана распределенная интеллектуальная система под ОС Windows, включающая сервер, консольный и графический клиент. Реализованы ключевые функции: авторизация, обмен сообщениями, передача файлов, хранение истории и ИИ-чат.

Поставленная цель достигнута: создана работоспособная система, демонстрирующая полный цикл взаимодействия клиента и сервера. Разработанное решение может использоваться как основа для дальнейшего развития и внедрения дополнительных механизмов безопасности, масштабирования и мониторинга.

Спасибо за внимание