## Презентация

Студент: Грачёв В.А.

Курс: Разработчик С++. Специалист

Чат сервис по наполнению базы данных через клиентские сообщения

В современном мире разработка программного обеспечения требует глубоких знаний не только в языках программирования, но и в таких смежных областях, как базы данных, сетевые технологии, виртуализация и контейнеризация. Современные серверные приложения, особенно в условиях растущей популярности микросервисных архитектур и распределённых систем, предъявляют высокие требования к производительности, надёжности и масштабируемости.

Язык **C++** является одним из наиболее мощных и гибких инструментов для создания высокопроизводительных решений. Его широкое применение в системном и серверном программировании обусловлено высокой производительностью, а также возможностью управления низкоуровневыми механизмами операционных систем.

C++ выступает ключевым инструментом при разработке различных критически важных серверных решений, включая:

* **Банковские системы** — такие как платформы, использующие C++ для обработки транзакций с высокой скоростью и надёжностью (например, **JPMorgan Chase**).
* **Распределённые микросервисы** — в таких крупных системах, как **Netflix** и **Spotify**, которые требуют низких задержек и высокой пропускной способности.
* **Системы мониторинга IoT** — где важен анализ большого объёма данных в реальном времени.
* **Чат-серверы** — для обмена сообщениями, где поддерживается большое количество подключений одновременно за счёт асинхронных возможностей C++.

**Актуальность темы** заключается в том, что такие системы требуют высокой эффективности обработки данных и взаимодействия с клиентами через сеть, а также лёгкости в развёртывании в разных окружениях. Технологии контейнеризации, такие как **Docker**, в сочетании с возможностями C++, позволяют создавать изолированные и легко масштабируемые решения, что особенно важно при работе в облачных платформах, таких как **Amazon Web Services (AWS)** и **Google Cloud Platform (GCP)**.

**Цель данной работы** — разработка серверного приложения на языке C++, которое включает работу с базой данных для хранения информации, использование сетевых протоколов для обмена сообщениями между клиентами, а также развёртывание с помощью контейнеризации для удобного управления и масштабирования.

**Задачи дипломной работы**:

1. Изучить и применить основы работы с базами данных и SQL в связке с C++.
2. Описать и реализовать сетевые взаимодействия через клиент-серверное приложение с использованием сокетов.
3. Использовать инструменты виртуализации и контейнеризации для развёртывания и масштабирования приложения.
4. Применить продвинутые техники программирования на C++, включая указатели, шаблоны и умные указатели.
5. Создать проект с использованием CMake, библиотек и Docker для демонстрации взаимодействия всех компонентов системы.

Для взаимодействия с РБД используется язык SQL (Structured Query Language), который является стандартом де-факто для управления данными, обеспечивая операции извлечения, добавления, изменения и удаления данных.

Важно отметить, что каждое из решений имеет свои сильные и слабые стороны:

* **SQLite** лучше всего подходит для небольших приложений и разработки, благодаря своей простоте и отсутствию необходимости в отдельном сервере.
* **MySQL** часто используется для веб-приложений и поддерживает масштабируемость, что делает его популярным выбором для больших систем.
* **PostgreSQL** предпочтителен для сложных запросов и данных с высокой согласованностью, предоставляя расширенные возможности работы с данными.

В рамках дипломного проекта можно реализовать клиент-серверное приложение, где сервер на C++ принимает данные от клиентов, обрабатывает их и сохраняет в базу данных. Клиенты подключаются к серверу, отправляют запросы и получают ответы. Например, можно использовать протокол HTTP или WebSocket для более удобного взаимодействия. Также важно рассмотреть использование SSL/TLS для шифрования данных, передаваемых между клиентом и сервером, подчеркивая важность безопасности в сетевых приложениях.

Конкретный пример взаимодействия: сервер может принимать сообщения в формате JSON, которые содержат данные пользователя, такие как имя и сообщение. Сервер обрабатывает эти данные и отправляет ответ в формате JSON. Это позволяет легко масштабировать систему и добавлять новые функции.

Это можно расширить до полноценного многопользовательского приложения, например, чат-системы или приложения для обмена данными.

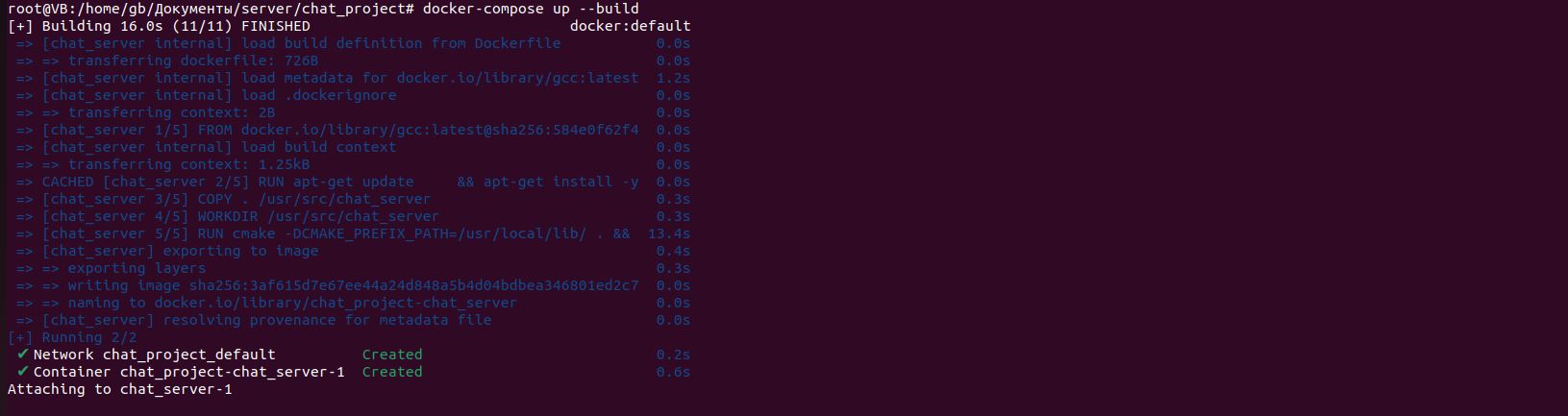
Контейнеризация — это технология, позволяющая упаковывать приложение и его зависимости в единый контейнер, который может запускаться на любой системе с поддержкой контейнеров. Это упрощает развертывание приложений, гарантирует одинаковое окружение на разных системах и минимизирует проблемы с зависимостями.

Контейнеризация обеспечивает:

* Изоляцию приложений.
* Повторяемость среды разработки и эксплуатации.
* Удобное масштабирование и управление приложениями в облачной среде.

Наиболее популярной технологией контейнеризации является **Docker**.

Контейнеризация с помощью Docker позволяет упростить развертывание чат-сервера и базы данных, обеспечивая стабильную и предсказуемую среду выполнения. Это также облегчает управление зависимостями и настройками, что делает процесс разработки и развертывания более эффективным.



Контейнеризация с помощью Docker стала важным стандартом в современной разработке, позволяя эффективно управлять приложениями на различных этапах их жизненного цикла. Использование Docker в дипломных проектах может продемонстрировать понимание современных технологий разработки и навыки работы с контейнерной инфраструктурой.

## Описание проекта

Данный проект направлен на создание чат-сервера, предоставляющего возможность пользователям зарегистрироваться, войти в систему и обмениваться сообщениями с сервером с дальнейшим добавлением в базу данных в режиме реального времени. Основной язык разработки — C++, который был выбран за высокую производительность, контроль над системными ресурсами и широкие возможности асинхронного сетевого взаимодействия с использованием библиотеки **Boost.Asio**.

### Задачи проекта:

1. **Регистрация и аутентификация пользователей**: Пользовательские данные, включая информацию для аутентификации (логин, пароль), сохраняются в базе данных.
2. **Обмен сообщениями**: После успешной аутентификации пользователи могут обмениваться сообщениями с сервером в режиме реального времени.
3. **Хранение данных**: Информация о пользователях, а также история сообщений сохраняются в базе данных для последующего доступа.
4. **Контейнеризация**: Приложение развёрнуто в контейнерах с использованием **Docker**, что обеспечивает его переносимость и удобство в управлении зависимостями.

### Причины выбора C++:

* **Высокая производительность**: Возможность управления системными ресурсами на низком уровне.
* **Асинхронность**: С помощью **Boost.Asio** можно организовать высокоэффективную работу с сетевыми запросами.
* **Модульность**: Проект демонстрирует применение нескольких ключевых тем курса, включая работу с базами данных, сетевые взаимодействия, операционные системы и контейнеризацию.

### Базы данных:

* **SQLite**: Подходит для небольших приложений с минимальной нагрузкой и упрощенным развертыванием.
* **MySQL/PostgreSQL**: Эти СУБД используются для проектов, где требуется поддержка больших объёмов данных и высокой производительности. В проекте можно выбрать СУБД в зависимости от предполагаемой нагрузки.

### Цель проекта:

Главная цель проекта — продемонстрировать применение теоретических знаний на практике, включая работу с базами данных, сетевыми взаимодействиями, операционными системами и контейнеризацией через **Docker**.

## Требования к проекту

Для успешной реализации проекта необходимо соблюдение следующих требований:

* **Язык программирования**: C++
  + Использование C++ обусловлено необходимостью эффективного управления системными ресурсами, а также возможностью работы с многопоточностью и асинхронными операциями через библиотеку Boost.
* **База данных**:
  + **SQLite** для локальных и небольших приложений с минимальной нагрузкой.
  + **MySQL/PostgreSQL** — для приложений с высокой нагрузкой и требующих масштабирования.
* **Сетевое взаимодействие**: Сокеты (Boost.Asio)
  + **Boost.Asio** обеспечивает асинхронное взаимодействие с сетью и позволяет управлять многопоточными соединениями, что критически важно для обработки множества клиентов в реальном времени.
* **Контейнеризация**: Docker
  + **Docker** используется для упаковки приложения и всех его зависимостей в контейнеры, что упрощает развертывание, переносимость и масштабируемость приложения.

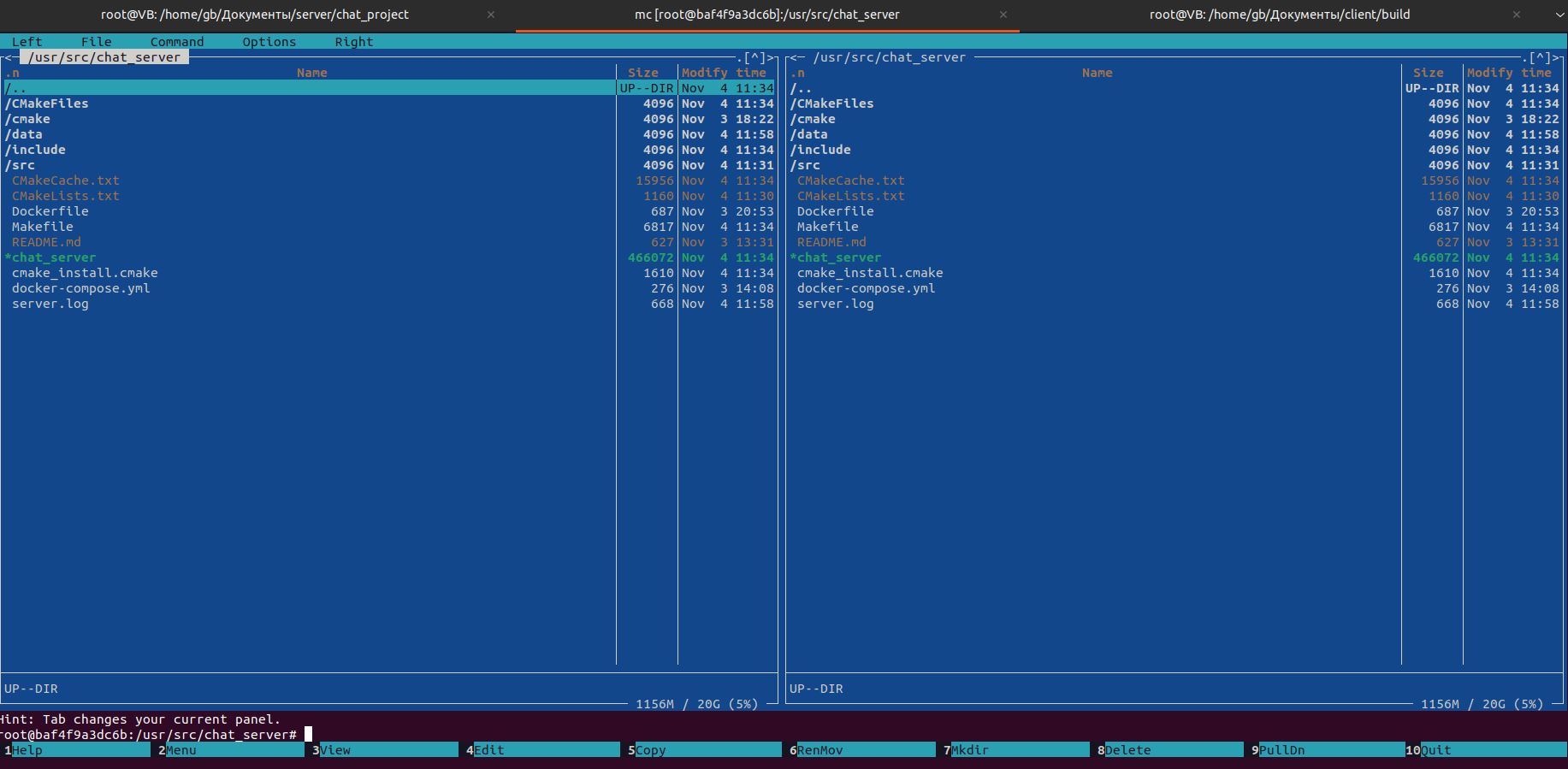
Для хранения информации о пользователях и сообщениях было решено использовать реляционную базу данных **SQLite**. Это решение обусловлено ее легкостью и простотой интеграции с проектом на C++. **SQLite** идеально подходит для приложений с небольшим количеством пользователей, так как не требует отдельного сервера базы данных, а данные хранятся локально.

База данных содержит две таблицы:

* **Users**: хранит информацию о пользователях приложения, включая их имена и пароли для регистрации и авторизации.
  + Поля:
    - id: уникальный идентификатор пользователя.
    - username: имя пользователя.
    - password: пароль пользователя.
* **Messages**: сохраняет сообщения, отправленные пользователями.
  + Поля:
    - id: уникальный идентификатор сообщения.
    - sender\_id: ID отправителя сообщения (внешний ключ на таблицу Users).
    - receiver\_id: ID получателя сообщения (внешний ключ на таблицу Users).
    - message: текст сообщения.
    - timestamp: временная метка отправки сообщения.

### Подробное описание архитектуры приложения

Архитектура чат-сервера реализует клиент-серверную модель, которая разделяет ответственность между клиентами и сервером, обеспечивая эффективное взаимодействие и управление данными. Основные компоненты архитектуры включают клиентскую часть, серверную часть и базу данных.



#### Основные компоненты архитектуры:

1. **Клиентская часть**:
   * **Функции**: Клиенты отправляют и получают сообщения через сетевые сокеты.
   * **Технологии**: Взаимодействие реализовано на C++ с использованием библиотеки Boost.Asio, которая обеспечивает асинхронное взаимодействие с сервером. Это позволяет клиенту выполнять другие задачи, пока ожидает ответа от сервера, что улучшает отзывчивость приложения.
   * **Примеры действий**:
     + Регистрация и аутентификация пользователя.
     + Отправка сообщений другим пользователям.
     + Получение сообщений от сервера в реальном времени.
2. **Серверная часть**:
   * **Функции**: Сервер принимает запросы от клиентов, обрабатывает их и пересылает данные.
   * **Технологии**: Серверная часть написана на C++ с использованием Boost.Asio для управления соединениями и обработки асинхронных запросов.
   * **Архитектура**:
     + Каждый клиент обрабатывается в отдельном потоке для обеспечения многопоточности и одновременной работы с несколькими клиентами.
     + Сервер взаимодействует с базой данных для выполнения операций, таких как сохранение и извлечение данных о пользователях и сообщениях.
3. **База данных**:
   * **Тип**: SQLite (или MySQL/PostgreSQL в зависимости от конфигурации).
   * **Функции**: Хранят информацию о пользователях (имена, пароли) и сообщениях (отправитель, получатель, текст сообщения, временная метка).
   * **Архитектурные решения**: Использование паттерна проектирования **Singleton** для управления подключением к базе данных, что гарантирует, что в приложении будет только один экземпляр подключения к базе данных, что предотвращает возможные проблемы с конкуренцией.

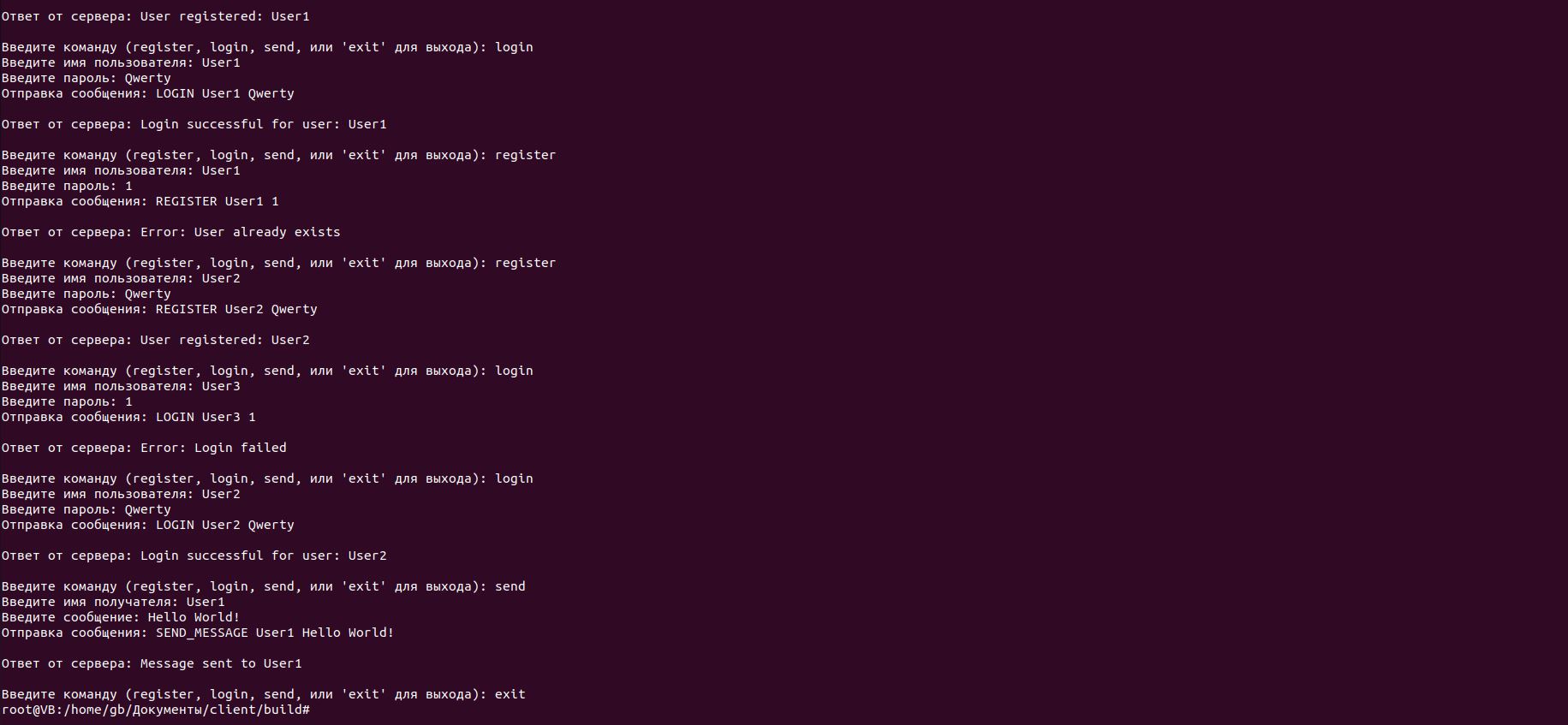
Архитектура приложения ориентирована на модульность, производительность и удобство обслуживания. Разделение на клиентскую и серверную части, а также использование паттернов проектирования, таких как Singleton и MVC, позволяют создать гибкую и расширяемую систему, которая легко адаптируется под новые требования и улучшения. Эта архитектура не только поддерживает текущие функции, но и закладывает основу для будущих улучшений и расширений приложения.

Клиентское приложение является важным компонентом чат-сервера, обеспечивая пользователю интерфейс для отправки и получения сообщений. Оно взаимодействует с сервером через TCP-соединения, используя библиотеку Boost.Asio для управления сетевыми операциями.

Клиентская часть чат-сервера, разработанная с использованием C++ и Boost.Asio, обеспечивает надежное и эффективное взаимодействие с сервером. Это простое приложение может быть расширено для добавления дополнительных функций, таких как аутентификация пользователей, обработка различных типов сообщений и управление несколькими клиентскими сессиями.

*Основные функции клиентского приложения:*

1. **Подключение к серверу**: Клиент устанавливает соединение с сервером по указанному адресу и порту.
2. **Отправка сообщений**: Пользователь может отправить текстовое сообщение на сервер.
3. **Получение сообщений**: Клиент получает ответ от сервера и отображает его пользователю.



Логирование и мониторинг являются критически важными для обеспечения надежности и производительности сервера. Система логирования позволяет отслеживать события и ошибки, в то время как мониторинг предоставляет информацию о работе приложения в реальном времени. Эти аспекты способствуют повышению качества разработки и обеспечивают возможность быстрой реакции на возникающие проблемы.

Тестирование приложения подтвердило его функциональность и надежность. Проведение различных типов тестирования помогло выявить и устранить потенциальные проблемы, а также обеспечило уверенность в том, что чат-сервер готов к развертыванию и использованию.

Обработка ошибок и безопасность являются основополагающими аспектами разработки данного чат-сервера. Реализованные механизмы обеспечивают защиту данных пользователей и повышают надежность приложения. Использование хэширования паролей, параметризованных SQL-запросов и надлежащей обработки ошибок значительно снижает риски, связанные с сетевыми приложениями.

В рамках данного проекта был успешно разработан чат-сервер, который демонстрирует практическое применение знаний в области баз данных, сетевых технологий и контейнеризации. Основные достижения проекта включают:

* **Разработка на C++**: Использование языка C++ позволило добиться высокой производительности и низкого уровня контроля над системными ресурсами, что критически важно для сетевых приложений.
* **Интеграция с базой данных**: Реализация системы управления пользователями и сообщений через реляционную базу данных SQLite (или MySQL/PostgreSQL) продемонстрировала навыки работы с SQL и основами проектирования баз данных.
* **Сетевое взаимодействие**: Применение библиотеки Boost.Asio для создания асинхронного TCP-сервера обеспечило эффективную обработку нескольких клиентских соединений, что повышает отзывчивость и стабильность приложения.
* **Контейнеризация с помощью Docker**: Использование Docker для упаковки приложения в контейнеры упростило процесс развертывания и управления, что делает приложение более переносимым и устойчивым к изменениям в окружении.
* **Обработка ошибок и безопасность**: Внедрение методов аутентификации пользователей и защиты данных позволило повысить безопасность приложения, а также реализовать грамотную обработку ошибок для улучшения пользовательского опыта.
* **Оптимизация производительности**: Примененные методы, такие как асинхронное взаимодействие, транзакции и кэширование, способствовали созданию высокопроизводительного и отзывчивого сервиса.

Данный проект не только демонстрирует теоретические знания, полученные в рамках курса, но и предоставляет практические навыки разработки программного обеспечения, которые могут быть применены в реальных условиях. Будущие улучшения могут включать расширение функциональности приложения, добавление новых возможностей и оптимизацию архитектуры для повышения масштабируемости.

**Заключение**

Разработанное серверное приложение на C++ демонстрирует применение широкого спектра технологий, изученных в рамках курса "Разработчик C++: Специалист". Оно взаимодействует с базой данных, реализует сетевую логику для общения с клиентами и использует контейнеризацию для упрощения развертывания и управления. Проект можно развивать дальше, добавляя поддержку новых функций, таких как шифрование данных, улучшенная авторизация, а также улучшение клиентской части. Проект демонстрирует применение современных технологий для разработки сетевых приложений и может быть легко расширен для использования в более крупных проектах.