ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ
«САНКТ-ПЕТЕРБУРГСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ПЕТРА ВЕЛИКОГО»
Институт компьютерных наук и кибербезопасности
Высшая школа программной инженерии

ИНДИВИДУАЛЬНЫЙ ПЛАН (ЗАДАНИЕ И ГРАФИК) ПРОВЕДЕНИЯ ПРАКТИКИ

Абраамян Александр Манвелович

Направление подготовки (код/наименование) 09.03.04 «Программная инженерия»

Профиль (код/наименование) 09.03.04_01 «Технология разработки и сопровождения качественного программного продукта»

Вид практики: научно-исследовательская работа

Тип практики: распределенная

Место прохождения практики ФГАОУ ВО «СПбПУ», ИКНК, ВШПИ, СПб, ул.Политехническая, 29

Руководитель практической подготовки от ФГАОУ ВО «СПбПУ»:

Коликова Татьяна Всеволодовна, старший преподаватель ВШПИ ИКНК
(Ф.И.О., уч.степень, должность)

Руководитель практической подготовки от профильной организации: -
Петров Александр Владимирович, старший преподаватель ВШПИ ИКНТ

Рабочий график проведения практики

Сроки практики: с 02.09.2024 по 16.12.24

№ п/п	Этапы (периоды) практики	Вид работ	Сроки прохождения этапа (периода) практики
1	Организационный этап	Установочная лекция для разъяснения целей, задач, содержания и порядка прохождения практики, инструктаж по технике безопасности, выдача сопроводительных документов по практике	02.09
2	Основной этап	Разработка мессенджера с функциями голосовой и видеосвязи	03.09-15.12
3	Заключительный этап	Защита отчета по практике	16.12

Обучающийся _____ / Абраамян А. М. /

Руководитель практической подготовки
от ФГАОУ ВО «СПбПУ» _____ / Абраамян А. М. /

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ
«САНКТ-ПЕТЕРБУРГСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ПЕТРА ВЕЛИКОГО»
Институт компьютерных наук и кибербезопасности
Высшая школа программной инженерии

Отчет о прохождении научно-исследовательской работы

Абраамян Александр Манвелович

4 курс, 5130904/10101

09.03.04 «Программная инженерия»

Место прохождения практики: ФГАОУ ВО «СПбПУ», ИКНК, ВШПИ,

СПб, ул.Политехническая, 29

Сроки практики: с 02.09.2024 по 16.12.2024

Руководитель практической подготовки от ФГАОУ ВО «СПбПУ»:

Коликова Татьяна Всеволодовна, старший преподаватель ВШПИ ИКНК

Руководитель практической подготовки от профильной организации: -

Петров Александр Владимирович, старший преподаватель ВШПИ ИКНТ

Оценка: зачтено/не зачетно

Руководитель практической подготовки
от ФГАОУ ВО «СПбПУ»: / Абраамян А. М. /

Руководитель практической подготовки
от профильной организации: -

Обучающийся: / Абраамян А. М. /

Дата: 16.12.2024

Оглавление

Постановка задачи

Разработать мессенджер с поддержкой текстового общения, а также функций голосовой и видеосвязи. Приложение будет ориентировано на обеспечение стабильной связи между пользователями, удобный интерфейс и безопасность данных. Особое внимание будет уделено реализации современных технологий передачи данных в реальном времени и интеграции с пользовательскими устройствами.

Обоснование актуальности работы

С каждым годом использование цифровых сервисов охватывает все более широкий спектр жизненных аспектов: от коммуникации и развлечений до образования и финансовых операций. В основе большинства современных интернет-услуг лежат сложные веб-приложения и мобильные платформы, которые взаимодействуют с пользователями и обрабатывают большие объемы данных. Для того чтобы эти приложения могли эффективно работать, с минимальными задержками и без сбоев, необходимо надежное и масштабируемое серверное решение.

В последние десятилетия значительные изменения в технологической сфере привели к росту объема и сложности данных, которые должны быть обработаны в реальном времени. Увеличение числа пользователей и устройств, подключенных к сети, порождает новые вызовы для бэкенд-разработки: необходимо создавать системы, которые смогут обрабатывать огромные объемы данных без потери производительности, обеспечивать безопасность и конфиденциальность информации, а также сохранять высокую доступность и отказоустойчивость даже при интенсивных нагрузках.

Бэкенд-разработка — ключевой компонент для успешной работы многих сервисов. Не только крупные корпорации, но и стартапы, а также малые и средние предприятия сталкиваются с необходимостью создания высококачественных серверных решений, которые могут эффективно масштабироваться и обеспечивать высокую степень безопасности данных. В этой связи, разработка надежных и производительных бэкенд-систем становится не только актуальной задачей, но и основой для успеха в различных сферах, таких как электронная коммерция, социальные сети, финтех, здравоохранение, образование и другие.

Таким образом, тема разработки серверных решений для современных приложений представляет собой важный и перспективный объект исследования, поскольку она касается всех аспектов функционирования крупных и малых цифровых платформ, а также обеспечивает их конкурентоспособность на рынке.

Обзор литуратуры и существующих решений

Очень важно проанализировать имеющиеся решения чтобы научися на уже известных ошибках.

Discord

Discord — это многофункциональное приложение для обмена сообщениями, которое сочетает в себе текстовые чаты, голосовые и видеозвонки. Первоначально оно было разработано для геймеров, но быстро стало популярным среди разных сообществ благодаря удобству использования, широкому набору функций и высокой производительности.

На данный момент приложение находится в списке блокировок РКН, однако это не повод списывать его со счетов, ведь это действительно очень удобное, функциональное и гениальное приложение и задумка в целом.

Основные особенности Discord

1. Текстовые чаты

- **Каналы**: Серверы организованы в текстовые каналы, где пользователи могут обмениваться сообщениями, ссылками, изображениями и файлами.
- **Темы (Threads)**: Возможность создавать отдельные ветки обсуждений в каналах.
- **Приватные сообщения**: Личное общение между пользователями.

2. Голосовые и видеозвонки

- **Голосовые каналы**: Постоянные комнаты, где пользователи могут подключаться и общаться в реальном времени.
- **Видеозвонки**: Возможность делиться экраном и общаться через видео.
- **Качество связи**: Регулируется в зависимости от пропускной способности и настроек сервера.

3. Серверы

- Серверы представляют собой виртуальные пространства, где собираются сообщества.
- Поддержка большого количества участников (до сотен тысяч).
- Настраиваемая структура каналов и ролей.

4. Интеграции и расширения

- **Боты**: Возможность добавлять автоматизированных помощников для модерации, игр, опросов и других задач.
- **Интеграция с другими платформами**: Twitch, YouTube, Spotify, GitHub, и т.д.

5. Настраиваемость

- Роли и разрешения: Гибкая система прав доступа для участников.
- Темы интерфейса и пользовательские настройки.

6. Безопасность

- Двухфакторная аутентификация (2FA).
- Настройки конфиденциальности для серверов и пользователей.
- Возможность модерации и фильтрации контента.

7. Платформы

- Доступен на Windows, macOS, Linux, iOS, Android и через веб-браузер.

Технологии Discord

1. Бэкенд

- Написан на **Elixir** для обеспечения высокой производительности и масштабируемости.
- Использует **WebRTC** для голосовых и видеозвонков.
- **Redis** и **PostgreSQL** для кэширования и управления данными.

2. Фронтенд

- Основные технологии: **React** и **TypeScript**.
- Удобный интерфейс с акцентом на производительность и UX.

3. Сети

- Discord использует свои собственные сервера для обработки аудио- и видеопотоков.
- **STUN/TURN-серверы** для обеспечения работы через NAT.

Преимущества Discord

1. **Удобство использования**: Интуитивный интерфейс, понятный даже для новых пользователей.
2. **Богатый функционал**: Подходит для общения как в небольших группах, так и в больших сообществах.
3. **Масштабируемость**: Поддержка большого количества пользователей на серверах.
4. **Бесплатность**: Базовые функции доступны бесплатно. Дополнительные возможности (Nitro) предлагаются за подписку.

Недостатки Discord

1. **Высокое потребление ресурсов**: Приложение может быть тяжёлым для старых устройств.
2. **Привязка к серверам Discord**: Нет возможности создать полностью автономные серверы.
3. **Проблемы конфиденциальности**: Некоторые пользователи выражают обеспокоенность сбором данных.
4. **Зависимость от интернет-соединения**: Требуется стабильное подключение для работы.

Итог

Discord — это мощное, универсальное приложение для общения, подходящее как для личного использования, так и для создания сообществ. Оно предоставляет пользователям гибкость в общении через текстовые, голосовые и видеоформаты. Для анализа архитектуры мессенджеров Discord может стать полезным примером благодаря своему акценту на масштабируемости, удобстве использования и интеграции современных технологий.

Telegram

Telegram — это популярный мессенджер с акцентом на скорость, безопасность и богатую функциональность. Он был запущен в 2013 году братьями Павлом и Николаем Дуровыми и быстро завоевал популярность благодаря своей простоте, удобству и инновациям.

Основные особенности Telegram

1. Текстовые сообщения

- **Быстрая отправка**: Оптимизирован для быстрой передачи сообщений даже при низкой скорости интернета.
- **Редактирование и удаление**: Пользователи могут редактировать и удалять свои сообщения.
- **Отправка файлов**: Поддержка отправки файлов больших размеров (до 2 ГБ на файл).

2. Чаты

- **Личные чаты**: Обычные и зашифрованные.
- **Группы**: Поддерживают до 200,000 участников.
- **Супергруппы**: Для больших сообществ с расширенными функциями модерации.
- **Каналы**: Односторонние трансляции сообщений для неограниченного числа подписчиков.

3. Безопасность

- **Секретные чаты**:
    - Шифрование "клиент-клиент" (end-to-end).
    - Сообщения не сохраняются на серверах Telegram.
    - Самоуничтожающиеся сообщения.
- **Регулярные чаты**: Шифруются между клиентом и сервером, но не являются end-to-end.

4. Голосовые и видеозвонки

- Высокое качество связи с использованием собственной технологии компрессии данных.
- Поддержка групповых звонков и видеоконференций.

5. Боты

- Telegram предоставляет мощную платформу для разработки ботов.
- Боты могут выполнять сложные задачи: принимать платежи, запускать мини-игры, обеспечивать поддержку.

6. Кроссплатформенность

- Доступен на Windows, macOS, Linux, iOS, Android и через веб.
- Синхронизация сообщений между устройствами.

7. Настраиваемость

- Темы, обои, звуки уведомлений.
- Гибкие настройки приватности и безопасности.

8. Уникальные функции

- **Голосовые чаты (Spaces)**: Мгновенное общение в реальном времени.
- **Интеграция с публичными сервисами**: Поддержка RSS, интеграция платежей и других внешних систем.
- **Стикеры и GIF**: Поддержка анимированных стикеров и встроенный поиск GIF.

Технологии Telegram

1. Бэкенд

- Telegram использует собственный протокол **MTProto**:
    - Высокая безопасность данных.
    - Оптимизация передачи данных в сетях с высоким уровнем задержек.
- Серверная часть высокопроизводительна, что позволяет обрабатывать огромное количество сообщений.

2. Хранилище

- Телеграм хранит данные пользователей (кроме секретных чатов) на своих серверах, позволяя синхронизацию между устройствами.
- Использует распределённое хранилище для масштабирования.

3. Клиент

- Открытый исходный код для всех клиентских приложений.
- Обеспечивает высокую производительность и безопасность.

Преимущества Telegram

1. **Скорость и масштабируемость**: Работает быстро даже при низкой скорости интернета.
2. **Простота использования**: Интуитивный интерфейс и кроссплатформенность.
3. **Безопасность**: Поддержка end-to-end шифрования в секретных чатах.
4. **Масштабные чаты и каналы**: Возможность охватывать огромную аудиторию.
5. **Разработка ботов**: Гибкость и мощность платформы для автоматизации.

Недостатки Telegram

1. **Конфиденциальность**:
    - Регулярные чаты не используют end-to-end шифрование.
    - Некоторые пользователи критикуют привязку аккаунта к номеру телефона.
2. **Требовательность к хранилищу**:
    - Большое количество медиафайлов может занимать значительное место на устройствах.
3. **Зависимость от серверов Telegram**:
    - Отсутствие возможности полностью автономного использования.

Итог

Telegram — это мощный мессенджер, который сочетает в себе удобство, безопасность и гибкость. Он предоставляет богатый функционал для обычных пользователей и разработчиков. Telegram может быть отличным примером для изучения архитектуры мессенджеров благодаря инновационному подходу к безопасности, быстродействию и масштабируемости.

Итоги анализа существующих решений

Мой мессенджер не претендует на статус инновационного или революционного продукта. Его цель — не изобрести что-то новое, а предоставить возможность глубоко погрузиться в разработку сложных систем, получить ценные практические навыки и понять, как работают современные мессенджеры на архитектурном уровне. Перечислим основные характеристики которым будет обладать мессенджер:

Обмен сообщениями:

- Текстовые чаты: личные и групповые.
- Поддержка редактирования и удаления сообщений.
- Отправка медиафайлов (изображения, видео, документы).

Голосовые и видеозвонки:

- Личные звонки: голосовые и видеосвязь.
- Групповые звонки и конференции.
- Поддержка демонстрации экрана.

Серверная функциональность:

- Возможность развёртывания собственных серверов (аналог Discord).
- Настраиваемая структура серверов: каналы, категории, роли.
- Гибкая система прав доступа для пользователей.

Мгновенные уведомления:

- Push-уведомления для новых сообщений, звонков и событий.

Шифрование:

- End-to-end шифрование личных чатов и звонков.
- Настраиваемые параметры безопасности для серверов.

Кроссплатформенность:

- Доступность на Windows, macOS, Linux, iOS и Android.
- Веб-версия для браузеров.

Интеграции:

- Поддержка ботов и автоматизации.
- Интеграция с внешними сервисами (через API).

Масштабируемость:

- Работа с большим количеством пользователей на сервере.

- Поддержка групп и публичных каналов.

Управление медиа:

- Встроенный просмотр изображений и видео.
- Поддержка файлов большого размера.

Безопасность:

- Аутентификация через токены, пароли или сертификаты.
- Настраиваемая система разрешений для пользователей и администраторов.

Настраиваемость интерфейса:

- Темы, цветовые схемы, настройки уведомлений.

Модерация:

- Инструменты для управления контентом и пользователями на сервере.
- Логи действий модераторов и участников.

Обоснование выбора технологий и средств разработки

Для эффективного выбора необходимо кратко описать существующие технологии, их плюсы и минусы. В основе любой программы или библиотеки лежит язык программировния, поэтому логично будет начать именно с него.

Язык программирования C

C – это мощный язык программирования, который используется в системном программировании, встроенных системах и высокопроизводительных приложениях. Однако для разработки современных бэкенд-приложений, таких как мессенджеры, C имеет как плюсы, так и минусы.

Плюсы использования C для бэкенда:

1. **Высокая производительность**:
    - C компилируется в машинный код, что обеспечивает высокую скорость выполнения.
    - Полный контроль над памятью позволяет оптимизировать производительность.
2. **Минимальные зависимости**:
    - Программы на C не зависят от сторонних фреймворков или библиотек, что делает их компактными и независимыми.
3. **Кроссплатформенность**:
    - Код на C можно компилировать для различных операционных систем.
4. **Гибкость**:
    - Подходит для низкоуровневой работы, включая сетевые протоколы (например, создание собственного WebSocket-сервера или работа с WebRTC).

Минусы использования C для бэкенда:

1. **Отсутствие встроенных библиотек и фреймворков**:
    - В отличие от Python или Node.js, в C отсутствуют готовые решения для работы с HTTP, WebSocket и базами данных. Все придется писать с нуля или использовать сторонние библиотеки.
2. **Сложность разработки**:
    - Управление памятью (malloc/free) требует аккуратности, чтобы избежать утечек памяти.
    - Ошибки, связанные с указателями и буферным переполнением, часто приводят к уязвимостям.
3. **Низкая скорость разработки**:I have already tried to use cargo-local-registry to set up a local registry, apart from all the other means that are explained through the registries wiki.
    - Для создания современных приложений требуется больше времени, чем с языками, имеющими богатые экосистемы (например, Python или Node.js).
4. **Ограниченные возможности масштабирования**:

- Отсутствие встроенных средств для работы с многопоточностью и асинхронностью делает C менее удобным для масштабирования серверов.

5. **Трудности интеграции с современными технологиями**:
    - WebRTC, OAuth, и другие современные технологии сложнее интегрировать в приложении на C.

Заключение

Использование языка C для решения такой задачи нецелесообразно из-за необходимости написания всех необходимых утилит с нуля. К тому же стандартная библиотека и возможности языка C являются очень ограниченными из-за низкоуровневости.

Язык программирования C++

C++ является более современным и мощным языком программирования, чем C, сохраняя при этом его высокую производительность. Он часто используется для разработки сложных систем, включая мессенджеры и мультимедийные приложения. Рассмотрим C++ для создания бэкенда мессенджера.

Плюсы использования C++ для бэкенда:

1. Высокая производительность:

- C++ компилируется в машинный код, обеспечивая высокую скорость выполнения.
- Полный контроль над ресурсами (память, процессор) позволяет оптимизировать приложение.

2. Мощные возможности ООП и шаблонов:

- Поддержка объектно-ориентированного программирования облегчает организацию кода.
- Шаблоны (templates) и STL (Standard Template Library) ускоряют разработку.

3. Современные библиотеки и фреймворки:

- **Boost**: Набор библиотек для работы с потоками, сетями, файловыми системами и др.
- **Qt**: Фреймворк для разработки приложений, включая серверную часть.
- **gRPC**: Фреймворк для работы с RPC (удаленными вызовами) на базе HTTP/2.

4. Асинхронное программирование:

- Библиотеки, такие как ASIO (Boost.Asio), предоставляют мощные инструменты для асинхронной обработки данных, включая WebSocket и TCP/UDP.

5. Поддержка многопоточности:

- Встроенная библиотека std::thread и инструменты для синхронизации облегчают создание многопоточных приложений.

6. Кроссплатформенность:

- C++ позволяет разрабатывать приложения, которые легко переносятся между Windows, Linux и macOS.

7. Подходит для работы с мультимедиа:

- Библиотеки, такие как FFmpeg и OpenCV, упрощают обработку аудио и видео.

Минусы использования C++ для бэкенда:

1. Сложность языка:

- Более крутая кривая обучения по сравнению с Python или JavaScript.
- Необходимость управлять памятью, если не используются умные указатели (smart pointers).

2. Длительное время разработки:

- Код на C++ требует больше времени на написание, тестирование и отладку.

3. Меньше готовых решений для веба:

- В отличие от Python (Django, Flask) или Node.js, C++ имеет меньше специализированных фреймворков для работы с HTTP, WebSocket и REST API.

4. Компиляция:

- Компиляция больших проектов занимает больше времени, чем интерпретация скриптовых языков.

Когда использовать C++ для бэкенда?

Использование C++ оправдано, если:

1. Производительность и эффективность имеют критическое значение (например, для обработки видеопотоков в реальном времени).
2. Вы разрабатываете многопоточные или высоконагруженные системы.
3. Нужен полный контроль над низкоуровневыми аспектами системы.
4. Проект предполагает использование существующих библиотек или фреймворков C++ (Boost, Qt, gRPC).

Заключение

C++ отлично подходит для разработки мессенджера с голосовой и видеосвязью, особенно чтобы интегрировать сложные функции, такие как обработка мультимедиа или взаимодействие с WebRTC. Однако всё же C++ это тоже довольно низкоуровневый язык, и возможно стоит отдать предпочтение другим рассмотренным языкам. C++ будет оптимальным выбором, если проект требует баланса между высокой производительностью и мощной функциональностью.

Язык программирования Rust

Rust предоставляет растущую экосистему высококачественных библиотек (называемых "creates") для работы с базами данных, обмена данными в режиме реального времени (RTC) и API. Несмотря на то, что его экосистема меньше по сравнению с более зрелыми языками, такими как Python или JavaScript, он отличается высокой эффективностью, современностью и разработан с учетом требований безопасности и производительности. Вот краткое описание:

1. Работа с базами данных

Rust имеет множество библиотек для взаимодействия с реляционными и NoSQL базами данных:

Реляционные базы данных:

- **Diesel**: ORM (Object-Relational Mapper) для безопасной работы с SQL. Поддерживает PostgreSQL, MySQL, SQLite.
    - Преимущества: Компиляция SQL-запросов на этапе сборки.
- **SQLx**: Асинхронный ORM, поддерживает PostgreSQL, MySQL, SQLite. Работает без необходимости генерации кода.
    - Преимущества: Удобная интеграция с async/await.

NoSQL базы данных:

- **MongoDB**: Crate mongodb предоставляет полный доступ к MongoDB с асинхронной поддержкой.
- **Redis**: Crate redis-rs для работы с Redis. Поддерживает синхронные и асинхронные операции.

Связанные технологии:

- **SeaORM**: Альтернатива Diesel для асинхронного программирования.
- **SurrealDB**: Rust-сервер базы данных с современными возможностями.

2. Работа с RTC (Real-Time Communication)

Rust идеально подходит для создания высоконагруженных систем реального времени благодаря асинхронным библиотекам и инструментам:

WebRTC:

- **WebRTC Crates**:
    - webrtc: Библиотека для работы с WebRTC, поддерживает P2P соединения для видео и аудио.
    - Полностью открытый исходный код и активно поддерживается.
- **Janus Integration**: Rust может использоваться для интеграции с Janus SFU (Selective Forwarding Unit) через FFI.

Сетевые библиотеки:

- **Tokio**: Асинхронный фреймворк, который идеально подходит для реализации WebSocket-соединений.
- **Quinn**: Реализация протокола QUIC (современная альтернатива TCP), идеально подходящая для RTC.
- **Actix Web**: Быстрый фреймворк для создания API с WebSocket-поддержкой.

3. Работа с API

Rust предоставляет удобные библиотеки для создания и взаимодействия с API:

HTTP-серверы:

- **Actix Web**: Высокопроизводительный фреймворк для создания REST и WebSocket API.
- **Warp**: Легковесный фреймворк с поддержкой асинхронного программирования.

gRPC:

- **Tonic**: Высокопроизводительный фреймворк для работы с gRPC. Поддерживает HTTP/2 и асинхронные вызовы.

HTTP-клиенты:

- **Reqwest**: Легкий и мощный HTTP-клиент с поддержкой async/await.
- **Surf**: Альтернатива Reqwest для простых HTTP-запросов.

Заключение

Rust действительно предлагает богатую экосистему для работы с базами данных, RTC, и API:

1. **Базы данных**: Diesel, SQLx, MongoDB, Redis.
2. **RTC**: WebRTC, Tokio, Quinn, Actix Web.
3. **API**: Actix Web, Warp, Tonic, Reqwest.

Эти библиотеки оптимизированы для производительности и безопасности, что делает Rust отличным выбором для мессенджеров с голосовой и видеосвязью.

Язык программирования Python

Python может быть фундаментальным выбором для разработки мессенджера, особенно если вы фокусируетесь на быстроте разработки, удобстве сопровождения и прототипировании. Однако Python имеет свои плюсы и минусы, которые стоит учитывать.

Плюсы Python как основного языка для мессенджера

1. Быстрота разработки

- Python известен своей простотой и читаемостью кода.
- Большое количество готовых библиотек и фреймворков для различных задач:
    - **FastAPI** или **Flask** для создания REST API.
    - **WebSocket** через библиотеки, такие как websockets или aiohttp.

2. Асинхронное программирование

- Python поддерживает асинхронное программирование с использованием asyncio, что позволяет эффективно обрабатывать множество запросов одновременно.
- Фреймворки, такие как **FastAPI**, идеально подходят для построения асинхронных приложений.

3. Богатая экосистема библиотек

- Работа с базами данных:
  - **SQLAlchemy** для ORM.
  - **asyncpg** для асинхронного взаимодействия с PostgreSQL.
- Работа с WebRTC:
  - Python может использовать сторонние библиотеки или серверы (например, Janus или Pion) через API.
- Обработка данных:
  - Pandas, NumPy — для анализа данных, если нужно.

4. Удобство интеграции

- Python легко интегрируется с другими языками и технологиями:
  - Использование Rust или C++ для высокопроизводительных модулей через **PyO3**, **Cython** или **FFI**.

5. Широкий выбор хостинга и инструментов

- Python поддерживается всеми крупными облачными провайдерами (AWS, GCP, Azure).
- Легко настроить CI/CD через инструменты, такие как GitHub Actions или Jenkins.

6. Комьюнити и поддержка

- Огромное сообщество разработчиков, множество учебных материалов и примеров.

Минусы Python как основного языка для мессенджера

1. Ограничения производительности

- Python интерпретируемый язык, что делает его медленнее компилируемых языков (Rust, C++).
- GIL (Global Interpreter Lock) ограничивает одновременное выполнение потоков, что может стать проблемой при высокой нагрузке.

2. Меньшая поддержка мультимедиа

- Для работы с аудио и видео Python не предлагает такой же глубокой поддержки, как C++ или Rust.
- Работа с WebRTC и обработка медиапотоков часто требует использования внешних серверов или библиотек.

3. Ресурсоёмкость

- Python более требователен к ресурсам, чем языки с низкоуровневым управлением памятью.

Когда Python подходит как фундаментальный язык

- **Прототипирование и быстрый запуск**: Python идеален для быстрого создания минимально жизнеспособного продукта (MVP).
- **API и управление**: Подходит для создания REST API и работы с базой данных.
- **Гибкость**: Если планируется интеграция с высокопроизводительными языками для отдельных частей.

Когда Python недостаточен

- Если мессенджер ориентирован на обработку видео и аудио в реальном времени с низкой задержкой.
- Если ожидается высокая нагрузка (десятки тысяч одновременных подключений).
- Если важна минимальная задержка и максимальная производительность.

Заключение

Python может быть отличным фундаментом для разработки текстового и базового функционала мессенджера. Однако для высоконагруженных задач (RTC, обработка мультимедиа) лучше использовать Python как основу для API и управления, дополняя его более производительными технологиями (например, Rust, C++ или специализированные RTC-серверы).

Заключение по выбору языка программирования

Rust является отличным выбором для реализации мессенджера, если ваша цель — высокопроизводительное, безопасное и современное приложение. Несмотря на крутой порог входа и более сложный процесс разработки по сравнению с Python, Rust обеспечивает:

1. **Максимальную производительность**: благодаря компиляции в машинный код и отсутствию глобальной блокировки интерпретатора (как в Python). Это особенно важно для работы в реальном времени (RTC), где критична минимальная задержка.
2. **Безопасность**: уникальная система владения (ownership) предотвращает утечки памяти и ошибки работы с многопоточностью, что делает приложение надёжным.
3. **Современные инструменты и библиотеки**: Rust предлагает богатую экосистему для работы с базами данных, WebRTC, API и асинхронными задачами (библиотеки, такие как Tokio, Actix Web, SQLx).
4. **Кроссплатформенность**: код на Rust легко компилируется под разные платформы, что упрощает развертывание.

Хотя Python удобен для быстрого прототипирования, его производительность ограничена, особенно при обработке большого количества соединений или мультимедиа.

Использование Rust для всех частей приложения — API и RTC — позволит создать единое, высокопроизводительное решение, минимизируя потребление ресурсов и повышая надёжность.

Таким образом, Rust является оптимальным выбором для разработки мессенджера с учетом современных требований к производительности, безопасности и масштабируемости.

Заключение

Проект мессенджера представляет собой не просто разработку приложения для обмена сообщениями, а полноценное исследование современных технологий и принципов построения коммуникационных систем. Этот мессенджер будет сочетать функциональность и гибкость, предоставляя пользователям мощный инструмент для общения, работы и управления сообществами.

Ключевые аспекты проекта

1. **Практическое значение**:
    - Проект позволит не только создать полезное приложение, но и углубить понимание архитектуры мессенджеров, таких как Telegram и Discord.
    - Опыт работы с шифрованием, масштабируемостью, сетевыми протоколами и пользовательскими интерфейсами станет незаменимым в дальнейшем профессиональном развитии.
2. **Уникальность**:
    - Мессенджер предложит серверную функциональность, аналогичную Discord, позволяя пользователям самостоятельно управлять своими данными.
    - Благодаря встроенной системе шифрования, пользователи будут уверены в приватности своего общения.
3. **Функциональность**:
    - Поддержка текстовых, голосовых и видеосообщений, а также развёртывания серверов обеспечит широкие возможности для использования.
    - Кроссплатформенность и масштабируемость сделают приложение удобным как для личного, так и для профессионального использования.
4. **Образовательный аспект**:
    - Проект станет отличной платформой для изучения сложных технологий, таких как WebRTC, асинхронное программирование, шифрование, и работы с базами данных.